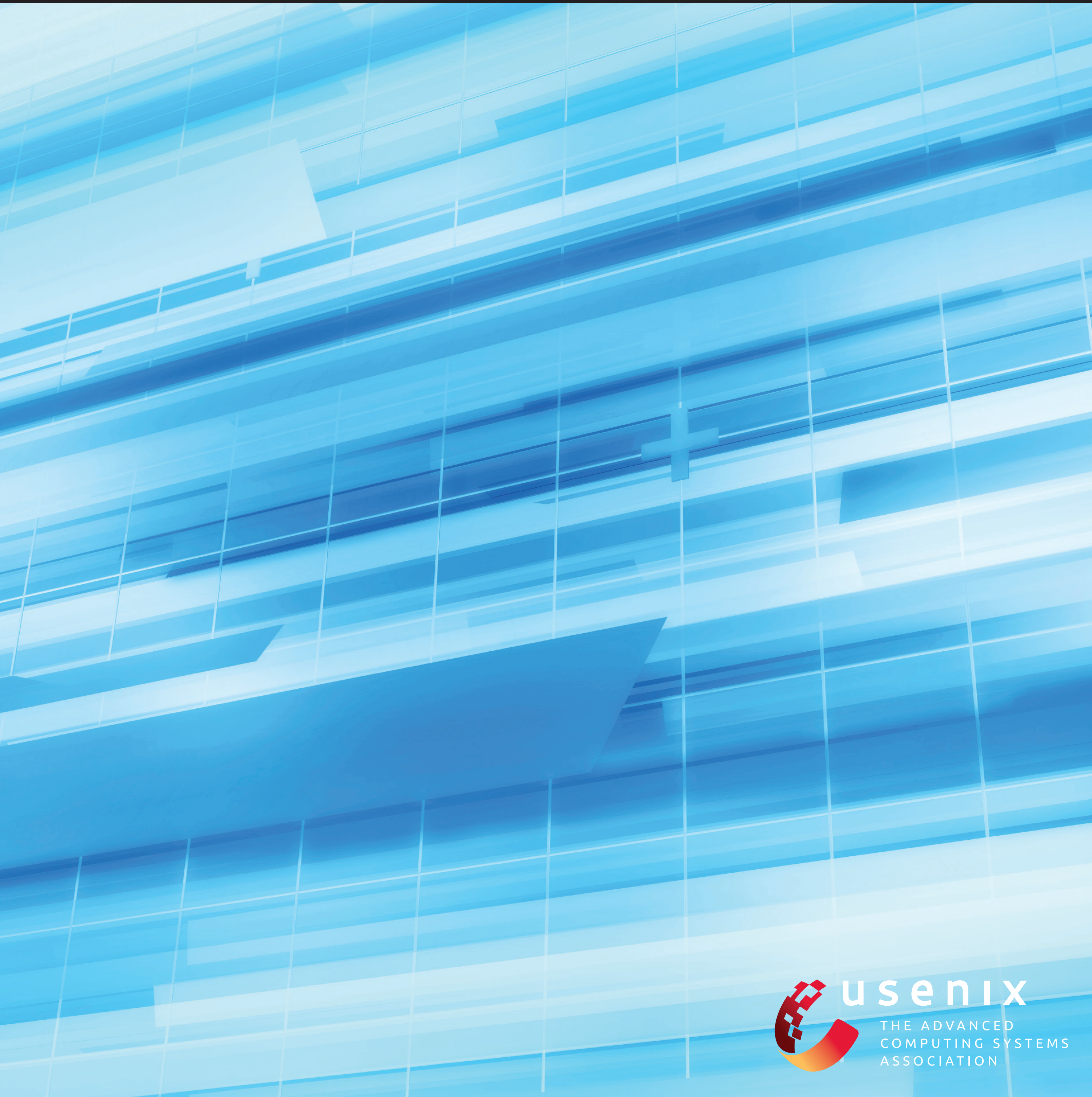
usenix ;login:

DECEMBER 2011 VOL. 36, NO. 6



usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

usenix ASSOCIATION

UPCOMING EVENTS

10th USENIX Conference on File and Storage Technologies (FAST '12)

SPONSORED BY USENIX IN COOPERATION WITH ACM SIGOPS

February 14–17, 2012, San Jose, CA, USA
http://www.usenix.org/fast12

In Cooperation: EuroSys 2012

SPONSORED BY ACM SIGOPS IN COOPERATION WITH USENIX

April 10–13, 2012, Bern, Switzerland
http://eurosys2012.unibe.ch

2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE '12)

CO-LOCATED WITH NSDI '12

April 24, 2012, San Jose, CA, USA
http://www.usenix.org/hotice12
Paper registration due: January 6, 2012

5th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '12)

CO-LOCATED WITH NSDI '12

April 24, 2012, San Jose, CA, USA
http://www.usenix.org/leet12
Submissions due: February 13, 2012

9th USENIX Symposium on Networked Systems Design and Implementation (NSDI '12)

SPONSORED BY USENIX IN COOPERATION WITH ACM SIGCOMM AND ACM SIGOPS

April 25–27, 2012, San Jose, CA, USA
http://www.usenix.org/nsdi12

In Cooperation: 5th Annual International Systems and Storage Conference (SYSTOR 2012)

IN COOPERATION WITH ACM SIGOPS (PENDING) AND USENIX

June 4–6, 2012, Haifa, Israel
http://www.research.ibm.com/haifa/conferences/systor2012

4th USENIX Workshop on Hot Topics in Parallelism (HotPar '12)

SPONSORED BY USENIX IN COOPERATION WITH ACM SIGMETRICS, ACM SIGSOFT, ACM SIGOPS, ACM SIGARCH, AND ACM SIGPLAN

June 7–8, 2012, Berkeley, CA, USA
http://www.usenix.org/hotpar12
Paper registration due: January 24, 2012

2012 USENIX Federated Conferences Week

June 12–15, 2012, Boston, MA, USA

2012 USENIX Annual Technical Conference (USENIX ATC '12)

June 13–15, 2012
http://www.usenix.org/atc12
Paper titles and abstracts due: January 10, 2012

3rd USENIX Conference on Web Application Development (WebApps '12)

June 13–14, 2012
http://www.usenix.org/webapps12
Submissions due: January 23, 2012

4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '12)

4th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '12)

21st USENIX Security Symposium (USENIX Security '12)

August 6–10, 2012, Bellevue, WA, USA

10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)

October 8–10, 2012, Hollywood, CA, USA
http://www.usenix.org/osdi12
Submissions due: May 3, 2012

26th Large Installation System Administration Conference (LISA '12)

December 9–14, 2012, San Diego, CA, USA

FOR A COMPLETE LIST OF ALL USENIX AND USENIX CO-SPONSORED EVENTS,
SEE HTTP://WWW.USENIX.ORG/EVENTS

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Jane-Ellen Long
jel@usenix.org

COPY EDITOR
Steve Gilmartin
proofshop@usenix.org

PRODUCTION
Arnold Gatilao
Casey Henderson
Jane-Ellen Long

TYPESETTER
Star Type
startype@comcast.net

USENIX ASSOCIATION
2560 Ninth Street, Suite 215,
Berkeley, California 94710
Phone: (510) 528-8649
FAX: (510) 548-5738

http://www.usenix.org
http://www.sage.org

;login: is the official magazine of the USENIX Association. *;login:* (ISSN 1044-6397) is published bi-monthly by the USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

$90 of each member's annual dues is for a subscription to *;login:*. Subscriptions for nonmembers are $125 per year. Periodicals postage paid at Berkeley, CA, and additional offices.

POSTMASTER: Send address changes to *;login:*, USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

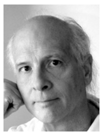
USENIX ;login:

DECEMBER 2011, VOL. 36, NO. 6

OPINION

Musings

RIK FARROW



Rik is the editor of *;login:*.
rik@usenix.org

The latest effort to improve computer and network security has been for the US Congress to pass new laws. Currently proposed laws will increase penalties for attackers, including the perhaps misguided hackers who point out easy-to-find flaws in public servers, to the point of treating them like racketeers. The real culprits, the companies who leave treasure troves of information with real worth easily accessible, will be encouraged to do better. I think there are better solutions.

And for a change, this issue actually includes an article that points the way forward to improving network security. I have whined way too often about how bad things are, giving the appearance that I'm terribly depressed, so I am happy to present something much more useful than more complaints about the state of things.

The Exploit Issue

The first three articles in this issue deal with exploits, the methods used for attacking server and client software. You might not think that reading more about exploits will result in techniques for preventing them, but please give us a moment of your time.

We begin with an article by Ed Schwartz. Ed presented a paper at USENIX Security '11 on Q [1], software that finds short code segments that can be used in one type of exploit. I wanted Ed to write for *;login:* because he did an outstanding job of explaining return-oriented programming (ROP) during his Security presentation. Ed's excellent article starts by explaining how different exploits work, two of the current countermeasures, and why they do not always succeed. He also provides a first-rate discussion of both past and current methods for taking control of the flow of execution, then executing the code of the attacker's choice. As Ed writes, "At a high level, all control flow exploits have two components: a computation and a control hijack. The computation specifies what the attacker wants the exploit to do. For example, the computation might create a shell for the attacker, or create a back-door account. A typical computation is to spawn a shell by including executable machine code called shellcode."

And as Ed has pointed out, the attacker wants to do something that the target software was neither designed nor written to do. Instead, the attacker needs to use existing mechanisms in the software, and its supporting libraries, to do something completely unexpected.

Next, the paper by Sergey Bratus and his co-authors ties in quite beautifully with Ed's article. I first met Sergey Bratus during WOOT '11, where he was explaining how code found in every Linux executable actually includes a complete Turing machine that can be abused [2]. Sergey called this code, and examples like it, *weird machines*, and I find I like this terminology. A weird machine provides an attacker with the ability to execute his own code, completely contrary to the intentions of the software's designers. Yet these weird machines must be present for this to work. And we have hundreds, if not thousands, of existing exploits that prove that these weird machines actually exist.

Sergey also acted as the point person for the third article in this collection. He had to, because its lead author, Len Sassaman, died this summer, before the article was proposed. Len, his wife, Meredith Patterson, and others had been working on a paper that looked at exploits in a different light. What makes exploits work, besides having weird machines to run them on, are the inputs to those weird machines. The authors' proposition was that every program that accepts inputs has its own input language. If that input language reaches beyond a minimal level of complexity, it is impossible to prove that the program that parses the input language will behave as expected. Instead, the program will be one of the weird machines that run exploits for attackers.

The Sassaman proposal has its basis in formal language theory, where a program's input forms the language and the program includes the parser for this language. We are all familiar with this concept, whether we have written programs or simply entered command lines with options. The options make up the input language, and the program must include a parser that interprets that input language. This example may sound too simple, but there have been command-line programs that were exploitable. And network servers have much more complex input languages, with databases supporting SQL perhaps near the pinnacle of complexity. The authors do a fine job of arguing this point.

When Sergey first described this idea to me, in the hallway at Security, I made an immediate connection to an early, and somewhat effective, security prophylactic: application gateways. Application gateways were used, usually as part of firewalls, to parse the input to the services they protected. For example, the application gateway for the SMTP protocol follows the RFC for the protocol exactly (RFC 821, when smapd [3] was written). One result of the slavish adherence to protocol specifications was that crafted inputs from exploits that violated the protocol *in any way* were prevented from reaching the mail server. If the exploit relied on email addresses that were longer than 256 bytes or message lines longer than 1024 bytes, these inputs would be blocked. This is how application gateways could block never-before-seen attacks: they only accepted a very precisely defined input language and rejected all others.

Although application gateways are uncommon today, their close relatives are still in use. Web application firewalls may either enforce a well-defined input language or act more like an Intrusion Prevention System by watching for and blocking known attack signatures. But these were just perspectives that I had when I began talking with Sergey.

Sassaman et al. lay out three points that can be used as a test of their theory's usefulness. They also contrast programming languages, which have input parsers derived from a formal grammar, to most programs, whose input parsers are improvised to support specifications—not nearly as robust a process. And once the input language has grown behind a very simple level of complexity, the parser for the language can become Turing machine complete: itself a complete computer for which the halting problem can never be decided.

If the suggestions in Sassaman et al. were perfectly understood and universally adopted, they would not end exploitation. There would still be bugs in implementation that a simple and provably correct input parser could not protect from exploitation. But the attack surface would be greatly diminished by keeping the input simple, and thus easier for programmers to understand and for designers to specify. And there will always be some input languages (JavaScript and X.509 are examples) which can never be made secure from crafted inputs.

More Security

Let's shift gears a bit, as I describe the next article in the lineup. Adam Langley works for Google as a Chrome developer. I didn't meet him in San Francisco during Security, but I did get to read the summary of the panel on SSL/TLS certificates. Given the recent event where a Dutch certificate authority was compromised [4] and their signing key used to produce bogus and yet totally valid certificates, I thought it would be useful to have someone who could write about this issue.

Adam surprised me by writing about how to properly configure Web servers, when I thought he might be writing about browser insecurity. Yet he is in a very good position to be talking to Web server administrators, as he knows what the browser developers (including Mozilla Firefox) have done or plan to do. Adam is also aware of issues impacting browser users that can be mitigated through the correct use of HTTPS, as well as the proper design of Web pages that will be served over HTTPS.

Adam also discusses the potential for DNSSEC and certificate verification, something that I had really hoped he would do. Paul Vixie's article about DNSSEC in the October 2011 issue had gotten me more interested in the potential for DNSSEC to help with the complex issue of certificate authorities, and Adam presents a clear and direct browser developer's perspective.

Sliding more directly into the intersection of sysadmin and security, Jan Schaumann tells us about a tool he wrote while working at Yahoo! that has since been released as open source. Jan's tool, sigsh, allows the execution of shell scripts that have been signed by authorized users. The signature verifies both the integrity of the script and that the script itself has been authorized for use. Just by maintaining the public key file found on the servers—in this case, many thousands of distributed systems—arbitrary commands can be executed safely for the purposes of system maintenance.

Patrick Debois completes the lineup with an article about DevOps from the sysadmin's perspective. As DevOps is LISA '11's theme, I wanted to understand more about it. And Patrick, as one of the standard bearers for this movement, seemed like just the person to write about DevOps. Patrick does a great job of explaining both the motivation behind DevOps and the goals that can be achieved through its use.

Columns

David Blank-Edelman explains how you can invoke Perl from within your favorite text editor and perform useful, Perl-related tasks. I had often wondered if David just might be a bit obsessive, because of the way he would line everything up *just so* in his code examples. It turns out he wasn't wasting time, but was using Perl modules invoked from within his editor of choice to beautify his code. David also explains other tools you can use. Thanks, David, as I for one am relieved.

Dave Josephsen has found a new suite of system monitoring tools, Graphite, that has got him excited. Graphite is a game changer, writes Dave, and consists of three Python programs: Whisper is a reimplementation of a round-robin data program, Carbon collects data from the network and writes it to Whisper, and Graphite (with the same name as the suite) provides the Web front end. Dave promises to write much more about the elegant solution provided by Graphite in future columns.

Robert Ferrell decided to stick to writing about exploits. Well, sort of. Robert engages in a search for the real meaning of "exploit," and the outcome is as unpredictable as ever.

Elizabeth Zwicky tells us about the books she has been reading, including two that consider what motivates people—in this case, two very distinct groups of people. She also reviews a book about software quality and statistics (another of her favorite topics). Sam Stover, meanwhile, got really excited about the new Kevin Mitnick book. While Mitnick will always be a villain in many people's eyes, Sam loved the way Mitnick and his co-author manage to tell stories with many technical details while taking the reader for a great ride. I don't expect that this book will change anyone's feelings about Mitnick, but it will certainly educate anyone who reads it about the hacking scene in the 1990s, as well as how one of the most accomplished hackers of that era went about learning his craft.

We also have reports from the Security Symposium and many of the co-located workshops. We had a good crew of summarizers, as well as lots of fascinating presentations. I particularly enjoyed the Symposium, which I spent mainly in the Papers track, as well as WOOT and HotSec. But those just represent my own interests.

Besides the summaries, USENIX also provides both recordings and videos of most presentations. These are available online, via the pages for the Symposium and the workshops. Also, as of this August, thanks to the support of you, the members, USENIX expanded its Open Access policy to open all videos to everyone.

I want to leave you with a couple of thoughts. Back in the '90s, when Kevin Mitnick was plying his skills with great success, few people knew, or cared, about computer and network security. That lack of concern helped people like Mitnick simply because people were not expecting to be compromised.

Today, things are a little different. Having watched organizations that should *not* be vulnerable fall to fairly simple attacks, organizations such as security companies, security contractors, defense contractors, and even a certificate authority, expectations are different. People have grown to expect that their systems will be exploited. That is even more true for desktop users, where being able to do whatever the user wants to do, including entertainment, is the norm for both work and home systems.

If we want to have more secure systems, we will need to accept some changes. The Sassaman proposal that appears in this issue provides a concrete step toward creating more secure programs in the future, as well as identifying programs and protocols that can never be made secure. Perhaps the best we can do is to sandbox the unsafe programs as best we can. But that works poorly when the program in question has access to our personal information or defense secrets. In those cases, the Sassaman proposal becomes a much more critical tool for deciding what protocols can be used when security is required.

References

[1] Edward J. Schwartz, Thanassis Avgerinos, and David Brumley, “Q: Exploit Hardening Made Easy,” *Proceedings of the 20th USENIX Security Symposium (USENIX Security ’11)*: http://www.usenix.org/events/sec/tech/full_papers/Schwartz.pdf.

[2] James Oakley and Sergey Bratus, “Exploiting the Hard-Working DWARF: Trojan and Exploit Techniques with No Native Executable Code,” 5th USENIX Workshop on Offensive Technologies: http://www.usenix.org/events/woot11/tech/final_files/Oakley.pdf.

[3] The Firewall Toolkit: http://www.fwtk.org/fwtk/.

[4] Dutch CA compromised: https://www.net-security.org/secworld.php?id=11565.

The Danger of Unrandomized Code

EDWARD J. SCHWARTZ

Ed is currently working on his PhD in computer security at Carnegie Mellon University. His current research interests include operating system defenses and automatic discovery and exploitation of security bugs.

edmcman@cmu.edu

You might not be aware of it, but your operating system is defending you from software attacks: Microsoft Windows, Mac OS X, and Linux all include operating system (OS)–level defenses. In this article, I focus on two modern defenses: address space layout randomization (ASLR) and data execution prevention (DEP). The beauty of these defenses is that they have little runtime overhead and can provide some protection for all programs—even ones with serious vulnerabilities—without requiring access to the program's source code. This flexibility comes with a tradeoff, however: the deployed defenses are not perfect. In this article, I shed some light on how these defenses work, when they work, and how the presence of unrandomized code can allow attackers to bypass them.

To understand the motivation behind ASLR and DEP, we need to understand the exploits they were designed to protect against. Specifically, when we say *exploit* in this article, we are referring to a control flow hijack exploit. Control flow hijack exploits allow an attacker to take control of a program and force it to execute arbitrary code.

At a high level, all control flow exploits have two components: a computation and a control hijack. The computation specifies what the attacker wants the exploit to do. For example, the computation might create a shell for the attacker or create a back-door account. A typical computation is to spawn a shell by including executable machine code called shellcode.

The control hijack component of an exploit stops the program from executing its intended control flow and directs (hijacks) execution to attacker-selected code instead. In a traditional stack-based buffer overflow vulnerability, the attacker overflows a buffer and overwrites control structures such as function pointers or saved return addresses. As long as the attacker knows the address of his shellcode in memory, the attacker can hijack control of the program by overwriting one of these structures with the shellcode's address. In this article, I will call such exploits that use shellcode and a pointer to the shellcode *traditional exploits*. Such an exploit is illustrated in Figure 1.

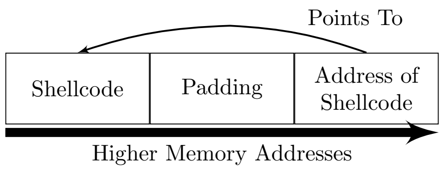


Figure 1: Anatomy of a traditional code injection exploit

Non-Executable Memory

Operating systems have two primary defenses against traditional exploits, called DEP and ASLR. Data execution prevention (DEP) [8] stops an attacker from injecting her own code and then executing it. This effectively prevents the shellcode in traditional exploits from executing successfully. DEP is based on a simple policy: memory should be writable or executable, but never both, at program runtime. To understand the motivation for this policy, consider user input. User input must be written somewhere to memory at runtime, and thus cannot be executable when DEP is enforced. Since shellcode is user input and thus not executable, an exploit utilizing shellcode will crash when DEP is enabled, and the attacker will not be able to execute her computation.

At a high level, DEP sounds like a great defense; it prevents shellcode, and many existing exploits that rely on shellcode will therefore not work. However, one practical limitation is that some programs, such as JIT (just-in-time) compilers, intentionally violate the DEP policy. If DEP is simply enabled for all programs, these programs would stop functioning correctly. As a result, some DEP implementations only protect code modules explicitly marked as DEP-safe.

Another, more fundamental problem is that even when DEP is enabled, *code reuse* attacks can encode a computation in a way that bypasses the defense entirely. The idea is that rather than injecting *new* code into the program, the attacker reuses the executable code that is already there. Because user input is not directly executed, DEP does not prevent the attack.

To perform code reuse attacks, the attacker can find *gadgets*, which are short instruction sequences that perform useful actions. For instance, a gadget might add two registers, or load bytes from memory to a register. The attacker can then chain such gadgets together to perform arbitrary computations. One way of chaining gadgets is to look for instruction sequences that end in `ret`. Recall that `ret` is equivalent to popping the address on the top of the stack and jumping to that address. If the attacker controls the stack, then she can control where the `ret` will jump to. This is best demonstrated with an example.

Assume that the attacker controls the stack and would like to encode a computation that writes `0xdeadbeef` to memory at address `0xcafecafe`. Consider what happens if the attacker finds the gadget `pop %eax; ret` in the executable segment of libc and transfers controls there. The instruction `pop %eax` will pop the current top of the stack—which the attacker controls—and store it in the register %eax. The `ret` instruction will pop the next value from the stack—which the attacker also controls—and jump to the address corresponding to the popped value. Thus, a `ret` instruction allows the attacker to jump to an address of her choosing, and the gadget `pop %eax; ret` allows the attacker to place a value of her choosing in %eax and then jump to an address of her choosing. Similarly, `pop %ebp; ret` allows the attacker to control %ebp and jump somewhere. Finally, if the attacker transfers control to the gadget `movl %eax, (%ebp); ret`, it will move the value in %eax to the memory address specified in %ebp. By executing these three gadgets in that order, the attacker can control the values in %eax and %ebp and then cause the value in %eax to be written to the address in %ebp. In this way the attacker can perform an arbitrary memory write. Figure 2 illustrates a code reuse payload for using these gadgets.

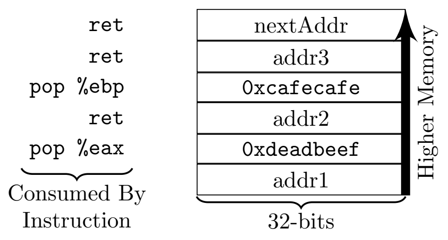


Figure 2: Example of a code reuse payload

Krahmer [3] pioneered the idea of using simple gadgets like these to call functions, which he called the borrowed code chunks technique. Later, Shacham [12] found a set of gadgets in libc that could execute arbitrary programs (or, more formally, arbitrary Turing machines). Shacham's technique is facetiously called return oriented programming (ROP), due to the `ret` at the end of each gadget. Both Krahmer's and Shacham's techniques demonstrate that code reuse can allow very serious attacks when DEP is the only defense enabled.

Randomizing Code Layout

Address space layout randomization (ASLR) [7] is the second primary OS defense. ASLR breaks code reuse attacks by randomizing the location of objects in memory. The idea is that, at some point, exploits need to know something about the layout of memory. For instance, the traditional exploit (shown in Figure 1) uses a pointer to the attacker's shellcode. Similarly, a ROP payload (shown in Figure 2) includes addresses of gadgets in program or library code. If the attacker sets one of these pointers incorrectly, the exploit will fail and the program will probably crash.

It might seem that ASLR completely prevents ROP from succeeding deterministically. If ASLR implementations were perfect, this could be true for some programs. Unfortunately, in practice, ASLR implementations leave at least small amounts of code unrandomized, and this unrandomized code can still be used for ROP.

The details of what gets randomized and when differ for each operating system. On Linux, shared libraries like libc are always randomized, but the program image itself cannot be randomized without incurring a runtime performance penalty. Windows is capable of randomizing both program images and libraries without overhead, but will only do so for modules marked as ASLR-safe when compiled. Although Microsoft's Visual Studio C++ 2010 compiler now marks images as ASLR-safe by default, a great deal of software still contains some modules marked as ASLR-unsafe [9, 6].

These limitations beg the question of why modern operating systems don't randomize all code in memory. On Linux, the reason is because randomizing the program image adds a runtime overhead. Linux programs must be compiled as position-independent executables (PIEs) for the program image to be randomized. On x86 Linux, PIEs run 5–10% slower than non-PIEs [14], but not for x86-64.

Interestingly, the Windows ASLR implementation does not have this problem. The difference is related to how the OSes share code between processes. For instance, if 10 processes of the same program are open, the OS should only have to allocate memory for one copy of the code. One challenge for these shared code mechanisms is that code often needs to refer to objects in memory by their absolute addresses (which might not be known at link time, because of ASLR). Linux uses PIEs to address this problem. PIEs replace each use of an absolute address with a table lookup, which is where the 5–10% overhead comes from. On Windows, the shared code implementation does not require PIEs (and avoids the 5–10% overhead) but typically randomizes each code module only once per boot as a result. Even though Windows can fully randomize program images and libraries with little overhead, it does leave code unrandomized if it is not marked as ASLR-safe, to avoid breaking old third-party programs or libraries which might assume they are always loaded at the same address [6]. This tension between backwards compatibility and security is not just limited to ASLR, either; Windows by default will only protect modules with DEP if they are explicitly marked as safe.

Unrandomized Code

Until fairly recently, it hasn't been clear how dangerous it is to leave small amounts of code unrandomized. Prior work has shown that large unrandomized code bases are very dangerous [12, 3]. For instance, we know we can execute arbitrary programs using libc, which is approximately 1.5 MB. But what can an attacker do with only 20–100 KB of unrandomized code from a program image?

There is already some evidence that even these small amounts of code can lead to successful exploits. Although academic research on ROP has typically focused on Turing completeness, attackers get by in practice with only a few types of gadgets. For instance, rather than encoding their entire computations using ROP, attackers often include shellcode in their exploit but use ROP to disable DEP and transfer control to the shellcode. This can be done by calling the `mprotect` and `VirtualProtect` functions on Linux and Windows, respectively. Many real-world exploits that use these techniques can be found in Metasploit [4], for instance.

Evaluating how applicable these attacks are on large scale is difficult, for two reasons. First, it is intuitively harder to reuse code when there is less code to choose from. This means it will take longer to manually find useful gadgets in the program image of `/bin/true` than it will to find gadgets in libc. Second, the unrandomized code in each program is usually different. This implies that we have to look for different ROP gadgets in every program we would like to exploit. Clearly, constructing ROP attacks by hand for a large number of programs is a tedious and potentially inconsistent task. To evaluate how universal ROP attacks can be, some form of automation is needed. This has motivated some of the joint research with my colleagues, Thanassis Avgerinos and David Brumley.

Specifically, we wanted to know how much unrandomized code is needed for attackers to launch practical attacks such as calling `system('/bin/bash')`. To study this, we built an automated ROP system, Q [11], that is specifically designed to work with small amounts of code, similar to what you might find in a program image. Q takes as input a binary and a target computation and tries to find a sequence of instructions in the binary that is semantically equivalent to the target computation.

The results from our experiments surprised us. Very little code is actually necessary to launch practical ROP attacks. If a function f is linked by the vulnerable program, then Q could create a payload to call f with any arguments in 80% of programs, as long as the program was at least as large as the `/bin/true` command (20 KB). However, attackers often want to call a function g in libc (or another library) that the program did not specifically link. Q was able to create payloads for calling g with any argument in 80% of programs at least as big as `nslookup` (100 KB).

These results are particularly disturbing, because `/bin/true` and `nslookup` are much smaller than the programs often targeted by real attackers.

Other attacks, such as derandomizing libc, can be performed with even greater probability [10]. Derandomizing libc allows the attacker to call functions in libc, but does not necessarily allow the attacker to specify pointer arguments when ASLR randomizes the stack and heap. So, the attacker can call `exit(1)` but not necessarily `system('/bin/bash')`. Even with this restriction, this is a dangerous attack. This type of attack has been shown to be possible in 96% of executables as large as `/bin/true` (20 KB).

Researchers [13] have also noted that the implementations of ASLR on x86 only randomize by 16 bits. This means that an attacker can expect her attack to work after approximately $2^{16} = 65536$ attempts, which is feasible. The suggested fix for this particular problem is to upgrade to a 64-bit architecture.

Defenses

Given that attacks like ROP are so dangerous against ASLR and DEP, it is natural to think of defenses against such attacks. One natural defense against ROP is to disallow unrandomized code in memory. Unfortunately, for performance and backwards compatibility reasons, this is unlikely to happen by default on x86 Windows and Linux. However, Microsoft has released the Enhanced Mitigation Experience Toolkit (EMET) [5], which allows system administrators to force full randomization for selected executables. On Linux, achieving full randomization is possible, but requires recompilation of programs as position-independent executables (PIEs) and incurs a noticeable performance overhead.

A number of other defenses have been proposed to thwart ROP attacks. Many of these defenses are based on the assumption that ROP gadgets always end with a `ret` instruction. Unfortunately, researchers [2] have proven that this assumption is not always true, suggesting that more general defenses, such as control flow integrity (CFI) [1], are needed in practice. Although researchers consider CFI and similar defenses to be low overhead, operating system developers seem unwilling to add defenses with *any* overhead or backwards compatibility problems. Thus, developing a negligible-overhead defense that can prevent ROP without compatibility problems remains an important open problem.

Conclusion

ASLR and DEP are important and useful defenses when used together, but can be undermined when unrandomized code is allowed by the operating system. Recent research [11, 10] has shown that as little as 20 KB of unrandomized code is enough to enable serious attacks when ASLR and DEP are enabled. Unfortunately, modern operating systems currently allow more than 20 KB of unrandomized code, which is unsafe. Until this is remedied, ASLR and DEP are more likely to slow an attacker down than to prevent a reliable exploit from being developed.

References

[1] Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti, "Control-Flow Integrity Principles, Implementations, and Applications," *ACM Transactions on Information and System Security*, vol. 13, no. 1, October 2009.

[2] Stephen Checkoway, Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, Hovav Shacham, and Marcel Winandy, "Return-Oriented Programming without Returns," *Proceedings of the ACM Conference on Computer and Communications Security*, 2010.

[3] Sebastian Krahmer, "x86-64 Buffer Overflow Exploits and the Borrowed Code Chunks Exploitation Technique," 2005: http://www.suse.de/~krahmer/no-nx.pdf.

[4] Metasploit: http://metasploit.org.

[5] Microsoft: Enhanced Mitigation Experience Toolkit v2.1, May 2011: z://www.microsoft.com/download/en/details.aspx?id=1677.

[6] Microsoft, SDL Progress Report, 2011: http://www.microsoft.com/download/en/details.aspx?id=14107.

[7] PaX Team, PaX Address Space Layout Randomization (ASLR): http://pax.grsecurity.net/docs/aslr.txt.

[8] PaX Team., PaX Non-executable Stack (NX): http://pax.grsecurity.net/docs/noexec.txt.

[9] Alin Rad Pop (Secunia Research), "DEP/ASLR Implementation Progress in Popular Third-Party Windows Applications": http://secunia.com/gfx/pdf/DEP_ASLR_2010_paper.pdf, 2010.

[10] Giampaolo Fresi Roglia, Lorenzo Martignoni, Roberto Paleari, and Danilo Bruschi, "Surgically Returning to Randomized lib(c)," *Proceedings of the Annual Computer Security Applications Conference*, 2009, pp. 60–69.

[11] Edward J. Schwartz, Thanassis Avgerinos, and David Brumley, "Q: Exploit Hardening Made Easy," *Proceedings of the USENIX Security Symposium*, 2011.

[12] Hovav Shacham, "The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86)," *Proceedings of the ACM Conference on Computer and Communications Security*, October 2007, pp. 552–561.

[13] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh, "On the Effectiveness of Address-Space Randomization," *Proceedings of the ACM Conference on Computer and Communications Security*, 2004, pp. 298–307.

[14] Ubuntu wiki, Security Features: https://wiki.ubuntu.com/Security/Features?action=recall&rev=52.

Exploit Programming

From Buffer Overflows to “Weird Machines” and Theory of Computation

SERGEY BRATUS, MICHAEL E. LOCASTO, MEREDITH L. PATTERSON, LEN SASSAMAN, AND ANNA SHUBINA

In memory of Len Sassaman, who articulated many of the following observations, connecting the mundane and the deeply theoretical aspects of hacking.

Sergey Bratus is a Research Assistant Professor of Computer Science at Dartmouth College. He sees state-of-the-art hacking as a distinct research and engineering discipline that, although not yet recognized as such, harbors deep insights into the nature of computing. He has a PhD in mathematics from Northeastern University and worked at BBN Technologies on natural language processing research before coming to Dartmouth.
sergey@cs.dartmouth.edu

Michael E. Locasto is an assistant professor in the Computer Science Department at the University of Calgary. He seeks to understand why it seems difficult to build secure, trustworthy systems and how we can get better at it. He graduated magna cum laude from The College of New Jersey (TCNJ) with a BSc degree in computer science. Dr. Locasto also holds an MSc and PhD from Columbia University.
locasto@ucalgary.ca

Meredith L. Patterson is a software engineer at Red Lambda. She developed the first language-theoretic defense against SQL injection in 2005, as a PhD student at the University of Iowa, and has continued expanding the technique ever since. She lives in Brussels, Belgium.
mlp@thesmartpolitenerd.com

Len Sassaman was a PhD student in the COSIC research group at Katholieke Universiteit Leuven. His early work with the Cypherpunks on the Mixmaster anonymous remailer system and the Tor Project helped establish the field of anonymity research, and in 2009 he and Meredith Patterson began formalizing the foundations of language-theoretic security, which he was working on at the time of his death in July 2011. He was 31.

Anna Shubina chose “Privacy” as the topic of her doctoral thesis and was the operator of Dartmouth’s Tor exit node when the Tor network had about 30 nodes total. She is currently a research associate at the Dartmouth Institute for Security, Technology, and Society and manages the CRAWDAD.org repository of traces and data for all kinds of wireless and sensor network research.
ashubina@cs.dartmouth.edu

Hacker-driven exploitation research has developed into a discipline of its own, concerned with practical exploration of how unexpected computational properties arise in actual multi-layered, multi-component computing systems, and of what these systems could and could not compute as a result. The staple of this research is describing unexpected (and unexpectedly powerful) computational models inside targeted systems, which turn a part of the target into a so-called “weird machine” programmable by the attacker via crafted inputs (a.k.a. “exploits”). Exploits came to be understood and written as programs for these “weird machines” and served as *constructive proofs* that a computation considered impossible could actually be performed by the targeted environment.

This research defined and fulfilled the need of such practical exploration in real systems that we must trust. Hacker research has also dominated this area, while academic analysis of the relevant computational phenomena lagged behind.

We show that at its current sophistication and complexity, exploitation research as a discipline has come full circle to the fundamental questions of computability and language theory. Moreover, application of language-theoretic and computation-theoretic methods in it has already borne impressive results, helping to discover and redefine computational models and weaknesses previously overlooked. We believe it is time to bring the hacker craft of finding and programming “weird machines” inside targets and the theorists’ understanding of computational models together for the next step in designing secure, trustworthy computing systems.

The Rise of the Weird Machines

It is hard to say exactly when their story began; chances are that at the beginning they were thought of as just handy tricks to assist more important techniques rather than the essence of exploitation.

The classic "Smashing the Stack for Fun and Profit" by Aleph One [11] manages to explain the conversion of an implicit input data flow into altered program control flow in two short paragraphs:

> So a buffer overflow allows us to change the return address of a function. In this way we can change the flow of execution of the program. …
>
> strcpy() will then copy [the shellcode] onto buffer without doing any bounds checking, and will overflow the return address, overwriting it with the address where our code is now located. Once we reach the end of main and it tried to return it jumps to our code, and execs a shell.

For readers who concentrated on the details of constructing the shellcode (and encountered a hands-on exposition of syscalls and ABI for the first time), it was easy to miss the fact that both the implicit data flow and the subsequent transfer of control were performed by *the program's own code, borrowed by the exploit for its own purposes*. Yet it was this borrowed code, the copying loop of strcpy() and the function's post-amble, that added up to the "remote execution" call as good as any API, into which the shellcode was fed.

This borrowing turned out to be crucial, far more important than the details of shellcode's binary instructions, as Solar Designer showed next year (1997): more of the target's code could be borrowed. In fact, enough code could be borrowed that there was no longer any need to bring *any* of your own executable code to drop a shell—the target process's runtime already conveniently included such code, in libc. One just needed to arrange the overwriting stack data the way that borrowed code expected it, faking a stack frame and giving control to the snippet inside libc's exec().

This was a handy technique for bypassing non-executable stack protections, and it was pigeonholed by many as such. But its real meaning was much deeper: the entire process's runtime address space contents were ripe for borrowing, as long as one spoke the language of implicit data flows (owing to the target's input handling of logic flaws or features) that those borrowed pieces understood.

The borrowings did not need to be a one-off: they could be chained. Quoting all of non-code contents of Tim Newsham's 2000 post that probably holds the record for fewest words per idea value:

> Here's an overflow exploit [for the lpset bug in sol7 x86] that works on a non-exec stack on x86 boxes. It demonstrates how it is possible to thread together several libc calls. I have not seen any other exploits for x86 that have done this. [10]

It was soon generalized to any code snippets present in the target, unconstrained by the code's originally intended function or granularity. Borrowed pieces of code could be strung together, the hijacked control flow linking them powered by their own effects with the right crafted data arranged for each piece. Gerardo (gera) Richarte, presenting this technique, wrote less than half a year later: "Here I present a way to code any program, or almost any program, in a way such that it can be fetched into a buffer overflow in a platform where the stack (and any other place in memory, but libc) is executable" [12].

So exploitation started to look like programming—with *crafted input data* for overflows or other memory corruptions—in really weird assembly-like instructions ("weird instructions") borrowed from the target. Nergal's "Advanced return-into-lib(c) Exploits" [9] described the chaining of faked overflow-delivered stack frames in detail, each borrowed post-amble with its RET instruction bringing the control flow back to the next faked frame, and out into the target's code or libraries, in careful stitches. Also, the granularity of features so stitched can be mixed-and-matched: should the load addresses of the desired snippets be obscured (e.g., with the help of PaX hardening), then why not craft the call to the dynamic linker itself to resolve and even load the symbols, as is its job, let it do its thing, and then go back to snippet-stitching?

It does feel weird to so program with crafted data, but then actual assembled binary code is nothing but data to the CPUs in its fetch-decode-execute cycle, snippets of silicon circuits responsible for performing predictable actions when fed certain formatted inputs, then fetching more inputs. The exploit merely makes a "processor" out of the borrowed target code snippets, which implement the "weird instructions" just as digital logic implements conventional ones.

Altogether, they make up a "weird machine" inside the target on which the crafted-input program executes.

"Weird instructions" can be subtle, multi-step, and spread through the target's execution timeline. The original combination of strcpy() and a RET was a fair example, but just about any interface or library data interpretation code may offer a graceful specimen.

For example, Doug Lea's original memory allocator implementation keeps the freed blocks in a doubly linked list, realized as pointers in chunk headers interspersed with the chunks themselves. A bug in code writing to a heap-allocated buffer may result in a write past the end of the buffer's malloc-ed chunk, overwriting the next chunk's header with our crafted data. When the overwritten chunk is `free`-ed, the allocator's bookkeeping code will then traverse and patch the doubly linked list whose pointers we now control. This gives us a "weird MOV instruction" that takes four overwritten chunk header bytes and writes them where another four bytes are pointing!

This is beautiful, and we can program with it, if only we can cause the overwrite of a freed block and then cause the free() to happen. Such "weird instruction" techniques derived from a combination of an application-specific dynamically allocated buffer overwrite that corrupts the chunk headers and the normal malloc-ed chunk maintenance code are explained in detail in "Vudo malloc tricks" and "Once upon a free()" [7, 1].

Another famous example of a "weird instruction" is provided by the (in)famous printf-family format string vulnerabilities (in which the attacker could control the format string fed to aprintf()). From the computational point of view, any implementation of printf() must contain a parser for the format string, combined with an automaton that retrieves the argument variable's values from the stack and converts them to the appropriate string representations as specified by the `%`-expression. It was not commonly understood, however, that the `%n` specifier in the format string caused that automaton to write the length of the output string printed so far to the stack-fetched address—and therefore the attacker who controlled the format string and the quantity of output could write that length-of-output to some completely unanticipated address! (Even though printf was not passed a proper pointer to such a variable, it would grab whatever was on the stack at the offset that argument would be at, and use *that* as a pointer.)

What unites the printf's handling of the format string argument and an implementation of malloc? The "weird instruction" primitives they supply to exploits. This strange confluence is explained in "Advanced Doug Lea's malloc Exploits" [5], which follows the evolution of the format string-based "4-bytes-write-anything-anywhere" primitive in "Advances in Format String Exploitation" [14] to the malloc-based "almost arbitrary 4 bytes mirrored overwrite," for which the authors adopted a special "weird assembly" mnemonic, `aa4bmo`.

Such primitives enable the writing of complex programs, as explained by Gerardo Richarte's "About Exploits Writing" [13]; Haroon Meer's "The(Nearly) Complete History of Memory Corruption" [8] gives a (nearly) complete timeline of memory corruption bugs used in exploitation.

Remarkably, weird machines can be elicited from quite complex algorithms such as the heap allocator, as Sotirov showed with his "heap feng shui" techniques [16]. The algorithm can be manipulated to place a chunk with a potential memory corruption next to another chunk with the object where corruption is desired. The resulting implicit data flow from the bug to the targeted object would seem "ephemeral" or improbable to the programmer, but can in fact be arranged by a careful sequence of allocation-causing inputs, which help instantiate the "latent" weird machine.

The recent presentation by Thomas Dullien (aka Halvar Flake) [3], subtitled "Programming the 'Weird Machine,' Revisited," links the craft of exploitation at its best with the theoretical models of computation. He confirms the essence of exploit development as "setting up, instantiating, and programming the weird machine."

The language-theoretic approach we discuss later provides a deeper understanding of where to look for "weird instructions" and "weird machines"—but first we'll concentrate on what they are and what they tell about the nature of the target.

Exploitation and the Fundamental Questions of Computing

Computer security's core subjects of study—*trust* and *trustworthiness* in computing systems—involve practical questions such as "What execution paths can programs be trusted to not take under any circumstances, no matter what the inputs?" and "Which properties of inputs can a particular security system verify, and which are beyond its limits?" These ultimately lead to the principal questions of computer science since the times of Church and Turing: "What can a given machine compute?" and "What is computable?"

The old anecdote of the Good Times email virus hoax provides a continually repeated parable of all security knowledge. In the days of ASCII text-only email, the cognoscenti laughed when their newbie friends and relatives forwarded around the hoax warning of the woes to befall whoever reads the fateful message. We *knew* that an ASCII text could not possibly hijack an email client, let alone the rest of the computer. In a few years, however, the laugh was on us, courtesy of Microsoft's push for "e-commerce-friendly" HTMLized email with all sorts of MIME-enabled goodies, including "active" executable code. Suddenly, seriously giving "security" advice to not “open” or “click” emails from “untrusted sources” was a lesser evil, all the sarcastic quotes in this paragraph notwithstanding. So long as our ideas met the computational reality we were dead right, and then those of us who missed the shift were embarrassingly wrong.

Successful exploitation is always evidence of *someone’s* incorrect assumptions about the computational nature of the system—in hindsight, which is 20-20. The challenge of practical security research is to reliably predict, expose, and demonstrate such fallacies for common, everyday computing systems—that is, to develop a methodology for answering or at least exploring the above fundamental questions for these systems. This is what the so-called “attack papers” do.

What “Attack Papers” Are Really About

There is a growing disconnect between the academic and the practitioner sides of computer security research. On the practitioner side, so-called “attack papers”—which academics tend to misunderstand as *merely* documenting attacks on programs and environments—are the bread and butter of practical security (hacker) education, due to their insights into the targets’ actual computational properties and architectures. On the academic side, however, the term “attack paper” has become something of a pejorative, implying a significant intellectual flaw, an incomplete or even marginal contribution.

However, a review of the often-quoted articles from Phrack, Uninformed.org, and similar sources reveals a pattern common to successful papers. These articles describe what amounts to an *execution model and mechanism* that is explicitly or implicitly present in the attacked environment—unbeknownst to most of its users or administrators. This mechanism may arise as an unforeseen consequence of the environment’s design, or due to interactions with other programs and environments, or be inherent in its implementation.

Whatever the reasons, the point of the description is that the environment is capable of executing unforeseen computations (say, giving full shell control to the attacker, merely corrupting some data, or simply crashing) that can be reliably caused by attacker actions—essentially, programmed by the attacker in either the literal or a broader sense (creating the right state in the target, for example, by making it create enough threads or allocate and fill enough memory for a probabilistic exploitation step to succeed).

The attack then comes as a *constructive proof* that such unforeseen computations are indeed possible, and therefore as *evidence* that the target actually includes the described execution model (our use of “proof” and “evidence” aims to be rigorous). The proof is accomplished by presenting a computationally stronger automaton or machine than expected. Exploit programming has been a productive empirical study of these accidental or unanticipated machines and models and of the ways they emerge from bugs, composition, and cross-layer interactions.

Following [2], we distinguish between formal *proofs* and the forms of mathematical reasoning de-facto communicated, discussed, and checked as proofs by the community of practicing mathematicians. The authors observe that a long string of formal deductions is nearly useless for establishing believability in a theorem, no matter how important, until it can be condensed, communicated, and verified by the mathematical community. The authors of [2] extended this community approach to validation of software—which, ironically, the hacker research community approaches rather closely in its modus operandi, as we will explain.

In other words, a hacker research article first describes a collection of the target's artifacts (including features, errors, and bugs) that make the target "programmable" for the attacker. These features serve as an equivalent of elementary instructions, such as assembly instructions, and together make up a "weird machine" somehow embedded in the target environment. The article then demonstrates the attack as a *program* for that machine—and we use the word "machine" here in the sense of a computational model, as in "Turing machine," and similar to "automaton" in "finite automaton."

Accordingly, the most appreciated part of the article is usually the demonstration of how the target's features and bugs can be combined into usable and convenient *programming primitives*, as discussed above. The attack itself comes almost as a natural afterthought to this mechanism description.

Exploits Are Working, Constructive Proofs of the Presence of a "Weird Machine"

It may come as a great surprise to academic security researchers that the practice of exploitation has provided an empirical exploration methodology—with strong formal implications.

For decades, hacker research on exploitation was seen by academia as at best a useful sideshow of vulnerability specimens and ad hoc attack "hacks," but lacking in general models and of limited value to designers of defenses. The process of finding and exploiting vulnerabilities was seen as purely opportunistic; consequently, exploiting was not seen as a source of general insights about software or computing theory.

However, as we mentioned above, a more attentive examination of exploit structure and construction shows that they are *results akin to mathematical proofs* and are used within the community in a similar pattern. Just like proofs, they are checked by peers and studied for technical "tricks" that made them possible; unlike most mathematical proofs, they are runnable, and are in a sense dual to correctness proofs for software such as the *seL4* project.

This proof's *syntactic* expression is typically a sequence of crafted inputs—colloquially known as the "exploit," the same term used for the program/script that delivers these inputs—that reliably cause the target to perform a computation it is deemed incapable of (or does so with no less than a given probability). In some cases—arguably, the most interesting, and certainly enjoying a special place of respect among hackers—these crafted inputs are complemented with physical manipulations of the targeted computing environment, such as irradiating or otherwise "glitching" IC chips, or even controlling the values of the system's analog inputs.

The semantics of the exploit is that of a program for the target's computational environment in its entirety, i.e., the composition of all of its abstraction layers, such as algorithm, protocol, library, OS API, firmware, or hardware. This composition by definition includes any bugs in the implementation of these abstractions, and also any potential interactions between these implementations. The practical trustworthiness of a computer system is naturally a property of the composed object: a secure server that relies on buggy libraries for its input processing is hardly trustworthy.

Constructing “Weird Machines”

From the methodological point of view, the process of constructing an exploit for a platform (common server or application software, an OS component, firmware, or other kind of program) consists of :

1. identifying computational structures in the targeted platform that allow the attacker to affect the target’s internal state via crafted inputs (e.g., by memory corruption);
2. distilling the effects of these structures on these inputs to tractable and isolatable *primitives*;
3. combining crafted inputs and primitives into *programs* to comprehensively manipulate the target computation.

The second and third steps are a well-understood craft, thanks to the historical work we described. The first step, however, requires further understanding.

Halvar Flake [3] speaks of the original platform’s state explosion in the presence of bugs. The exploded set of states is thus the new, *actual* set of states of the target platform.

As much as the “weird machines” are a consequence of this state explosion, they are also defined by the set of reliably triggered transitions between these “weird” states. It is the combination of the two that make up the “weird machine” that is, conceptually, the substrate on which the exploit program runs, and, at the same time, proves the existence of the said “weird machine.” In academic terms, this is what the so-called “malicious computation” runs on.

From the formal language theory perspective, these transitions define the computational structure on this state space that is driven by the totality of the system’s inputs, in turn determining which states are reachable by crafted inputs such as exploit programs. The “weird machine,” then, is simply a concise description of the transition-based computational structures in this exploded space.

In this view, the exploitation primitives we have discussed provide the state transitions that are crucial for the connectivity of the “weird states” graph’s components. In practice, the graph of states is so large that we study only these primitives, but it is the underlying state space that matters.

In a nutshell, it is only a comprehensive exploration of this space and transitions in it that can answer the fundamental question of computing trustworthiness: what the target can and cannot compute. The language-theoretic approach is a tool for study of this space, and it may be the only hope of getting it right.

The Next Step: Security and Computability

The language-theoretic approach and “weird machines” meet at exploitation.

Practical exploration of real-world computing platforms has led to a discipline whose primary product is concise descriptions of unexpected computation models inherent in everyday systems. The existence of such a model is demonstrated by creating an exploit program for the target system in the form of crafted inputs that cause the target to execute it.

This suggests that studying the target's computational behavior on all possible inputs as a language-theoretic phenomenon is the way forward for designing trustworthy systems: those that compute exactly what we believe, and do not compute what we believe they cannot. Starting at the root of the problem, exploits are programs for the *actual* machine—with all its weird machines—presented as input (which is what "crafted" stands for).

This approach was taken by Sassaman and Patterson in their recent research [15, 6]. They demonstrate that computational artifacts (which, in the above terms, make "weird machines") can be found by considering the target's input-processing routines as *recognizers* for the language of all of its valid or expected inputs.

To date, language-theoretic hierarchies of computational power, and the targets' language-theoretic properties, were largely viewed as orthogonal to security, their natural application assumed to be in compilation and programming language design. Sassaman and Patterson's work radically changes this, and demonstrates that theoretical results are a lot more relevant to security than previously thought.

Among the many problems where theory of computation and formal languages meets security, one of paramount importance to practical protocol design is algorithmically checking the *computational equivalence* of parsers for different classes of languages that components of distributed systems use to communicate. Without such equivalence, answering the above questions for distributed or, indeed, any composed systems becomes mired in undecidability right from the start. The key observation is that the problem is decidable up to a level of computational power required to parse the language, and becomes undecidable thereafter—that is, unlikely to yield to any amount of programmer effort.

This provides a mathematical explanation of why we need to rethink the famous "Postel's Principle"—which coincides with Dan Geer's reflections on the historical trend of security issues in Internet protocols [4].

References

[1] "Once upon a free()," *Phrack* 57:9: http://phrack.org/issues.html?issue=57&id=9.

[2] Richard A. DeMillo, Richard J. Lipton, and Alan J. Perlis, "Social Processes and Proofs of Theorems and Programs," technical report, Georgia Institute of Technology, Yale University, 1982: http://www.cs.yale.edu/publications/techreports/tr82.pdf.

[3] Thomas Dullien, "Exploitation and State Machines: Programming the 'Weird Machine,' Revisited," Infiltrate Conference presentation, April 2011: http://www.immunityinc.com/infiltrate/2011/presentations/Fundamentals_of_exploitation_revisited.pdf.

[4] Dan Geer, "Vulnerable Compliance," *;login: The USENIX Magazine*, vol. 35, no. 6, December 2010: http://db.usenix.org/publications/login/2010-12/pdfs/geer.pdf.

[5] jp, "Advanced Doug Lea's malloc Exploits," *Phrack* 61:6: http://phrack.org/issues.html?issue=61&id=6.

[6] Dan Kaminsky, Len Sassaman, and Meredith Patterson, "PKI Layer Cake: New Collision Attacks against the Global X.509 Infrastructure," Black Hat USA, August 2009: http://www.cosic.esat.kuleuven.be/publications/article-1432.pdf.

[7]Michel “MaXX” Kaempf, “Vudo malloc Tricks,” *Phrack* 57:8: http://phrack.org/issues.html?issue=57&id=8.

[8] Haroon Meer, “The (Almost) Complete History of Memory Corruption Attacks,” Black Hat USA, August 2010.

[9] Nergal, “Advanced return-into-lib(c) Exploits: PaX Case Study,” *Phrack* 58:4: http://phrack.org/issues.html?issue=58&id=4.

[10] Tim Newsham, “Non-exec Stack,” *Bugtraq*, May 2000: http://seclists.org/bugtraq/2000/May/90 (pointed out by Dino Dai Zovi).

[11] Aleph One, “Smashing the Stack for Fun and Profit,” *Phrack* 49:14. http://phrack.org/issues.html?issue=49&id=14.

[12] Gerardo Richarte, “Re: Future of Buffer Overflows,” *Bugtraq*, October 2000: http://seclists.org/bugtraq/2000/Nov/32.

[13] Gerardo Richarte, “About Exploits Writing,” Core Security Technologies presentation, 2002.

[14] riq and gera, “Advances in Format String Exploitation,” *Phrack* 59:7: http://phrack.org/issues.html?issue=59&id=7.

[15] Len Sassaman and Meredith L. Patterson, “Exploiting the Forest with Trees,” Black Hat USA, August 2010.

[16] Alexander Sotirov, “Heap Feng Shui in JavaScript,” Black Hat Europe, April 2007: http://www.blackhat.com/presentations/bh-europe-07/Sotirov/Presentation/bh-eu-07-sotirov-apr19.pdf.

The Halting Problems of Network Stack Insecurity

LEN SASSAMAN, MEREDITH L. PATTERSON, SERGEY BRATUS, AND ANNA SHUBINA

Len Sassaman was a PhD student in the COSIC research group at Katholieke Universiteit Leuven. His early work with the Cypherpunks on the Mixmaster anonymous remailer system and the Tor Project helped establish the field of anonymity research, and in 2009 he and Meredith began formalizing the foundations of language-theoretic security, which he was working on at the time of his death in July 2011. He was 31.

Meredith L. Patterson is a software engineer at Red Lambda. She developed the first language-theoretic defense against SQL injection in 2005 as a PhD student at the University of Iowa and has continued expanding the technique ever since. She lives in Brussels, Belgium.
mlp@thesmartpolitenerd.com

Sergey Bratus is a Research Assistant Professor of Computer Science at Dartmouth College. He sees state-of-the-art hacking as a distinct research and engineering discipline that, although not yet recognized as such, harbors deep insights into the nature of computing. He has a PhD in Mathematics from Northeastern University and worked at BBN Technologies on natural language processing research before coming to Dartmouth.
sergey@cs.dartmouth.edu

Anna Shubina chose "Privacy" as the topic of her doctoral thesis and was the operator of Dartmouth's Tor exit node when the Tor network had about 30 nodes total. She is currently a research associate at the Dartmouth Institute for Security, Technology, and Society, and manages the CRAWDAD.org repository of traces and data for all kinds of wireless and sensor network research.
ashubina@cs.dartmouth.edu

Everyday computer insecurity has only gotten worse, even after many years of concerted effort. We must be missing some fundamental yet easily applicable insights into why some designs cannot be secured, how to avoid investing in them and re-creating them, and why some result in less insecurity than others. We posit that by treating valid or expected inputs to programs and network protocol stacks as *input languages that must be simple to parse* we can immensely improve security. We posit that the opposite is also true: *a system whose valid or expected inputs cannot be simply parsed cannot in practice be made secure.*

In this article we demonstrate why we believe this a defining issue and suggest guidelines for designing protocols as secure input languages—and, thus, secure programs. In doing so, we link the formal languages theory with experiences and intuitions of both program exploitation and secure programming.

Indeed, a system's security is largely defined by what computations can and cannot occur in it under all possible inputs. Parts of the system where the input-driven computation occurs are typically meant to act together as a *recognizer* for the inputs' validity (i.e., they are expected to reject bad inputs). Exploitation—an *unexpected* input-driven computation—usually occurs there as well; thinking of it as an input language *recognizer bug* helps find it (as we will show).

Crucially, for complex inputs (input languages) the recognition that matches the programmer's expectations can be equivalent to the "halting problem"—that is, UNDECIDABLE. Then no generic algorithm to establish the inputs' validity is

SIDEBAR

The Chomsky hierarchy ranks languages according to their expressive power in a strict classification of language/grammar/automata classes that establishes a correspondence between language classes, their grammars, and the minimum strength of a computational model required to recognize and parse them.

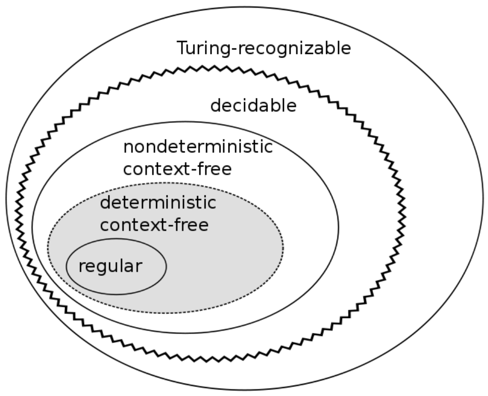
Regular languages, the weakest class of languages, need only a finite state machine and can be parsed with regular expressions.

Unambiguous context-free grammars, the first class that allows some recursively nested data structures, need deterministic pushdown automata (i.e., they require adding a stack to the limited memory of a finite state machine).

Ambiguous context-free grammars need non-deterministic pushdown automata to account for ambiguity.

The more powerful classes of languages, *context-sensitive languages* and *recursively enumerable languages*, require linear bounded automata and Turing machines, respectively, to recognize them. Turing-recognizable languages are UNDECIDABLE. There is a boundary of decidability which it is unwise for an input language or protocol designer to cross, as is discussed in Principle 1 (p. 29, below).

For the regular and deterministic context-free grammars, the *equivalence problem*—do two grammars produce exactly the same language?—is DECIDABLE. For all other classes of grammar, the equivalence problem is UNDECIDABLE, and they should be avoided wherever security relies on computational equivalence of parser implementations, as Principle 2 posits.



possible, no matter how much effort is put into making the input data "safe." In such situations, whatever actual checks the software performs on its inputs at various points are unlikely to correspond to the programmer assumptions of validity or safety at these points or after them. This greatly raises the likelihood of exploitable input handling errors.

A protocol that appears to frustratingly resist efforts to implement it securely (or even to watch it effectively with an IDS) behaves that way, we argue, because its very design puts programmers in the position of unwittingly trying to solve (or approximate a solution to) an UNDECIDABLE problem. Conversely, understanding the flavor of mismatch between the expected and the required (or impossible) recognizer power for the protocol as an input language to a program eases the task of 0-day hunting.

Yet we realize it is all too easy to offer general theories of insecurity without improving anything in practice. We set the following three-pronged practical test as a threshold for a theory's usefulness, and hope to convince the reader that ours passes it. We posit that a theory of insecurity must:

- explain why designs that are known to practitioners as hard to secure are so, by providing a fundamental theoretical reason for this hardness;
- give programmers and architects clear ways to avoid such designs in the future, and prevent them from misinvesting their effort into trying to secure unsecurable systems rather than replacing them;
- significantly facilitate finding insecurity when applied to analysis of existing systems and protocols—that is, either help point out new classes of 0-day vulnerabilities or find previously missed clusters of familiar ones.

As with any attempted concise formulation of a general principle, parts of an up-front formulation may sound similar to some previously mooted pieces of security wisdom; to offset such confusion, we precede the general principles with a number of fundamental examples. We regret that we cannot review the large corpus of formal methods work that relates to various aspects of our discussion; for this, we refer the reader to our upcoming publications (see langsec.org).

The Need for a New Understanding of Computer (In)Security

Just as usefulness of a computing system and its software in particular has become synonymous with it being network-capable, network-accessible, or containing a network stack of its own, we are clearly at an impasse as to how to combine this usefulness with security.

A quote commonly attributed to Einstein is, "The significant problems we face cannot be solved at the same level of thinking we were at when we created them." We possess sophisticated taxonomies of vulnerabilities and, thanks to hacker research publications, intimate knowledge of how they are exploited. We also possess books on how to program securely, defensively, and robustly. Yet for all this accumulated knowledge and effort, insecurity prevails—a sure sign that we are still missing this "next level" in theoretical understanding of how it arises and how to control it.

A Language Theory Look at Exploits and Their Targets

The "common denominator" of insecurity is *unexpected computation* (a.k.a. "malicious computation") reliably caused by *crafted inputs* in the targeted computing environment—to the chagrin of its designers, implementers, and operators. As we point out in [2], this has long been the intuition among hacker researchers, leading them to develop a sophisticated approach that should inform our theoretical next step.

The exploit is really a *program* that executes on a collection of the target's computational artifacts, including bugs such as memory corruptions or regular features borrowed for causing unexpected control or data flows. The view of creating exploits as a kind of macro-assembler programming with such artifacts as "primitives" or macros has firmly established itself (e.g., [3]), the full collection of such artifacts referred to as a "weird machine" within the target. In these terms, "malicious computation" executes on the "weird machine," and, vice versa, the weird machine is what runs the exploit program.

The crucial observation is that the exploit program, whatever else it is, is expressed as *crafted input* and is processed by the target's *input processing routines*. Furthermore, it is these processing routines that either provide the bugs for the weird machine's artifacts or allow crafted input clauses to make their way to such artifacts further inside the target.

Thus a principled way to study crafted input exploit programs and targets in conjunction is to study both the totality of the target's intended or accepted inputs as a language in the sense of the formal language theory, and the input-handling routines as machines that recognize this language. This means, in turn, that evaluating the design of the program's input-handling units and the program itself based on the properties of these languages is indispensable to security analysis.

Throughout this article, we refer to a program's inputs and protocols *interchangeably*, to stress that we view protocols as sequences of inputs, which for every intended or accepted protocol exchange or conversation should be considered as a part of the respective input language. Moreover, we speak of applications' inputs and network stack inputs interchangeably, as both stack layers and applications contain input-handling units that form an important part of the overall system's attack surface.

Let us now apply this general principle to the study of insecurities and finding 0-days in network stacks. We will then explain how it quantifies the hardness of secure design and testing, and helps the designers steer around potentially unsecurable or hard-to-secure designs.

We note that although insecurity obviously does not stop at input handling (which our own examples of composition-based insecurity will illustrate), a provably correct input parser will greatly reduce the reachable attack surface—even though it lacks the magical power to make the system unexploitable. Moreover, the language-theoretic approach applies beyond mere input-parsing, as it sheds light on such questions as, "Can a browser comprehensively block 'unsafe' JavaScript for some reasonable security model and what computational power would be required to do so?" and "Does JSON promote safer Web app development?"

Language-theoretic Attacks on Protocol Parsers

Since the 1960s, programming-language designers have employed automated *parser generators* to translate the unique defining grammar of a machine-parsable language into a parser for that language. Every parser for a given language or protocol is also a *recognizer* for that language: it accepts strings (e.g., binary byte strings) that are valid in its language, and rejects invalid ones—and is therefore a security-crucial component. Although this approach has been of great benefit to compiler and interpreter design, it has largely gone unused with respect to *protocol design and implementation*—at great detriment to security.

Most protocol implementations, in particular network protocol stacks, are still built on essentially *handwritten recognizers*. This leads to implementation errors that introduce security holes or actually accept a broader set of strings than the protocol recognizes. This, in turn, propagates security problems into other implementations that need to accommodate the broken implementation (e.g., several Web servers incorrectly implement TLS/SSL 3.0 in order to interoperate with Internet Explorer [9]).

Furthermore, most approaches to *input validation* also employ handwritten recognizers, at most using regular expressions to whitelist acceptable inputs and/or blacklist potentially malicious ones. Such recognizers, however, are powerless to validate stronger classes of languages allowing for recursively nested data structures, such as context-free languages, which require more powerful recognizers. The sidebar (p. 23, above) describes the Chomsky hierarchy of language grammar classes and their respective recognizer automata classes, by the required computational strength.

This suggests that our language-theoretic approach should reveal clusters of potential 0-days in network stacks, starting at the top, and descending through the middle layers to its very bottom, the PHY layer. Indeed, consider the following examples.

X.509 Parsing

In [8], Kaminsky, Patterson, and Sassaman observed that ASN.1 requires a context-sensitive parser, but the specification of ASN.1 is not written in a way conducive to implementing a parser generator, causing ASN.1 parsers to be handwritten. The parse trees generated by these parsers would thus most likely be different, and their mismatches would indicate potential vulnerabilities.

The authors examined how different ASN.1 parsers handle X.509 documents, focusing on unusual representations of their components, such as Common Name. The results of this examination were numerous vulnerabilities, some of which, when exploited, would allow an attacker to claim a certificate of any site.

Here are just two examples of the many problems with X.509 they discovered using this method:

1. Multiple Common Names in one X.509 Name are handled differently by different implementations. The string `CN=www.badguy.com/CN=www.bank.com/CN=www.bank2.com/CN=*` will pass validation by OpenSSL, which returns only

the first Common Name, but authenticate both www.bank.com and www.bank2.com for Internet Explorer, and authenticate all possible names in Firefox.

2. Null terminators in the middle of an X.509 Name can cause some APIs to see different names than others. In case of the name "www.bank.com00.badguy.com," some APIs would see "badguy.com," but IE's CryptoAPI and Firefox's NSS will see "www.bank.com". Due to NSS's permissive parsing of wildcards, it would also accept a certificate for "*00.badguy.com" for all possible names.

It should be stressed that individual protocol parser vulnerabilities can be found in other ways; for instance, the second item above was independently discovered by Moxie Marlinspike. However, by themselves they may look like "random" bugs and show neither the size of the attack surface nor the systematic nature of the implementers' errors, whereas a language-theoretic analysis reveals the roots of the problem; the difference is that between finding a nugget and striking a gold mine of 0-days.

SQL Parsing and Validation

In [7], Hansen and Patterson discuss SQL injection attacks against database applications. SQL injection attacks have been extremely successful, due to both the complicated syntax of SQL and application developers' habit of sanitizing SQL inputs by using regular expressions to ban undesirable inputs, whereas regular expressions are not powerful enough to validate non-regular languages.

In particular, SQL was context-free until the introduction of the WITH RECURSIVE clause, at which point it became Turing-complete [4] (although in some SQL dialects it may also be possible to concoct a Turing machine using triggers and rules; we are indebted to David Fetter for this observation). Mere regular expressions, which recognize a weaker class of languages, could not validate (i.e., recognize) it even when it was context-free. Turing completeness makes validation hopeless, since recognizing such languages is an undecidable problem. Trying to solve it in all generality is a misinvestment of effort.

The authors suggest that a correct way to protect from SQL injection is to define a safe subset of SQL, which is likely to be a very simple language for any particular application accepting user inputs, and to proceed by generating a parser for that language. This approach offers complete security from SQL injection attacks.

Generalization: Parse Tree Differential Analysis

In [8] Kaminsky, Sassaman, and Patterson further generalized their analysis technique to arbitrary protocols, developing *the parse tree differential attack*, a powerful technique for discovering vulnerabilities in protocol implementations, generating clusters of 0-days, and saving effort from being misinvested into incorrect solutions. This attack compares parse trees corresponding to two different implementations of the same protocol. Any differences in the parse trees indicate potential problems, as they demonstrate the existence of inputs that will be parsed differently by the two implementations.

This method applies everywhere where structured data is marshalled into a string of bytes and passed to another program unit, local or remote. In particular, it should be a required part of security analysis for any distributed system's design. We will discuss its further implications for *secure composition* below.

IDS Evasion and Network Stack Fingerprinting

Differences in protocol parsing at Layers 3 and 4 of TCP/IP network stacks have long been exploited for their fingerprinting (by *Nmap*, *Xprobe*, etc.). Then it was discovered that the impact of these differences on security was much stronger than just enabling reconnaissance: network streams could be crafted in ways that made the NIDS or "smart" firewalls "see" (i.e., have its network stack reassemble) completely different session contents than the targets they protected.

The seminal 1998 paper by Ptacek and Newsham [10] was the first to broach this new research direction. A lot of work followed; for a brief summary see [11]. In retrospect, Ptacek and Newsham's paper was a perfect example of analysis that implicitly treated network protocol stacks' code as protocol recognizers. It also suggested that the target and the NIDS were parts of a *composed* system, and a NIDS's security contribution was ad hoc at best (and negative at worst, for creating a false expectation of security) unless it matched the target in this composition.

Digital Radio Signaling

Recent discovery of overlooked signaling issues as deep as the PHY layer of a broad range of digital radio protocols (802.15.4, Bluetooth, older 802.11, and other popular RF standards) [6] shows another example of a surprisingly vulnerable design that might have gone differently had a language-theoretic approach been applied from the start—and that language-theoretic intuitions have helped to uncover.

The authors demonstrated that the abstraction of PHY layer encapsulation of Link Layer frames in most forms of unencrypted variable-frame-length digital radio can be violated simply by ambient noise. In particular, should the preamble or Start of Frame Delimiter (SFD) be damaged, the "internal" bytes of a frame (belonging to a crafted higher layer protocol payload) can be received by local radios as a PHY layer frame. This essentially enables remote attackers who can affect payloads at Layer 3 and above on the local RF to inject malicious frames without ever owning a radio.

This is certainly not what most Layer 2 and above protocol engineers expect of these PHY layer implementations. From the language recognizer standpoint, however, it is obvious that the simple finite automaton used to match the SFD in the stream of radio symbols and so distinguish between the noise, signaling, and payloads can be easily tricked into "recognizing" signaling as data and vice versa.

Defensive Recognizers and Protocols

To complete our outlook, we must point to several successful examples of program and protocol design that we see as proceeding from and fulfilling related intuitions.

The most recent and effective example of software specifically designed to address the security risks of an input language in common Internet use is Blitzableiter by Felix 'FX' Lindner and Recurity Labs [12, 13]. It takes on the task of *safely* recognizing Flash, arguably the most complex input language in common Internet use, due to two versions of bytecode allowed for backward compatibility and the complex SWF file format; predictably, Flash is a top exploitation vector with continually surfacing vulnerabilities. Blitzableiter (a pun on lightning rod) is an armored recognizer for Flash, engineered to maximally suppress implicit data and control flows that help turn ordinary Flash parsers into "weird machines."

Another interesting example is the observations by D.J. Bernstein on the 10 years of qmail [13]. We find several momentous insights in these, in particular *avoiding parsing* (i.e., in our terms, dealing with non-trivial input languages) whenever possible as a way of making progress in eliminating insecurity, and pointing to handcrafting input-handling code for efficiency as a dangerous distraction. In addition, Bernstein stresses using UNIX context isolation primitives as a way of enforcing *explicit data flows* (in our terms, hobbling construction of "weird machines"). Interestingly, Bernstein also names the Least Privilege Principle—as currently understood—as a distraction; we argue that this principle needs to be updated rather than discarded, and we see Bernstein's insights as being actually in line with our proposed update (see below).

There are also multiple examples of protocols designed with easy and unambiguous parsing in mind. Lacking space for a comprehensive review of the protocol design space, we point the reader to our upcoming publication, and only list a few examples here:

- The ATM packet format is a regular language, the class of input languages parsable with a finite-state machine, easiest to parse, which helps avoid signaling attacks as discussed above. The same is true for other fixed-length formats.
- JSON is arguably the closest to our recommendation for a higher-layer language for encoding and exchanging complex, recursive objects between parts of a distributed program. Such a language needs to be context-free (the classic example of this class is S-expressions), but not stronger.

Language-theoretic Principles of Secure Design

Decidability matters. Formally speaking, a correct protocol implementation is defined by the decision problem of whether the byte string received by the stack's input handling units is a member of the protocol's language. This problem has two components: first, whether the input is syntactically valid according to the grammar that specifies the protocol, and second, whether the input, once recognized, generates a valid state transition in the state machine that represents the logic of the protocol. The first component corresponds to the parser and the second to the remainder of the implementation.

The difficulty of this problem is directly defined by the class of languages to which the protocol belongs. Good protocol designers don't let their protocols grow up to be Turing-complete, because then the decision problem is UNDECIDABLE.

In practice, undecidability suggests that *no amount of programmer or QA effort is likely to expose a comprehensive selection of the protocol's exploitable vulnerabilities related to incorrect input data validity assumptions.* Indeed, if no generic algorithm to establish input validity is possible, then whatever actual validity checks the software performs on its inputs at various points are unlikely to correspond to the programmer's assumptions of such validity. Inasmuch as the target's potential vulnerability set is created by such incorrect assumptions, it is likely to be large and non-trivial to explore and prune.

From malicious computation as the basis of the threat model and the language-theoretic understanding of inputs as languages, several bedrock security principles follow:

Principle 1: Starve the Turing Beast—Request and Grant Minimal Computational Power

Computational power is an important and heretofore neglected dimension of the attack surface. Avoid exposing unnecessary computational power to the attacker.

An input language should only be as computationally complex as absolutely needed, so that the computational power of the parser necessary for it can be minimized. For example, if recursive data structures are not needed, they should not be specified in the input language.

The parser should be no more computationally powerful than it needs to be. For example, if the input language is context-free, then the parser should be no more powerful than a deterministic pushdown automaton.

For Internet engineers, this principle can be expressed as follows:

- a parser must not provide more than the minimal computational strength necessary to interpret the protocol it is intended to parse;
- protocols should be designed to require the computationally weakest parser necessary to achieve the intended operation.

An implementation of a protocol that exceeds the computational requirements for parsing that protocol's inputs should be considered broken.

Protocol designers should design their protocols to be as weak as possible. Any increase in computational strength of input should be regarded as a grant of additional privilege, thus increasing security risk. Such increases should therefore be entered into reluctantly, with eyes open, and should be considered as part of a formal risk assessment. At the very least, the designer should be guided by the Chomsky hierarchy (described in the sidebar, p. 23).

Input-handling parts of most programs are essentially Turing machines, whether this level of computational power is needed or not. From the previously discussed *malicious computation* perspective of exploitation it follows that this delivers the full power of a Turing-complete environment into the hands of the attacker, who finds a way of leveraging it through crafted inputs.

Viewed from the venerable perspective of Least Privilege, Principle 1 states that *computational power is privilege, and should be given as sparingly as any other kind of privilege to reduce the attack surface.* We call this extension the *Minimal Computational Power Principle.*

We note that recent developments in common protocols run contrary to these principles. In our opinion, this heralds a bumpy road ahead. In particular, HTML5 is Turing-complete, whereas HTML4 was not.

Principle 2: Secure Composition Requires Parser Computational Equivalence

Composition is and will remain the principal tool of software engineering. Any principle that aims to address software insecurity must pass the test of being applicable to practical software composition, lest it forever remain merely theory. In particular, it should specify how to maintain security in the face of (inevitable) composition—including, but not limited to, distributed systems, use of libraries, and lower-layer APIs.

From our language-theoretic point of view, any composition that involves converting data structures to streams of bytes and back for communications between components necessarily relies for its security on the different components of the system performing *equivalent* computations on the input languages.

However, computational equivalence of automata/machines accepting a language is a highly non-trivial language-theoretic problem that becomes UNDECIDABLE starting from non-deterministic context-free languages (cf. the sidebar for the decidability of the equivalence problem).

The X.509 example above shows that this problem is directly related to insecurity of distributed systems' tasks. Moreover, undecidability essentially precludes construction of efficient code testing and/or verification algorithmic techniques and tools.

On the Relevance of Postel's Law

This leads to a re-evaluation of Postel's Law and puts Dan Geer's observations in "Vulnerable Compliance" [5] in solid theoretical perspective.

Postel's *Robustness Principle* (RFC 793), best known today as *Postel's Law*, laid the foundation for an interoperable Internet ecosystem. In his specification of TCP, Postel advises to "be conservative in what you do, be liberal in what you accept from others." Despite being a description of the principle followed by TCP, this advice became widely accepted in IETF and general Internet and software engineering communities as a core principle of protocol implementation.

However, this policy maximizes interoperability at the unfortunate expense of consistent parser behavior, and thus at the expense of security.

Why Secure Composition Is Hard

The second principle provides a powerful theoretical example of why *composition*—the developer's and engineer's primary strategy against complexity—is hard to do securely. Specifically, a composition of communicating program units must rely on *computational equivalence* of its input-handling routines for security (or even correctness when defined); yet such equivalence is UNDECIDABLE for complex protocols (starting with those needing a nondeterministic pushdown automaton as a recognizer of their input language), and therefore cannot in practice be checked even for differing implementations of the same communication logic.

Conversely, this suggests a principled approach for reducing insecurity of composition: keep the language of the messages exchanged by the components of a system to a necessary minimum of computational power required for their recognition.

Parallels with Least Privilege Principle

The understanding of "malicious computation" programmed by crafted inputs on the "weird machine" made of a target's artifacts as a threat naturally complements and extends the Least Privilege Principle as a means of containing the attacker. In particular, just as the attacker should not be able to spread the compromise beyond the vulnerable unit or module, he should not be able to propagate it beyond the minimal computational power needed. This would curtail his ability to perform malicious computations.

Thus the Least Privilege Principle should be complemented by the Minimal Computational Power Principle. This approach should be followed all the way from the application protocol to hardware. In fact, we envision *hardware* that limits itself from its current Turing machine form to weaker computational models according to the protocol parsing tasks it must perform, lending no more power to the parsing task than the corresponding language class requires—and therefore no more power for the attacker to borrow for exploit programs in case of accidental exposure, starving the potential "weird machines" of such borrowed power. This restriction can be accomplished by reprogramming the FPGA to only provide the appropriate computational model—say, finite automaton or a pushdown automaton—to the task, with appropriate hardware-configured and enforced isolation of this environment from others (cf. [1]).

Conclusion

Computer security is often portrayed as a never-ending arms race between attackers seeking to exploit weak points in software and defenders scrambling to defend regions of an ever-shifting battlefield. We hold that the front line is, instead, a bright one: the system's security is defined by what computations can and cannot occur in it under all possible inputs. To approach security, the system must be analyzed as a recognizer for the language of its valid inputs, which must be clearly defined by designers and understood by developers.

The computational power required to recognize the system's valid input language(s) must be kept at a minimum when designing protocols. This will serve to both reduce the power the attacker will be able to borrow, and help to check that handling of structured data across the system's communicating components is *computationally equivalent*. The lack of such equivalence is a core cause of insecurity in network stacks and in other composed and distributed systems; undecidability of checking such equivalence for computationally demanding (or ambiguously specified) protocols is what makes securing composed systems hard or impossible in both theory and practice.

We state simple and understandable but theoretically fundamental principles that could make protection from unexpected computations a reality, if followed in the design of protocols and systems. Furthermore, we suggest that in future designs hardware protections should be put in place to control and prevent exposure of unnecessary computational power to attackers.

References

[1] Sergey Bratus, Michael E. Locasto, Ashwin Ramaswamy, and Sean W. Smith, "New Directions for Hardware-Assisted Trusted Computing Policies" (position paper), 2008.

[2] Sergey Bratus, Michael Locasto, Meredith L. Patterson, Len Sassaman, and Anna Shubina, "Exploit Programming: From Buffer Overflows to Theory of Computation," in preparation.

[3] Thomas Dullien, "Exploitation and State Machines: Programming the 'Weird Machine,' Revisited," Infiltrate Conference, April 2011: http://www.immunityinc.com/infiltrate/presentations/Fundamentals_of_exploitation_revisited.pdf.

[4] David Fetter, "Lists and Recursion and Trees, Oh My!" OSCON, 2009.

[5] Dan Geer, “Vulnerable Compliance,” *;login:,* vol. 35, no. 6 (December 2010): http://www.usenix.org/publications/login/2010-12/pdfs/geer.pdf.

[6] Travis Goodspeed, Sergey Bratus, Ricky Melgares, Rebecca Shapiro, and Ryan Speers, “Packets in Packets: Orson Welles’ In-Band Signaling Attacks for Modern Radios,” 5th USENIX Workshop on Offensive Technologies, August 2011: http://www.usenix.org/events/woot11/tech/final_files/Goodspeed.pdf.

[7] Robert J. Hansen and Meredith L. Patterson, “Guns and Butter: Towards Formal Axioms of Input Validation,” Black Hat USA, August 2005: http://www.blackhat.com/presentations/bh-usa-05/BH_US_05-Hansen-Patterson/HP2005.pdf.

[8] Dan Kaminsky, Len Sassaman, and Meredith Patterson, “PKI Layer Cake: New Collision Attacks against the Global X.509 Infrastructure,” Black Hat USA, August 2009: http://www.cosic.esat.kuleuven.be/publications/article-1432.pdf.

[9] Katsuhiko Momoi, “Notes on TLS-SSL 3.0 Intolerant Servers”: http://developer.mozilla.org/en/docs/Notes_on_TLS_-_SSL_3.0_Intolerant_Servers, 2003.

[10] Thomas H. Ptacek and Timothy N. Newsham, “Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection, “ technical report, Secure Networks, Inc., January 1998: http://insecure.org/stf/secnet_ids/secnet_ids.html.

[11] Sumit Siddharth, “Evading NIDS, Revisited”: http://www.symantec.com/connect/articles/evading-nids-revisited.

[12] Stefan Krempl, “Protection against Flash Security Holes,” December 30, 2009: http://www.h-online.com/security/news/item/26C3-Protection-against-Flash-security-holes-893689.html.

[13] Felix “FX” Lindner, “The Compromised Observer Effect,” *McAfee Security Journal*, 6 (2010): 16–19.

[13] Daniel J. Bernstein, “Some Thoughts on Security after 10 Years of qmail 1.0,” November 1, 2007: cr.yp.to/qmail/qmailsec-20071101.pdf.

Beyond the Basics of HTTPS Serving

ADAM LANGLEY

Adam Langley is a software engineer working on Google Chrome.
agl@chromium.org

A number of factors are contributing to an increase in the number of Web sites that choose to deploy HTTPS. Tools such as Firesheep [1] have highlighted the vulnerability of the transport layer now that hosts are often connected via public, wireless networks; at the much larger scale, there have been several instances of national-level actors performing countrywide attacks [2, 3].

As the incentives to use HTTPS have increased, the costs have decreased. CPU time spent on SSL is a diminishing fraction for increasingly complex Web sites, even as processing power becomes cheaper.

None of these forces seem likely to change, so running an HTTPS site is increasingly likely to be part of your job in the future, if it isn't already.

The Stripping Problem

When entering a URL into a browser's address bar, few users bother to enter any scheme part at all. Since the default scheme is HTTP, they are implicitly requesting an insecure connection. Although sites which use HTTPS pervasively will immediately redirect them to an HTTPS URL, this gap is all an attacker needs. By stopping the redirect, an attacker can proxy the real site over HTTP and effectively bypass HTTPS entirely. Very observant users may notice the lack of security indications in their browser, but this is generally a very effective attack that is commonly known as SSL stripping, after the demonstration tool by Moxie Marlinspike of the same name [4].

HTTP Strict Transport Security (HSTS) [5] allows a Web site to opt in to being HTTPS only. For an HSTS site, a browser will only send HTTPS requests, eliminating the window of insecurity. HSTS is currently an IETF draft, but has already been implemented by both Chrome and Firefox.

Web sites opt into HSTS by means of an HTTP header, such as:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

HSTS headers are only accepted over HTTPS to stop denial-of-service attacks. The HSTS property is cached for the given number of seconds (about a year in this example) and, optionally, also for all subdomains of the current domain. (Including subdomains is highly recommended, as it stops an attacker from directing a user to a subdomain over HTTP and capturing any domain-wide, insecure cookies.)

HSTS also makes certificate errors fatal for the site in question. As the Web has penetrated into everyday life, users are now ordinary people, and asking them to evaluate complex certificate validity questions is ridiculous. Fatal errors do, of course, make it critical that you renew certificates in a timely fashion, but remaining certificate validity should be part of any monitoring setup. It's also important that the names in your certificate match what you actually use. Remember that a wildcard matches exactly one label. So *.example.com doesn't match example.com or foo.bar.example.com. Usually you want to request a certificate from your CA that includes example.com and *.example.com.

As HSTS properties are learned on the first visit, there's still a gap before a user has visited a site for the first time where HSTS isn't in effect. This gap is also a problem after the user has cleared browsing history. In order to counter this, Chrome has a built-in list of HSTS sites which is always in effect. High-security sites are invited to contact the author in order to be included. Over time this list may grow unwieldy, but that's a problem that we would love to have. At the moment the list is just over 60 entries, although that does include sites such as Gmail, Paypal, and Twitter.

Mixed Scripting

Mixed scripting is a subset of the more general problem of mixed content. Mixed content arises whenever an HTTPS origin loads resources over an insecure transport. Since the insecure resources cannot be trusted, the security of the whole page is called into question.

The impact of a mixed content error depends on the importance of the resource in question. In the case of an insecure image, the attacker can only control that image. However, in the case of JavaScript, CSS, and embedded objects, the attacker can use them to take control of the whole page and, due to the same-origin policy, any page in that origin. When these types of resources are insecure, we call it mixed scripting because of the greatly increased severity of the problem.

Browsers typically inform the user of mixed scripting in some fashion, either by removing the security indicators that usually come with HTTPS, or by highlighting the insecurity itself. Chrome will track the problem across pages within the same origin, which yields a more accurate indication of which pages are untrustworthy, even though this has confused some developers.

But years of nasty warnings have failed to solve the problem, and mixed scripting is an insidious threat for sites that mix HTTP and HTTPS. Such sites will often have HTTP pages that are not expected to be served over HTTPS, but which are accessible as such. Since no normal interaction will lead to the HTTPS version, no warnings will ever appear. But with mixed scripting the attacker gets to choose the location. By injecting an iFrame or a redirect into any HTTP request (to any site), an attacker can cause a browser to load a given HTTPS page. By picking a page with mixed scripting, they can then hijack the insecure requests for resources and compromise the origin.

The real solution is for browsers to block mixed script loads. Internet Explorer version 9 already does this and Chrome will soon, so it's past time to pay attention to any mixed script warnings before your site breaks. But for other browsers, which will still be dominant for many years to come, you have to make sure that there's no mixed scripting on any of your pages, by actively searching for them in the same way that an attacker would (although some respite may come in the form of CSP, which we'll consider next).

Content Security Policy

I will only briefly reference CSP [6] here, as it's a large topic and one that strays outside of transport security. However, it is being implemented in both WebKit and Firefox, and it allows a site, by means of another HTTP header, to limit the origins from which active resources can be sourced. By using it to eliminate any non-HTTPS origins, mixed scripting can be solved for the set of browsers that implement CSP but don't block by default.

CSP also has the ability to send reports back when it encounters a violation of the policy. By monitoring these reports, a site can discover mixed-scripting errors that real-world users are experiencing.

Secure Cookies

For sites that aren't HSTS, it's important that they mark any sensitive cookies as "secure." By default, the same origin policy for cookies does not consider port or protocol [11]. So any insecure requests to the same domain will contain the full set of cookies in the clear, where an attacker can capture them. Without HSTS, users are likely to make HTTP requests even to HTTPS-only sites when they enter a domain name to navigate.

Cookies marked as secure will only be sent over HTTPS. Even if your site is using HSTS, with all subdomains included, you should still mark your sensitive cookies as secure, simply as a matter of good practice.

DNSSEC and Certificate Verification

Once you have all of the above sorted out, you may want to worry about how certificate verification works. The number of root certificate authorities trusted by Windows, OS X, or Firefox may provoke worry in some quarters, but, due to an unfortunate legacy, the situation is rather worse.

Normal certificates, used by sites for HTTPS, are end-entity certificates. They contain a public key which is used to provide transport security for a host, and that public key can't be used to sign other certificates. But there are a number of cases where organizations want an intermediate CA certificate, one that can be used to sign other certificates. There do exist mechanisms for limiting the power of intermediate CA certificates, but they are far from universally supported by clients, which will often reject such certificates. This leads to intermediate CA certificates typically being issued with the full signing power of the root CA that issues them.

Although the root CA is technically responsible for them, the identities of intermediate CA holders are not public, nor are the handling requirements that the root CA imposes on them. Thanks to stellar work by the EFF [7], which crawled the Web for the subset of intermediate CAs that have issued public Web site certificates, we know that there are at least 1,482 such certificates held by an estimated 651 different organizations.

This has prompted questions about the foundations of HTTPS and several efforts to address the problem. Probably the most prominent area of focus for solutions involves using DNSSEC, either as an alternative or as a constraint.

As a PKI, DNSSEC has much to commend it. Although it's certainly not simple, it is much less complex than PKIX (the standard for existing certificates). It inherently solves the intermediate CA problem, because of DNS's hierarchical nature. It may also encourage the use of HTTPS, because Web sites must already have a relationship with a DNS registrar, who can also provide DNSSEC.

Most existing certificates are based on proving ownership of a DNS name, often via email. These are called Domain Validation (DV) certificates and they fundamentally rest on DNS. A compromise of your DNS can easily be turned into a DV certificate via an existing certificate authority, so DNSSEC has a good claim to be at least as strong as DV certificates. (Extended Validation (EV) certificates are significantly more rigorous and are not candidates for any of the measures described here.)

With DNSSEC in hand, there are two broad categories of statements that we might want to make about certificates. We might wish to exclude existing certificates ("This site's CA is X, accept nothing else"), or we might wish to authorize certificates that wouldn't otherwise be valid ("X is a valid public key for this site"). On another axis, we might want these statements to be processed by clients or by CAs.

In the "exclusion" and "processed by CAs" corner, we have CA Authorization (CAA) records [8]. These DNS records are designed to inform a CA whether they are authorized to issue a certificate in a given domain. Although CAs will check DNS ownership in any case, this provides a useful second line of defense, and, since there aren't many root CAs that issue to the general public, deployment is a tangible goal in the medium term. Additionally, since DNSSEC is signature based, CAs can retain the DNSSEC chain that they resolved as a proof of correctness in the event of a dispute.

Covering both types of statements designed to be processed by clients is DANE [9], and this is where much of the work is occurring. Although DANE is by no means designed exclusively for browsers, browsers are the dominant HTTPS client on the Internet at the moment and much of the discussion tends to start with them.

On that basis, it's worth considering some of the headwinds that any DNSSEC solution designed for browsers to implement will face.

First, DNSSEC resolution ability on the client is almost non-existent at the moment. Without this, browsers would be forced to ship their own DNSSEC resolver libraries, which is a degree of complexity and bloat that we would really rather avoid. Even assuming that the client is capable of resolution, DNS is probably the most adulterated protocol on the Internet; it seems that every cheap firewall and hotel network abuses and filters it. In an experiment conducted by a large population of consenting Chrome users, around 0.5–1% of DNS requests for a random DNS resource record type (13172) failed to get any reply, even after retries, for domain names that were known to exist. Based on this, any DNSSEC resolution will have to assume a significant amount of filtering and misbehavior of the network when faced with DNSSEC record types.

These troubles suggest that any certificate exclusion will have a very troubled deployment as, in order to be effective, exclusion has to block on getting a secure answer. An exclusion scheme that can be defeated by filtering DNS requests is ineffective. Even DNSSEC-based certificate authorization would be unreliable and frustrating.

Setting aside the functionality problems for the moment, performance is also a concern. In the same experiment described above, 2.5% of the replies that were received arrived over 100 ms after the browser had set up a TCP connection to, performed a TLS handshake with, and verified the certificate of the same domain name. For authorization, the site bears the performance impact and so that, at least, is tenable. For exclusion, with its blocking lookup, these penalties would be imposed on every site.

Although DNSSEC resolution on the client would appear to face significant hurdles, there are still options. First, for exclusion we could sacrifice the absolute guarantees and use a learning scheme,as HSTS does. In this design, the lookups would proceed asynchronously, but the results would be persisted by the browser in order to protect future connections. This is workable, but a similar scheme that used HSTS-like HTTP headers would be able to achieve the same results with dramatically reduced complexity.

Finally, as DNSSEC is signature based, there's no reason why DNSSEC records have to be transported using the DNS protocol. If the correct data and signatures can be delivered in another fashion, they can be verified just as well. So, as an experiment, Chrome accepts a form of self-signed certificate that carries a DNSSEC chain proving its validity [10], effectively adding DNSSEC as another root CA. Since it'll be several decades before any new form of certificate can achieve 99 or even 95 percent acceptance in browsers, there's no chance of major sites using them. But there are many HTTPS sites on the Internet that don't currently have a valid certificate and this may be attractive for them. We'll be evaluating the level of support in twelve months and considering whether to continue the experiment.

In conclusion, there are several modern developments that HTTPS sites should be planning and implementing right now. If nothing else, I highly recommend using the HTTPS scanner at https://ssllabs.com to check your sites for any configuration errors. Securing cookies and fixing mixed scripting is essential, and for sites that are exclusively HTTPS, HSTS should be implemented. Certificate validation is likely to see some changes in the coming years and it's worth keeping an eye open, even if there's no immediate action needed for most sites.

References

[1] http://codebutler.com/firesheep.

[2] http://www.eff.org/deeplinks/2011/05/syrian-man-middle-against-facebook.

[3] https://www.eff.org/deeplinks/2011/08/iranian-man-middle-attack-against-google.

[4] http://www.thoughtcrime.org/software/sslstrip/.

[5] http://tools.ietf.org/html/draft-hodges-strict-transport-sec.

[6] https://wiki.mozilla.org/Security/CSP/Specification.

[7] https://www.eff.org/observatory.

[8] http://tools.ietf.org/html/draft-hallambaker-donotissue-04.

[9] http://tools.ietf.org/html/draft-ietf-dane-protocol-11.

[10] http://www.imperialviolet.org/2011/06/16/dnssecchrome.html.

[11] http://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy_for_cookies.

Of Headless User Accounts and Restricted Shells

JAN SCHAUMANN

Jan Schaumann is a Principal Paranoid at Yahoo!, a nice place to stay on the Internet, where he worries about scalable infrastructure and systems architecture. He is also a part-time instructor at Stevens Institute of Technology, where he teaches classes in system administration as well as in UNIX programming.
jschauma@netmeister.org

UNIX system accounts not bound to a particular user, so-called "headless user accounts," are frequently used to allow for automation of certain tasks. For security reasons, such headless accounts usually have a very restricted shell, allowing only a few select commands. At the same time, system administrators and service engineers frequently have a need to let such accounts execute additional commands, even though allowing an interactive shell is not an option. To address this problem, we developed a command interpreter called sigsh [1] that requires a cryptographic proof of authenticity and integrity (i.e., a "signature") by an authorized party before it executes a set of commands. sigsh(1) is currently used by Yahoo! Inc. on over a quarter-million hosts to help discover potential software vulnerabilities.

Systems engineers frequently make use of a headless account in order to, for example, automate the transfer files in and out of a host, perform certain asynchronous monitoring and reporting tasks, or run specific commands. In addition to possibly diluting the audit trail of any such actions, such accounts may pose a risk if access credentials (such as a public SSH key) are shared. To mitigate the risk of such an account being used in unauthorized ways, it is common practice to restrict the set of commands the account can execute to a select few. However, by restricting the set of allowed commands, usability is lost; people frequently need to be able to run additional commands not included in the restricted shell's internal whitelist. As a result, it is unfortunately not entirely uncommon for engineers to simply change the restricted shell to a fully interactive shell, proving once more, "The more secure you make something, the less secure it becomes" [2].

The conundrum posed can thus be described as the need to combine the conflicting requirements of unrestricted access for usability reasons with a restricted and individually authenticated way to execute commands: what's required is a way to allow arbitrary commands, provided they were sanctioned by somebody we trust. ("Arbitrary" here does not mean any random command, but, rather, any command not previously whitelisted.)

Account Types and Trusted Commands

In general, headless user accounts can be grouped into the following categories:

- system accounts provided by the operating system (OS)
- headless accounts with a completely disabled shell (such as /sbin/nologin or /usr/bin/false)
- headless accounts with a restricted shell
- headless account with an interactive shell

Let's take a close look at each of these cases, focusing on two core components in system security: data integrity and authenticity (the third main component, data confidentiality, is, in this context of command execution, irrelevant, although provided by the transport mechanism, that is, ssh/ssl) and the impact on the trustworthiness of the system as a whole.

System Accounts and Headless Accounts with Logins Disabled

The concept of system accounts and how they help implement the Principle of Least Privilege is assumed to be understood, so allow me to simply assert that these accounts are implicitly trusted to be running the given service (and any processes forked off it), but are not trusted to execute anything else. The most common examples for this type of account are the various system accounts used by some of the standard UNIX daemons such as named (commonly used by the DNS system), apache (commonly used by the Apache Web server) or sshd (used by the SSH service for the explicit purpose of privilege separation), to name but a few. With interactive logins explicitly disabled, they are included here purely for completeness' sake.

Headless Accounts with a Restricted Shell

Some headless accounts need to be able to run commands that are triggered asynchronously. That is, while it's possible to use these accounts to run scheduled commands (say, via cron(8)), their main purpose is to allow semi-interactive access to the system from the outside. A typical setup consists of a system account that is allowed to execute a pre-determined set of commands and a mechanism to authenticate and trigger remote invocations of said commands, such as retrieval or deposit of data files.

At Yahoo!, we usually perform these functions using a specific account included in our OS images with a custom restricted shell. This shell only allows a select set of commands (most notably the use of rsync(1), scp(1), and tar(1)) and is meant to let engineers safely transfer files between hosts in an automated fashion. The intention here is to avoid letting a headless user account run arbitrary commands that might be used to compromise a system.

The trust model for these kinds of accounts is somewhat complex: they are explicitly untrusted, but are simultaneously trusted to execute a very small set of commands only, because their use has been reviewed and determined to not pose a risk under the given circumstances (or, more precisely, the risk has been determined to be outweighed by the functionality gained by allowing this use).

The main problem with this approach is that it tries to impose a one-size-fits-all solution on all use cases. If a specific command needs to be added to the list of whitelisted executables, this requires careful review, as the command would then be made available to all users of this restricted shell. Thus, any and all use cases of the new command need to be considered (in contrast to the possibly very restrictive use case initially proposed). At a company the size of Yahoo!, this opens up a very wide field.

To illustrate the problem with this approach, consider the example of an often requested feature addition for this shell, namely, to allow execution of the ln(1) utility. Most specific use cases are non-controversial and ought to be allowed—however, ln(1) is also frequently used to set up an attack known as a "symlink race," which is based on a race condition when creating temporary files leading to information exposure or corruption. For this reason, ln(1) executions cannot be approved as a generic command in this restricted shell.

Similarly, the existing feature in some shells to invoke a "restricted shell" (think `bash -r` or rksh(1)) has proven itself to be much too stringent for practical use. Many tasks that need to be run are impossible in such an environment; at the same time, many such implementations can trivially be circumvented (for example, by invoking an editor that allows you to invoke a new shell).

Headless Account with an Interactive Shell

Due to the shortcomings of the restricted shell, and at times also simply due to ignorance or laziness, some people set up headless users with a fully interactive shell (for example, /bin/bash).

The concern here is that this effectively opens up a regular user account for use by many people and other automated systems. The more people have access to the login credentials (i.e., the ssh keys used to authorize as the headless account), the more likely it is that these credentials might be compromised or abused, be that by way of accidental exposure (granting read permissions to the private ssh key to members outside of your team) or even maliciously. This, in turn, may lead to unauthorized access to a host and any of the data stored on the host, enabling a possible attacker not only to access specific data, but also to mount additional privilege escalation attacks from inside the host.

With an interactive login shell, these accounts fall into a bizarre state of simultaneously being completely trusted (effectively, by virtue of being able to run any given command) and explicitly not being trusted at all (implicitly, by being a headless user account; explicitly, by policy).

A Signature Verifying Shell

To overcome the above-mentioned issues, a new solution is needed. Engineers should be given the ability to define a wide range of commands to be run headlessly, but at the same time it must be ensured that they cannot cause problems by being invoked in unintended ways. For example, the headless account should be able to run `ln -s <dated-dir> <dir>` but not be allowed to `ln -s /etc/passwd /tmp/4jc5ba`, for example.

Once we change the goal from trying to determine a fixed list of commands that are always safe to execute—a difficult task, given the intentional flexibility of the UNIX operating system family—and shift our focus to the underlying trust model, it quickly becomes clear what is needed: a shell that verifies that the commands it is about to execute come from a trusted user, but allowing such users to run any command they choose. That is, we are not trying to protect the account from authorized users running "bad" commands, but, rather, access by unauthorized users to any commands.

To implement this new paradigm, we need a way to feed the shell a signature of the code to be executed and for the shell to be able to verify validity and trustworthiness of the signature. The most obvious solution for many people might be to implement such signature verification using a PGP-based approach. However, even though an entirely suitable solution, the UNIX-based tools that implement a PGP-based public-key infrastructure may not be part of the basic OS images as installed on all of your servers. To ensure ease of adoption of the new tool, the prerequisites need to be restricted to the bare minimum. That is, the tool needs to work without requiring installation of any additional packages.

Fortunately, there is another set of tools that can be used to accomplish the same goal and which is included in most stock open source images as a core component. The openssl(1) utility implements the de facto industry public-key cryptography standard (PKCS) #7 secure message standard via its smime(1) utility, which allows, among other things, for signature creation and verification of secure/multipurpose Internet mail extension (S/MIME) messages as defined in RFC 5751. The signing of such a message simultaneously provides message integrity (the message was not modified in transport and is in fact what was sent) and authentication (the origin of the message is confirmed).

Even though S/MIME is, as the name suggests, mainly used in the context of email, a "message" can of course be anything—a shell script, for example. That means that you can authenticate and verify a given shell script so long as you have the right certificates installed on the host in question:

```
openssl smime -verify -inform pem -CAfile \
        /etc/sigsh.pem <input
```

The certificate file found in /etc/sigsh.pem contains the public-key certificates of all users who are authorized to sign commands for execution by this shell. A certificate is generated via a command like the following:

```
openssl req -x509 -nodes -days <expiration> \
        -newkey rsa:2048 -batch \
        -keyout <keyfile> -out <certfile>
```

It is worth noting that this certificate creation is a one-time step to be issued by the user in question and that /etc/sigsh.pem may contain any number of certificates. That is, multiple people can simultaneously be allowed to sign scripts for execution, eliminating the possibility of a single point of failure.

Once a certificate has been created and the public component has been installed on the desired hosts, a script can be signed for execution as follows:

```
openssl smime -sign -nodetach \
        -signer <public-cert> \
        -inkey <private-key> -in <script> \
        -outform pem
```

The result is a signed S/MIME message generated on stdout containing the given script. This can be either directly piped into an ssh(1) connection to the given host or simply stored in a separate file. Once signed, anybody can invoke the commands so long as they have access to the signed script. This is of particular importance, as it is desirable to limit the number of people able to sign scripts for remote execution by a headless account, but simultaneously to be able to let a larger number of engineers, or even other systems, run the commands headlessly without opening the door to let them run any other command.

Putting it all together, the following becomes a pipeline illustrating script signing, verification, and execution:

```
# script signing
openssl smime -sign -nodetach \
       -signer <public-cert> \
       -inkey <private-key> -in <script> \
        -outform pem   |   \
# script verification
openssl smime -verify -inform pem -CAfile \
        /etc/sigsh.pem  |  \
# script execution
/bin/sh -s
```

As a standard UNIX command pipeline, it is of course possible to redirect the output of any one of these steps into a file or to insert additional commands in between. For example, it would make sense to let a trusted engineer review and then sign the contents of the file script.sh into the file script.pem and then let another system execute this script on another host via:

```
cat script.pem | ssh headless@remote-host   \
        "openssl smime -verify -inform pem  \
        -CAfile /etc/sigsh.pem | /bin/sh -s"
```

This simple pipeline is the only thing the account on the remote host needs to be able to execute, yet using this construct it is possible to let it run any kind of command. In fact, this short openssl(1) command piping into a regular shell is the basis for the new tool we developed, sigsh(1). With some syntactic sugar, some assurance of meaningful exit codes, commentary, and the like, we managed to grow this simple pipeline to over 140 lines, but at its core sigsh(1) easily fits into a twitter message [3].

Threats Not Protected Against

Within the context of trust relationships between systems and users, headless users and commands executed, it is important to note that, while sigsh(1) provides assurance that the commands fed into it were the ones a trusted authority provided, there are a number of important caveats:

- sigsh(1) reads the list of certificates to trust from /etc/sigsh.pem, i.e., the local file system.
- /etc/sigsh.pem may contain multiple certificates.
- Certificates may expire.
- Host administrators have control over the certificates.
- sigsh(1) does not verify that the commands it executes are themselves trustworthy.

Looking at these items, a connection between the flexibility this program provides and possible issues can easily be seen. At the same time, it is worth remembering what we wanted—use of the headless user to gain unauthorized access to a host, that is, privilege escalation. We also made some assumptions, either explicitly or implicitly.

It is entirely true that a user with permissions to write to /etc/sigsh.pem can update that file to add their own certificate. However, an attacker capable of doing that already has, by definition, gained superuser privileges, and is able to install any other back door or wreak havoc in a myriad of other ways.

Multiple certificates, certificate expiration, and control over the certificates by host owners all provide precisely the desired flexibility: the respective drawbacks (multiple authorities, users forgetting to renew their certificates, the operational overhead involved in getting new certificates installed, engineers possibly removing a central authority's certificate, etc.) are all inherent in the design but are offset by exactly that flexibility, as required within the given threat model.

The last item listed above, however, deserves additional attention: it is worth stressing that protection against accidental execution of compromised binaries is not a goal. That is, the system does in fact assume that any of the commands fed into the shell after signature verification is indeed safe to execute.

Related Work

Our investigation of the initial problem statement led us to a number of interesting related projects. Within the UNIX world, there is a lot of focus on technologies that prevent accidental execution of compromised binaries, of detection of tampering with the system, and the like.

NetBSD, for example, has developed a file integrity subsystem named Veriexec [4]. This system allows you to permit execution of individual commands or command-interpreters only if they match a given signature. While this sounds similar to what sigsh(1) implements, it addresses a rather different threat model: here, the system does not verify that a sequence of commands was approved by a trustworthy entity to be executed but, rather, that a command, when it is to be executed, is in fact unmodified.

Other operating systems have similarly focused on this problem of a "trusted path." OpenBSD's Stephanie project initially developed a series of patches implementing "Trusted Path Execution" [5], which focuses on assurance of ownership and possible modifications of executables prior to invocation. A Linux Security Module was based on this work, similarly focusing on the integrity of the executables. While highly desirable functionality, it does not relate directly to the problems sigsh(1) was developed to address.

Early on in the conceptual development phase of sigsh(1), one system we encountered did, however, appear to exhibit all desired features: a shell that allows the administrator to set an "execution policy" specifying under what circumstances scripts may be executed, including settings that require a valid signature of said scripts. This tool had but one drawback: it only runs on Microsoft Windows.

The "Windows PowerShell" [6] is, much like a regular UNIX shell, both a command interpreter and a scripting language, which implements the concept of an "execution policy" that allows you to specify, for example, that any and all scripts executing using the PowerShell must be accompanied by a valid signature verified prior to execution.

It would be highly desirable to integrate the concept of multiple execution policies into our simple sigsh(1) implementation; at the same time, it would be nice if it were possible to allow signatures from a given certificate to work only for a subset of commands.

Of Signatures, Revocation of Privileges, and Audit Trails

One of the key points of the certificate-based solution is that certificates have an expiration date. That is, any given signing party may only be able to sign scripts for execution for a limited amount of time—this ensures regular audits of the list of certificates contained in /etc/sigsh.pem.

Similarly, certificates can be revoked. That is, if one of the people with the authority to sign scripts leaves the company, there is no need to wait for the certificate to expire. Instead, the ability to simply revoke the certificate and thus disable it without any changes occurring on the client hosts would be desirable. This, however, would require sigsh(1) to check a certificate revocation list (CRL) and thus would introduce additional complexity (and network communications at runtime). It was decided that letting the hosts' configuration management system handle control of the contents of /etc/sigsh.pem was sufficient: removal of a trusted cert prior to its expiration thus becomes trivial.

Finally, sigsh(1) itself does not currently implement any sort of audit trail. While certificates added to /etc/sigsh.pem can be tracked via the configuration management system's changelog, input to the shell is executed without logging if it can be verified against the list of certificates found on the host. It would be desirable to have the shell log the commands executed, the identity of the signing party, and any errors or repeated signature mismatches. Future versions will likely include this ability, allowing you to discover and react to (intentional or accidental) misuse of the tool.

Conclusion

sigsh(1), Yahoo!'s simple signature verifying command interpreter, allows you to use headless accounts with arbitrary yet trusted input scripts by checking them against public-key certificates prior to execution. This removes a number of hurdles in the setup of complex interconnected systems that require asynchronous event triggering or data collection via non-trivial scripts and commands, and it improves overall system security.

The tool, implemented in only a few lines of code and using the universally available openssl(1) tool for all heavy lifting, puts the power of self-administration into the engineer's hands and eliminates cumbersome and overly restrictive solutions that are frequently circumvented. Deployed on over a quarter-million hosts, sigsh(1) has a proven track record across all divisions of Yahoo!.

In February of 2011, Yahoo! open sourced sigsh(1): it is available for use by anybody under a BSD-style license from github.com. Future enhancements may include more fine-grained control over who may sign what kinds of scripts or what kinds of signatures are required under what circumstances. Integration with a host-based file integrity check would then complete the goal of having assurance that commands executed by headless users are…"safe."

References

[1] sigsh—a signature verifying command interpreter: http://www.netmeister.org/apps/sigsh/.

[2] Don Norman, "When Security Gets in the Way": http://jnd.org/dn.mss/when_security_gets_in_the_way.html.

[3] https://twitter.com/#!/jschauma/status/35490334251294720.

[4] The NetBSD Veriexec subsystem: http://www.netbsd.org/docs/guide/en/chap-veriexec.html.

[5] Niki A. Rahimi, "Trusted Path Execution for the Linux 2.6 Kernel as a Linux Security Module," in *Proceedings of the 2004 USENIX Annual Technical Conference:* http://www.usenix.org/events/usenix04/tech/freenix/full_papers/rahimi/rahimi_html/index.html.

[6] Microsoft, "Windows PowerShell": http://technet.microsoft.com/en-us/library/bb978526.aspx.

DevOps from a Sysadmin Perspective

PATRICK DEBOIS

Patrick Debois is a senior independent consultant, who has made a habit of changing both his consultancy role and the domain in which he works: sometimes as a developer, manager, sysadmin, tester, or even as customer. The one thing that annoys him most is the great divide between all these groups. Mr. Debois first presented concepts on agile infrastructure at Agile 2008 in Toronto, and in 2009 he organized the first DevOpsdays conference. Since then he has been promoting the notion of DevOps to exchange ideas between these different organizational groups and show how they can help each other achieve better results in business. He can be found via his blog, http://jedi.be/blog, and twitter, @patrickdebois. Patrick.Debois@jedi.be

While there is not one true definition of DevOps (similar to cloud) [0], four of its key points resolve around culture, automation, measurement, and sharing (CAMS) [1]. In this article, I will show how this affects the traditional thinking of the sysadmin.

As a sysadmin you are probably familiar with the automation and measurement part: it has been good and professional practice to script/automate work to make things faster and repeatable, and gathering metrics and doing monitoring is an integral part of the job to make sure things are running smoothly.

The Pain

For many years, operations, of which the sysadmin is usually part, has been seen as an endpoint in the software delivery process: developers code new functionality during a project in isolation from operations and, once the software is considered finished, it is presented to the operations department to run it.

During deployment a lot of issues tend to surface. some typical examples are that the development and test environment are not representative of the production environment, or not enough thought has been given to backup and restore strategies. Often it is too late in the project to change much of the architecture and structure of the code and it gives way to many fixes and ad hoc solutions. This friction has created disrespect between the two groups: developers feel that operations knows nothing about software, and operations feels that developers know nothing about running servers. Management tends to isolate the two groups from each other, keeping the interaction to the minimum required. The result is a "wall of confusion" [2].

Culture of Collaboration

Historically, two drivers have propelled DevOps: Agile Development, which led in many companies to many more deployments than operations was used to, and cloud and large-scale Web operations, where the scale required a much closer collaboration between development and operations.

When things really go wrong, organizations often create a multi-disciplined task force to tackle production problems. The truth is that in today's IT, environments have become so complex that they can't be understood by one person or even one group. Therefore, instead of separating developers and operations as we used to do, we need to bring them together more closely; we need more practice, and our motto should be, "If it's hard, do it more often."

DevOps recognizes that software only provides value if it's running in production. And running a server without software does not provide value either. Development and operations are both working to serve the customer, not to run their own departments.

Although many sysadmins have been collaborating with other departments, this has never been seen as a strategic advantage. The cultural part of DevOps seeks to promote this constant collaboration across silos, in order to better meet business demands. It goes for "friction-less" IT and promotes the cross-departmental/cross-disciplinary approach.

A good place to get started with collaboration is places where the discussion often escalates: deployment, packaging, testing, monitoring, and building environments. These places can be seen as boundary objects [3]: places about which every silo has its own understanding. These are exactly the places where technical debt accumulates, so they should contain real pain issues.

Culture of Sharing

Silos exist in many forms in the organization, not only between developers and operations. In some organizations there are even silos inside operations: network, security, storage, servers groups avoid collaboration, and each works in its own world. This has been referred to as the Ops-Ops problem. So in geek-speak, DevOps is actually a wildcard for *dev*ops* collaboration.

DevOps doesn't mean all sysadmins need to know how to code software now, or all developers need to know how to install a server. But by collaborating constantly, both groups can learn from each other and rely on each other to do the work. A similar approach has been promoted by Agile Development between developers and testers. DevOps can be seen as the extension of bringing system administrators into the Agile equation.

Starting the conversation sometimes takes courage, but think about the benefits: you get to learn the application as it grows, and you can actively shape it by providing your input during the process. A sysadmin has a lot to offer to the developers: for instance, you have the knowledge of what production looks like, and therefore you can build a representative environment in test/dev. You can be involved in load testing, failover testing. Or you can set up a monitoring system that developers can use to see what's wrong. You can provide access to production logs so developers can understand real-world usage.

A great way to share information and knowledge is by pairing with developers or colleagues: while you are deploying code he comments on what the impact is on the code and this allows you to directly ask questions. This interaction is of great value in understanding both worlds better.

Revisiting Automation

As specified in the Agile Manifesto [4], DevOps values "individuals and interactions over processes and tools." The great thing about tools, as opposed to culture, is that they are concrete and can have a direct benefit. It was hard to grasp the impact of virtualization and the cloud unless you started doing it. Tools can shape the way we work and consequently change our behavior.

A good example is configuration management and infrastructure as code. A lot of people rave about its flexibility and power for automation. If you look beyond the effect of saving time, you will find that it also has great sharing aspects: it has created a "shared" language that allows you to exchange the way you manage systems with colleagues and even outside your company by publishing recipes/cookbooks on GitHub. Because we use concepts such as version control and testing, we have a common problem space with developers. And, most importantly, automation is freeing us from the trivial stuff and allows us to discuss and focus on what really matters [5].

Revisiting Metrics

Measuring the effects of collaboration can't be done by measuring the number of interactions; after all, more interaction doesn't mean a better party. It's similar to a black hole; you have to look at the objects nearby [6]. So how do you see that things are improving? You collect metrics about the number of incidents, failed deploys, number of successful deploys, number of tickets. Instead of keeping this information in your own silo, you radiate these stats to the other parts of the company so they can learn from them. Celebrate successes and learn from failures. Do post-mortems with all parties involved and find ways to improve. Again, this changes the focus of metrics and monitoring from making fast repairs to supplying feedback to the whole organization. Aim to optimize the whole instead of only your own part.

The Secret Sauce

Several of the "new" companies have been leaders in these practices. Amazon with their two-pizza team approach [7] and Flickr with their 10 deploys a day [8] were front-runners in the field, but more traditional companies such as National Instruments are also seeing the value of this culture of collaboration. They see collaboration as the "secret sauce" that will set them apart from their competition [9].

Why? Because it recognizes the individual not as a resource but as resourceful enough to tackle the challenges that exist in this complex world called IT.

References

[0] Patrick Debois: http://www.jedi.be/blog/2011/09/06/DevOpsdays-melbourne-keynote/.

[1] John Willis: http://www.opscode.com/blog/2010/07/16/what-DevOps-means-to-me/.

[2] Damon Edwards: http://dev2ops.org/blog/2010/2/22/what-is-DevOps.html.

[3] Israel Gat: http://theagileexecutive.com/2010/07/06/boundary-objects-in-DevOps/.

[4] http://agilemanifesto.org/.

[5] Ernest Mueller: http://blog.cutter.com/2011/09/11/originality-and-operations/.

[6] Patrick Debois: http://www.jedi.be/blog/2011/09/06/velocityconf-DevOps-metrics/.

[7] http://highscalability.com/amazon-architecture.

[8] John Allspaw and Paul Hammond: http://www.slideshare.net/jallspaw/10-deploys-per-day-dev-and-ops-cooperation-at-flickr.

[9] Jesse Robbins: http://radar.oreilly.com/2007/10/operations-is-a-competitive-ad.html.

Practical Perl Tools

From the Editor

DAVID N. BLANK-EDELMAN

David N. Blank-Edelman is the director of technology at the Northeastern University College of Computer and Information Science and the author of the O'Reilly book *Automating System Administration with Perl* (the second edition of the Otter book), available at purveyors of fine dead trees everywhere. He has spent the past 24+ years as a system/network administrator in large multi-platform environments, including Brandeis University, Cambridge Technology Group, and the MIT Media Laboratory. He was the program chair of the LISA '05 conference and one of the LISA '06 Invited Talks co-chairs. David is honored to have been the recipient of the 2009 SAGE Outstanding Achievement Award and to serve on the USENIX Board of Directors beginning in June of 2010.
dnb@ccs.neu.edu

The vast majority of the columns we've spent together so far have focused on how to improve your life within the bubble of the programming experience. We've looked at tools to make programming easier, more efficient, perhaps even a little more fun. For this column, let's try something different and bust out of our usual snow globe. We're going to look at three ways we can call out to Perl or Perl-based tools from within the editor we are using to improve our lives. So still Perl, but perhaps a little bit more at the periphery than before.

Reflowing and Reformatting Text

Once upon a time, Damian Conway, one of the leading lights of the Perl community, decided he didn't like any of the existing tools for reformatting and reflowing plain text. They couldn't handle bulleted lists, indentation, quoting, embedded structures like lists within quoted text, and so on. Or if they handled them, they didn't handle all of them simultaneously. As the documentation for the module we are about to see notes, if you take this sample text:

```
In comp.lang.perl.misc you wrote:
: > <CN = Clooless Noobie> writes:
: > CN> PERL sux because:
: > CN>    * It doesn't have a switch statement and you have to put $
: > CN>signs in front of everything
: > CN>    * There are too many OR operators: having |, || and 'or'
: > CN>operators is confusing
: > CN>    * VB rools, yeah!!!!!!!!!
: > CN> So anyway, how can I stop reloads on a web page?
: > CN> Email replies only, thanks - I don't read this newsgroup.
: >
: > Begone, sirrah! You are a pathetic, Bill-loving, microcephalic
: > script-infant.
: Sheesh, what's with this group - ask a question, get toasted! And how
: *dare* you accuse me of Ianuphilia!
```

and run it through the UNIX fmt tool (or even the Perl module Text::Wrap), you get this:

```
In comp.lang.perl.misc you wrote:  : > <CN = Clooless Noobie> writes:  : > CN>
PERL sux because:  : > CN>    * It doesn't have a switch statement and you
have to put $ : > CN>signs in front of everything : > CN>    * There are too
```

```
many OR operators: having |, || and 'or' : > CN>operators is confusing : > CN>
* VB rools, yeah!!!!!!!!!  : > CN> So anyway, how can I stop reloads on a web
page?  : > CN> Email replies only, thanks - I don't read this newsgroup.  : >
: > Begone, sirrah! You are a pathetic, Bill-loving, microcephalic : > script-
infant.  : Sheesh, what's with this group - ask a question, get toasted! And
how : *dare* you accuse me of Ianuphilia!
```

Not exactly an improvement. Conway decided to write a Perl module that would grok all of these things, and so the modules Text::Autoformat and Text::Reform were born. Text::Autoformat tries to determine the various structures found in text and then call Text::Reform to reformat them in a pleasing fashion. How pleasing? Here are the results when we run them on our sample text above:

```
In comp.lang.perl.misc you wrote:
: > <CN = Clooless Noobie> writes:
: > CN> PERL sux because:
: > CN>  *It doesn't have a switch statement and you
: > CN>   have to put $ signs in front of everything
: > CN>  *There are too many OR operators: having |, ||
: > CN>   and 'or' operators is confusing
: > CN>  *VB rools, yeah!!!!!!!!! So anyway, how can I
: > CN>   stop reloads on a web page? Email replies
: > CN>   only, thanks - I don't read this newsgroup.
: >
: > Begone, sirrah! You are a pathetic, Bill-loving,
: > microcephalic script-infant.
: Sheesh, what's with this group - ask a question, get toasted!
: And how *dare* you accuse me of Ianuphilia!
```

When Conway wrote the Text::Autoformat module, I believe his main desire was not to call it from within a larger Perl program, but, rather, to let it be used more handily from your favorite text editor of choice. To do that, you need to pass the text you want to reformat out of your text editor into an invocation of the Perl interpreter that looks like this:

```
perl -MText::Autoformat -e "{autoformat{all=>1,right=>75};}"
```

To break this down, it says:

- load the Text::Autoformat module
- then call the autoformat subroutine with the following options:

  all = 1 to instruct the module to reformat all of the text (vs. just the first paragraph)
  right = 75 to instruct the module to reformat things with a right margin of 75

I use that command all the time from within a TextMate macro (for example, on the very text you are reading), but you could map a key in vim to do the same thing:

```
map <C-J> !G perl -MText::Autoformat -e "{autoformat{all=>0,right=>75};}"<cr>
```

I apologize if this seems obvious, but if you attempt to run a command like this in vim (or another editor), and instead of returning nicely reformatted text, your original paragraph is replaced with something that looks like this:

```
Can't locate Text/Autoformat.pm in @INC (@INC contains:
/opt/local/lib/perl5/site_perl/5.14.1/darwin-multi-2level
/opt/local/lib/perl5/site_perl/5.14.1 /opt/local/lib/perl5/vendor_perl/5.14.1/
darwin-multi-2level
/opt/local/lib/perl5/vendor_perl/5.14.1
/opt/local/lib/perl5/5.14.1/darwin-multi-2level
/opt/local/lib/perl5/5.14.1
/opt/local/lib/perl5/site_perl
/opt/local/lib/perl5/vendor_perl/5.14.0
/opt/local/lib/perl5/vendor_perl .).
BEGIN failed--compilation aborted.
```

it means that you will need to install the Text::Autoformat module before you can proceed. For those of you who have multiple versions of Perl installed on your machine (e.g., because you have both the Perl that ships with the system and the one you installed through MacPorts/Homebrew/Fink), sometimes you will find you will get this message because your editor configuration is picking up the wrong Perl (the one without Text::Autoformat installed in its @INC) from your path. An easy fix is to change the command being run to include a full path to the right Perl interpreter (e.g., /opt/local/bin/perl -MText::Autoformat...).

Tidy Your Lousy Code

Although we are not actually doing any programming in this column, this seems like a natural place to point out two other tools that can be called from an editor to improve the programming process. Both of these have made at least one appearance in this column, but I love them too much not to mention them again: Perl::Tidy and Perl::Critic. Both of these things are modules designed to work on Perl code and both come with a script that runs on the command line.

In the case of Perl::Tidy, or, more precisely, when using its accompanying command-line perltidy, code can get read in from stdin and printed out again in a much, much prettier form to stdout. As a demonstration, here's some sample code found embedded in the Perl::Tidy documentation:

```
use strict;
my @editors=('Emacs', 'Vi   '); my $rand = rand();
print "A poll of 10 random programmers gave these results:\n";
foreach(0..10) {
my $i=int ($rand+rand());
print " $editors[$i] users are from Venus" . ", " .
"$editors[1-$i] users are from Mars" .
"\n";
```

If I run it through perltidy from my editor (the command I call is `perltidy -st -q $FILENAME`, but for vi we could use just `:%!perltidy`) using some defaults (more on that in a moment), I get:

```
use strict;
my @editors = ( 'Emacs', 'Vi   ' );
my $rand = rand();
print "A poll of 10 random programmers gave these results:\n";
foreach ( 0 .. 10 ) {
    my $i = int( $rand + rand() );
    print " $editors[$i] users are from Venus" . ", "
      . "$editors[1-$i] users are from Mars" . "\n";
}
```

If you look at the difference between the two, there are lots of little cleanups going on (e.g., the space around arguments in parenthesis). I realize it is a particularly geeky thing to say this, but when I start with code that looks like this:

```
my %a = (
        $a => 1,
        $apple => 2,
        $bigapple => 3,
        $verylargeapple => 'new york',);
```

and I turn it into this using a single keystroke:

```
my %a = (
    $a              => 1,
    $apple          => 2,
    $bigapple       => 3,
    $verylargeapple => 'new york',
);
```

such that the arrows all line up it is deeply satisfying. If you've noticed that all, or at least most, of the arrows in this column have lined up over the years, that's not my doing. I have Perl::Tidy to thank. One last note before I move on to Perl::Critic: I mentioned running perltidy with defaults. Perl::Tidy has a ton of configurable options. Don't like it if your arrows line up? (Of course you do!) Prefer to leave a closing parenthesis at the end of a line of code without wrapping it as above? All of these things can be set as options. By default, if you create a .perltidyrc, Perl::Tidy will attempt to read it to set your favorite options. At the moment I use the following .perltidyrc file, which was recommended in Conway's excellent *Perl Best Practices*:

```
# PBP .perltidyrc file

-l=78       # Max line width is 78 cols
-i=4        # Indent level is 4 cols
-ci=4       # Continuation indent is 4 cols
-st         # Output to STDOUT
-se         # Errors to STDERR
-vt=2       # Maximal vertical tightness
-cti=0      # No extra indentation for closing brackets
-pt=1       # Medium parenthesis tightness
-bt=1       # Medium brace tightness
-sbt=1      # Medium square bracket tightness
-bbt=1      # Medium block brace tightness
-nsfs       # No space before semicolons
-nolq       # Don't outdent long quoted strings
-wbb="% + - * / x != == >= <= =~ < > | & **= += *= &= <<= &&= -= /= |=+ >>=
||= .= %= ^= x="  # Break before all operators
```

(The last line, beginning `-wbb`, should be all one line.)

If you don't feel like setting up a .perltidyrc as I did many moons ago when I first read the book, you can now use a -pbp argument to perltidy and it will set these parameters for you.

The second Perl-based command I mentioned above was perlcritic, installed as part of the Perl::Critic module. The mention of *Perl Best Practices* above is a good segue because that book basically helped spawn Perl::Critic. Perl::Critic is meant to analyze Perl code and determine if it is complying with certain policies meant to enforce coding best practices. The original rules were based on the Conway book, but more have been added over time. Perl::Critic also lets you use add-on modules to add all sorts of different policies to the checking process. When it finds anything that violates any of these rules it will spit out warning messages. If you would like to see examples of these messages, take a peek back at the December 2009 column where I first mentioned both Perl::Critic and Perl::Tidy.

These error messages have a similar form to those you might expect to see emitted from another language's compiler. As a result, most of the editors that offer perlcritic integration do so using a variation of their already existing functionality that lets a user try to compile code from within the editor (jumping to the lines with errors if any are found). There are add-on packages for a number of the more popular editors/IDEs, including vim, Emacs, Komodo, Eclipse (within the Eclipse Perl Integration project), BBEdit, Padre, and so on.

And You Thought Grep Was Cool

For the last tool that we are going to see which you can integrate into your editor, I want to introduce you to three little letters that may significantly improve how you find things in your ever-increasing mountain of data: ack. ack is a grep-ish utility by Andy Lester. Like grep, it was designed to help you find data within files. It is just a bit smarter (okay, a lot smarter). I don't think I can do any better describing why you might want to use it than to quote from the documentation:

Top 10 reasons to use ack instead of grep.

1. It's blazingly fast because it only searches the stuff you want searched.
2. ack is pure Perl, so it runs on Windows just fine. It has no dependencies other than Perl 5.
3. The standalone version uses no non-standard modules, so you can put it in your ~/bin without fear.
4. Searches recursively through directories by default, while ignoring .svn, CVS, and other VCS directories.
   Which would you rather type?
   ```
   $ grep pattern $(find . -type f | grep -v '\.svn')
   $ ack pattern
   ```
5. ack ignores most of the crap you don't want to search:
   o VCS directories
   o blib, the Perl build directory
   o backup files like foo~ and #foo#
   o binary files, core dumps, etc.
6. Ignoring .svn directories means that ack is faster than grep for searching through trees.
7. Lets you specify file types to search, as in --perl or --nohtml. Which would you rather type?
   ```
   $ grep pattern $(find . -name '*.pl' -or -name '*.pm' -or -name '*.pod' \
        | grep -v .svn)
   $ ack --perl pattern
   ```
   Note that ack's `--perl` also checks the shebang lines of files without suffixes, which the find command will not.

8. File-filtering capabilities usable without searching with ack -f. This lets you create lists of files of a given type.

```
$ ack -f --perl > all-perl-files
```

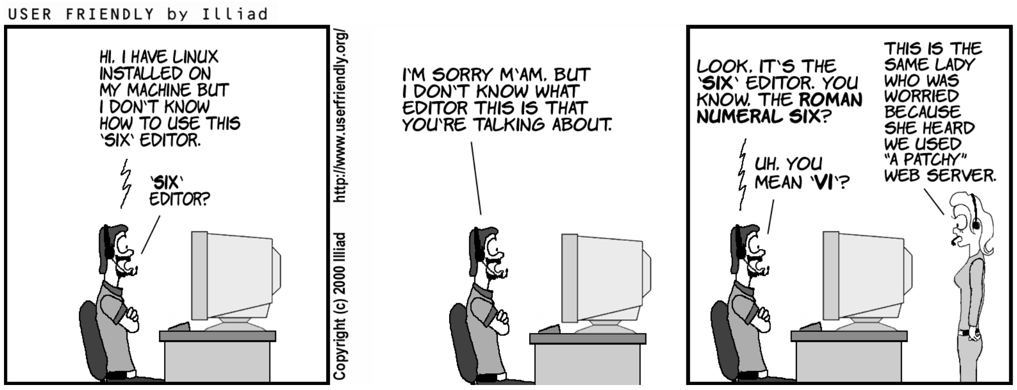
9. Color highlighting of search results.
10. Uses real Perl regular expressions, not a GNU subset.
11. Allows you to specify output using Perl's special variables. To find all #include files in C programs:

```
ack --cc '#include\s+<(.*)>' --output '$1' -h
```

12. Many command-line switches are the same as in GNU grep:
    `-w` does word-only searching
    `-c` shows counts per file of matches
    `-l` gives the filename instead of matching lines
    etc.
13. Command name is 25% fewer characters to type! Save days of free-time! Heck, it's 50% shorter compared to `grep -r`.

So there you go, 13 of the top 10 reasons why ack may replace grep as a command you type on a regular basis. TextMate, Vim, Emacs and other add-ons let you do things like conduct fast searches from within the editor and then jump to the places in the files where your search text was found.

With that high note, I think we'll end our exploration of Perl utilities that can be called from an editor. If you have a particularly cool example of this sort of thing that you use all the time, please write me a note so I can include it in a future column. Take care, and I'll see you next time.


USER FRIENDLY by Illiad
HI. I HAVE LINUX INSTALLED ON MY MACHINE BUT I DON'T KNOW HOW TO USE THIS 'SIX' EDITOR.
'SIX' EDITOR?
http://www.userfriendly.org/
Copyright (c) 2000 Illiad
I'M SORRY M'AM, BUT I DON'T KNOW WHAT EDITOR THIS IS THAT YOU'RE TALKING ABOUT.
LOOK, IT'S THE 'SIX' EDITOR. YOU KNOW, THE ROMAN NUMERAL SIX?
UH, YOU MEAN 'VI'?
THIS IS THE SAME LADY WHO WAS WORRIED BECAUSE SHE HEARD WE USED "A PATCHY" WEB SERVER.

iVoyeur
Changing the Game

DAVE JOSEPHSEN

Dave Josephsen is the author of *Building a Monitoring Infrastructure with Nagios* (Prentice Hall PTR, 2007) and is senior systems engineer at DBG, Inc., where he maintains a gaggle of geographically dispersed server farms. He won LISA '04's Best Paper award for his co-authored work on spam mitigation, and he donates his spare time to the SourceMage GNU Linux Project.
dave-usenix@skeptech.org

At my last job, the Windows sysadmin would plan, for every new software implementation, sufficient time to install and configure the application a minimum of three times. This was not padding—they actually installed and configured every new application the company brought in three times. For more complex applications they would rebuild more often. I seem to recall that they rebuilt "Documentum" [1] something like nine times.

I thought it was kind of crazy at the time, but, looking back, I think I can, if not relate, at least understand. With technology, there is some operational coefficient of long-term success that defies prediction. Sometimes you don't know what you're not going to like about a solution until you've installed it. Sometimes proper integration into the existing environment isn't obvious until a solution is improperly integrated. Sometimes you need to throw a few things at the wall to see what sticks, and sometimes you need to break a thing, to see what can be made of the pieces.

I don't know if this property has a name, but I get that it's there. The Windows guys at my last job built and rebuilt to tune their systems for this variable in a kind of institutionalized brute force attack. They did this every time (probably without being able to articulate exactly why) partly because their choices were limited and partly because that's just the kind of thing they do. I think the reason I (and probably you) find their technique questionable is that, to some extent, optimizing for this property is the meat of what professional system administrators do; that this is what it means to hone our craft. We strive to excel at solving for elegance. A lot of the time, even when we get it right something new will come along that makes us rethink our architecture. Game changers create new possibilities, and our solutions might need to change to encompass them.

I can remember reading the first papers on RRDTool and considering the options for Nagios integration. Various Nagios plugins (as you're no doubt aware) pass back performance data along with their normal output. The way Nagios deals with this performance data is configurable. The shortest path is to configure Nagios to send performance data to a tool that parses out the metrics and loads them into RRDs. Several tools in various languages exist to do this, such as NagiosGraph [2] and PNP [3].

This light glue-layer between Nagios and RRDTool seemed elegant. You were using data you already had in a new way. The regex-based parsing gave you performance graphs across all of your hosts and services for just a few lines of configuration, RRDTool lends itself to exactly this sort of data exportation, and you were using hooks built into Nagios to make it happen. For the price of a Perl script (or whatever) and a few lines of configuration, you'd just bought yourself performance data for every monitored service.

The problems become apparent at around 350 hosts for most modest hardware. It just doesn't scale well. Even for small installations, Nagios isn't going to get anywhere near real-time data; it's intended to operate on the order of minutes, not seconds. Accompanying this realization was a second: namely, that you couldn't easily scale the system horizontally by adding more hardware, nor could you bolt on a new solution in a way that would make it easy to display the data gleaned from both Nagios and the new stuff. Nagios and RRDTool had been tied together for better *and* for worse.

For most sysadmins, Cacti [4] and/or Ganglia [5] changed the game toward discrete systems for availability and performance monitoring. We had to go through and install new agents on all of our hosts, but we did it because these solutions (and Ganglia in particular) do a fantastic job of getting near-real-time data from a massive number of hosts with very little overhead. This also seems elegant, but there are still several problems. For one, Ganglia assumes a cluster model, which is a handy assumption that helps us combine and summarize data, but also forces a dashboard view of our environment that may not always be optimal. For another, it's still difficult to mix and match the data from different sources. If I want to graph something new, then I'm going to have to send it through Gmond or use a different front end to do my graphing.

It seems odd to me, given the problem RRDTool was intended to solve, that taking data from different places and storing it together in such a way that a generic front-end can graph it is this difficult. That last sentence could just about be a reworded mission statement for RRDTool, and yet nearly all the tools we've built on top of it are purpose-specific. I've long thought that the folks writing the front ends were just not thinking big enough, but now I'm beginning to believe this isn't accidental.

Some of the problem might have to do with the way RRDTool itself is architected. Different data sources and types of data, and even different intended uses for the same data imply different RRD requirements. For example, the heartbeat for a metric collected from Ganglia is going to differ wildly from one collected from Nagios. Any higher-level tool must make some assumptions and provide sane defaults, and while I don't think it's impossible to write something on top of RRDTool that could deal with a much larger set of assumptions, I have to admit that RRDTool's configuration rigidity is encouraging the higher level tools to be purpose-specific to some degree.

To have a more source-agnostic storage layer, it would be easier if we had something akin to RRDTool that was more relaxed about how often data was updated, and less concerned about categorizing it into pre-defined types.

Enter Graphite [6].

Graphite was developed internally by the engineers at Orbits.com and changes the game all over again. The name actually refers to a suite of three discrete but complementary Python programs, one of which is itself called "Graphite" (I assume they did this to make it more difficult to write articles about Graphite).

The first of these is Whisper, a reimplementation of the RRD format that makes the modifications to the data layer I mentioned above. Whisper does not particularly care how far apart your data points are spaced, or, indeed, if they arrive in sequential order. It also does not care what kind of data it is internally. Whisper stores all values the same way RRDTool would store a "Gauge" data point.

Data interpretation is handled by the front end using various built-in functions that modify the characteristics of the data when it's displayed. For example, at display time, the user runs the "derive" function to obtain a bytes-per-second graph from byte counter data stored in Whisper in its raw format.

The critically important upshot is that by making the storage layer agnostic to data type and frequency, new Whisper databases may be created on the fly with very little pre-configuration. In practice, the sysadmin specifies a default storage configuration (and, optionally, more specific configurations for metrics matching more specific patterns), and after that all Whisper needs to record a data point is a name, a value, and a date-stamp.

Carbon, the second Python program, listens to the network for name/value/date-stamp tuples and records them to Whisper RRDs. Carbon can create Whisper DBs for named metrics that it has never heard of, and begin storing those metrics immediately. Metric names are hierarchal from left to right, and use dots as field separators. For example, given the name "appliances.breakroom.coffee.pot1.temp", Carbon will create a Whisper DB called temp in the $WHISPER_STORAGE/appliances/breakroom/coffee/pot1 directory on the Graphite server. Carbon listens on TCP port 2003 for a string of the form "name value date". Dates are in EPOCH seconds. Continuing the coffee pot example, I could update that metric with the value 105 with the following command line:

```
echo "appliances.breakroom.coffee.pot1.temp 105 1316996698" | nc -c <IP> 2003
```

I passed `-c` to netcat so that it wouldn't hang waiting for a reply from Carbon. Obviously, you need the netcat with `-c` support to do that (http://netcat.sourceforge.net). Most large Graphite installations front-end Carbon with a UDP datapoint aggregator, but more on that later. The critically important upshot of this is that there is a socket on your network to which anyone (with access) can send data and have it stored and ready for graphing immediately. Whisper's data agnosticism and Carbon's network presence combine in such a way that data collection and presentation is no longer an ops-specific endeavor. For example, Carbon clients have been written for about every popular programming language out there, making it trivial for developers to build applications that send interesting metrics to the Graphite server. There's no reason why the security guys couldn't tie in their snort stats and/or logsurfer instances for that matter.

In his now famous blog post "Tracking Every Release" [7], Mike Brittain shares how the engineers at Etsy have their deployment tool sending a 1 to their Graphite server every time a code deployment takes place. Since Whisper doesn't care about data frequency, it's possible to graph instances of things like deployments that only happen every so often. At Etsy they superimpose these data points as vertical lines over other metrics to correlate events, such as PHP warnings per second, to code releases.

Finally, Graphite is the Web front end to ... well, Graphite. Graphite runs on the Apache Web server with mod_python, and includes a novel Web-based command-line interface (with tab completion) that makes it easy to create on-the-fly graphs from any combination of stored metrics. It also has a tree view reminiscent of Cacti, and a user-configurable dashboard view. My favorite piece of the front end is the URL interface, which allows the creation of graphs by specifying URLs. This feature is something I've been wanting for a long time. It enables integration with just about every monitoring system out there, including Nagios via its "action_url" attribute.

This seems elegant. We've certainly come a step closer to separating the polling engines from the storage engines from the display engines. It's unfortunate that, once again, I'm looking at a single application that is storing and displaying the data, but I think this has more to do with the lack of independent front ends that support the Whisper data store than any intrinsic dependency between the two. Graphite changes things. It introduces new possibilities. I plan to write a few more articles exploring Graphite, its installation and usage intricacies, and especially the integration possibilities. So stay tuned.

Take it easy.

References

[1] EMC Documentum: http://www.emc.com/domains/documentum/index.htm.

[2] NagiosGraph: http://nagiosgraph.sourceforge.net/.

[3] PNP: http://docs.pnp4nagios.org/pnp-0.4/start.

[4] Cacti: http://www.cacti.net/.

[5] Ganglia: http://ganglia.sourceforge.net/.

[6] Graphite: http://graphite.wikidot.com/.

[7] Mike Brittain's "tracking every release" post: http://codeascraft.etsy.com/2010/12/08/track-every-release/.

/dev/random

ROBERT G. FERRELL

Robert G. Ferrell is a fourth-generation Texan, literary techno-geek, and finalist for the 2011 Robert Benchley Society Humor Writing Award.
rgferrell@gmail.com

Today, whilst cogitating on whether I had brain cancer or just a headache from smelling cabbage cooking, I decided instead to cogitate on the word "exploit." I thought deeply about what it means *to exploit,* because that's what I wanted to write about. I came face to face in the process with one of the more insidious curses to which writers are subject, which I will term "lexicopathy" with the full knowledge that this name has probably already been taken by a Croatian metal band and is stapled to utility poles across Zagreb even as I write this. My defense will be that I don't speak Croatian and so can't be held accountable.

Lexicopathy, as I have defined it, is the condition that arises when writers get so close to a word that they get the urge to dismantle it and play with the structural components as though they were parts from an Erector set (with which activity I was, not surprisingly, inordinately enamored as a lad). More to the point, it prevents said writers from actually *writing* anything (which many will no doubt consider a salutary result) and instead lures them along the primrose path to a secret garden where the similarity between *etym*ology and *entom*ology is revealed to be no mere linguistic coincidence.

Crawling around on the bark and buds of the tree of English are a plethora of multi-legged beasties, prominent among which are the adjective-flies, noun-beetles, verbipedes, pronoun-bugs, adverbydids, gerund-hoppers, conjunction-worms, preposition-mites, and participle-thrips. Interjection-midges can often be found buzzing in the writer's face in an annoying manner, surprisingly loud for their tiny size. The forest floor is aswarm with armies of article-ants. (Two species are present: definite and indefinite; the latter are much more difficult to pin down.) This is where words go to be broken down, digested, and regurgitated as new offerings to provide the folks who compile dictionaries with a robust livelihood.

Those suffering from lexicopathy are able miraculously to see and interact with these grammaranimals and the curious world they inhabit. To a normal person the word "exploit" is just that: a word, a sequential list of alphabetic characters that forms a unit we English speakers have been taught to interpret in a certain manner. It creates an abstract image in our minds—a concept with specific associated memories and constructs. When the system works right it's more or less the same concept for me as it is for you. To the denizens of the secret garden, however, "exploit" is a juicy, crunchy, sweetmeat ripe for the feasting. They pull it apart like a succulent crab leg and suck out the savory marrow, accompanied by a cool, refreshing mixed metaphor salad.

Let us begin this lexicological repast with the prefix, a very fine place to start. "Ex" is one of the more versatile of two-letter Latin expropriations, being gainfully employed in words sprinkled liberally throughout the dictionary. (Most of them are located in the "E's," come to think of it, but you get my drift.) (By "Latin" I mean, incidentally, "Roman." How "Latin" came to be applied as well to those with a rich and complex heritage based on a mélange of Spanish and indigenous Mesoamerican/South American cultures is puzzling to me, but then so are a great many other things in life.)

"Ex" can indicate "out of," "landed from," "former," "exclusive of," "drive out," "not including," "no longer occupying," and several other related meanings. The *Compact Oxford English Dictionary* (2nd edition), in fact, spans two pages (pp. 480–481) in an exploration of "ex." It has gained a measure of modern linguistic notoriety serving as the monosyllabic representation, often uttered harshly, with venomous disdain, and occasionally accompanied by forcible expectoration, for a former significant other. "Ex" additionally morphed into prefixes like "ef" and "eb" over time, or so sayeth the venerable OED. It may also be heard in the classic pre-WWII college cheer: "give 'em the ex, give 'em the ex, give 'em the e-x ex!" At least, that's what it sounded like to me watching old cartoons before dawn on Saturday mornings in the '60s. It could have been the intense sugar buzz affecting my hearing.

"Ploit" has no real meaning by itself (although "ploiter" once signified to putter around ineffectively), except as an acronym for "Path Loss Over Irregular Terrain." However, a few minutes' consideration will reveal that this is singularly appropriate. To navigate successfully to one's destination is to "pilot." Why, then, wouldn't an unsuccessful application of that procedure be to "ploit"? Why, it's as sensible as lemon in your iced tea.

This brings us around once again, somewhat the worse for wear, to "exploit." By now you're probably looking at that word in a wholly different light (the day having worn on considerably since you started reading, slowby). Our little bout with lexicopathy has left us weak, perspiring, and vaguely nauseous: true. But in exchange it has whisked away a deceptive camouflage covering the rich tapestry woven by that simple and increasingly oft-encountered infosec cliché, e-x-p-l-o-i-t. Let me hear you say it. No, wait until you've finished swallowing your coffee first. Jiminy. I can't take you *anywhere*.

Employing our newfound lexicological onion-peeling skills, let us drag our word of the day over under the streetlamp and examine its components more closely: "out of," "landed from," "former," "exclusive of" "drive out," "not including," or "no longer occupying," "path loss over irregular terrain." Here, then, is the deceptively simple-appearing word "exploit," which the unwashed masses so smugly assume they understand, laid out bare naked on the driveway. We alone are able to derive its true meaning from our examination of the deepest roots: those that lay shrouded in shadows in the secret garden. We alone are enlightened. (I must admit that many of my trips to the dictionary leave me endarkened.)

After exhaustive examination of the historical evidence, and taking into account the various etymological elements that come into play—bearing in mind, of course, the principles of enfilade and defilade and cross-indexing to the commodities markets—we are able to synthesize using least-reasoning analysis a definition for "exploit" that well and truly represents both its simplistic overt and more complex underlying metaphoric linguistic fabric (85% cotton, 10% silk, 5% Rhodesian ridge-back iguana hair. Machine wash, dry in the microwave using the "artichoke hearts casserole" setting).

Where was I? Oh, yes—the definition of "exploit." That's easy: a retired dyslexic aircraft operator. I have no idea how this relates to taking advantage of computer vulnerabilities. Possibly a misunderstanding.

Book Reviews

ELIZABETH ZWICKY, WITH SAM STOVER

Drive: The Surprising Truth About What Motivates Us

Daniel Pink

Riverhead Trade, 2011. 272 pp.

ISBN 978-1-59-448480-3

It is a coincidence that this book and *Gamification* showed up in the same lot of books to review, but it is in most ways a happy coincidence, precisely because they come at things from pretty much opposite directions. *Drive* is about inherent motivation, doing things because they are worth doing. *Gamification* is about extrinsic motivation, offering people prizes in the hope that they will do things. Before you decide that gamification is the way you want to go, it's worth paying careful attention to *Drive*'s section on when straightforward reward systems make sense.

Drive is a convincing explanation of why working with rewards and threats is not an effective way to get great results; it ties nicely to other books I've recommended (there's a pleasant synergy with Carol Dweck's work on mindsets, for instance). It is, for me, just on the acceptable side of breathless enthusiasm and catchy naming schemes, although, frankly, "Motivation 3.0" does make me wince; if your tolerance for management-speak is low, *Drive* may be over the line. On the other hand, if you want to try to convert traditional managers of your acquaintance to a less controlling style, it's likely to be palatable to them. It should also be very useful if you are a non-controlling manager in a work environment where that's not the norm, and you need some way to get a stamp of approval for what may be seen as letting your people run wild.

The Economics of Software Quality

Capers Jones and Olivier Bonsignour

Addison Wesley, 2011. 559 pp.

ISBN 978-0-13-258220-9

This book is much like waybread is said to be: nourishing but not particularly enjoyable. It does have moments of humor, especially if you appreciate the spectacle of well-earned academic ire, but it is repetitive, full of useful but difficult tables, and generally a slog to read. On the other hand, if you feel that what you've always wanted is a good feel for the actual data about what makes better software, particularly when building big systems, this is what you've been looking for.

Agile? It actually helps people make better software! That is, as long as you're not building anything with 10,000+ users, so all you Web companies out there, feel worried now. Almost any official standard? Will help you not be completely lousy, but will not help you excel. Lines of code? Just as stupid a measurement as you might have imagined. All those claims, with dollar amounts, of how it's cheap to fix a problem you discover in the requirements stage, but expensive once you've deployed? The authors very carefully verified that, indeed, those claims appear to have been made up, and are for many reasons nonsensical. But to the extent they are true, they imply that you should very, very carefully test your requirements—which almost nobody does.

This is fascinating meaty stuff, and really fun to think about. It's worth the trouble if the topic interests you, but it would be nice if it were somewhat less arduous.

Think Stats

Allen B. Downey

O'Reilly, 2011. 112 pp.

ISBN 978-1-44-930711-0

This is a nice, experimental approach to statistics for programming types, with good questions, real data sets, and practical instructions on how to write programs to work with statistics (in Python, which might or might not be your first choice but is at least a general-purpose programming language). Unfortunately, it's a textbook, and it has exercises. What it doesn't have is answers to the exercises. If you can do the exercises (especially if you can, but you won't if you can look the answers up), this is not a problem. For the right sort of personality, this is going to be an extremely effective way of learning basic probability-based statistics. If you want a voyage of discovery, go for it; if you were looking for more of a guided tour, pick another book.

You should also be aware that this book is going to be of much more use to you in doing hands-on statistics than it is in passing statistics exams in any other course. The programming-based methods it teaches are useful, but they are not always mainstream approaches, which is entirely intentional on the author's part.

Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps

Gabe Zichermann and Christopher Cunningham

O'Reilly, 2011. 192 pp.

ISBN 978-1-44-939767-8

This time seems to be the column for recommendations with caveats. Suppose you have a Web site, or you're about to have one, and you have determined that what you want out of life is game features, some kind of a scoring system, but you're not sure how you would do that. What do you give scores to? What are the common pitfalls? How do you implement the system? If so, this is the book for you.

This is not the book for you if you need a thoughtful discussion of when gamification is a good idea, because its discussion of this is not deep or convincing. If you want your child to eat broccoli, eat broccoli happily yourself, serve very small amounts in varied ways, repeat often in a context where experimentation is safe—or, better yet, just don't worry about it, broccoli is not a deal-breaker, there is really no need for your child to eat it, there are lots of other vegetables in the world, and if you don't do anything silly, most kids will eventually grow up to eat and enjoy broccoli. That's not really unsolicited parenting advice; that's semi-solicited gamification advice. Making broccoli-eating into a game may "work." If done well, it will probably have no long-term ill effects.

But it's a waste of your valuable time and energy at best, and at worst, it's an invitation to turn a non-issue into a struggle. What works as a way to get kids to eat broccoli involves some general principles (model the behavior you want, provide different ways of reaching the desired goal, don't fight about it) and some domain-specific knowledge (kids often don't eat broccoli because it's bitter, and bitterness is best cut by sourness or by creating sweetness, for instance by roasting—my kid eats raw broccoli with balsamic vinegar, which is sour and sweet, very happily, partly for these excellent scientific reasons and partly because she'll eat anything with balsamic vinegar on it). Fortunately, child broccoli eating is not actually a Web or mobile app, putting it outside the book's actual scope, but it is almost impossible to resist discussing once the authors bring it up.

Making something into a game often changes the context. That's fine if you were already in a superficial context, but it can be a real loss. For instance, many of your worst customer service experiences were probably created at least partially by somebody's bright idea about adding a scoring system to the customer service process. Many of your worst children-and-food nightmares were probably created by somebody deciding that they needed to win some food-related game. Any deep meaningful human interaction can be turned into a power struggle if you're not careful. I find this book's discussion of these issues far too uncritical.

I'm also annoyed by the book's tendency to mention sites as if the reader will automatically know what's being discussed. I'm supposed to go to Huffington Post just to discover what is undesirable about their badges? There couldn't be an example? There were frequent references to games or sites without quite enough context; I often mostly knew what they were talking about, but sometimes, as in the Huffington badges, I had no idea. Perhaps you can only gamify things if you already hang out on every popular gamified site and play some version of every popular game, but it seems like this could be avoided.

Ghost in the Wires

Kevin Mitnick

Brown and Company, 2011. 413 pp.

ISBN 978-0-31-603770-9

Wow. Just wow. I know the topic of Kevin Mitnick is a volatile one, but regardless of your opinion, you gotta read this book. I've never experienced an emotional rollercoaster in a technical book before—this book steps out of the ordinary and takes you for a ride. I find myself alternating between condemnation and adulation. Some parts of the story aren't pretty (or legal), but it's all interesting. For everyone who doesn't know who Kevin Mitnick is, let me give a brief overview. Back in the early '90s he was one of a small group of hackers. I don't mean to say he was a member of a small group, I mean to say that in those days there weren't that many hackers. It was a different world back then, and security was a shadow of what it is now. I've heard a lot of criticism against Kevin along the lines of "all he did was social engineer some people—big deal," and I think this book should put that line of thinking to bed. It's chock full of techno jargon and I'm amazed at the level of detail used to describe hacks that took place 15 years ago. That's not to say there isn't a lot of social engineering going on, because there is, but to say that's all he was good at is not accurate at all.

Starting at an early age with magic tricks, Kevin slowly became obsessed with outsmarting other people. Once he discovered the telephone system, starting with learning ham radio, he had found a way to couple the rush of outsmarting people with his fascination with technology. As he progressed, he began to meet other people with similar interests, which added impetus to his activities. He and his friends soon began to try to outdo each other, which meant pushing the envelope further and further. This eventually led to investigation by the FBI, which caused Kevin to go on the lam. I found this part of the book fascinating—how many people can just relocate to a different state, construct a new identity, and basically start a whole new life? Amazing.

Ultimately, Kevin was captured and incarcerated in 1995, released in 2000, and kept under supervision until 2003. It's an amazing story, and I'm not going to go into the level of detail I would normally offer in a book review, for a couple of reasons. First, I don't want to spoil anything for you, the reader. Second, I'm marginally uncomfortable doing a book review about a person and not a technology. It's Kevin's story to tell, and I'm going to let him tell it. All I can do is try to convince you that you should listen. Whatever your feelings are about Kevin, I can assure you that this book will not be a waste of your time, money, or effort.

Kevin Free.

—Reviewed by Sam Stover

Statement of Ownership, Management, and Circulation, 9/30/11

Title: ;login: Pub. No. 0008-334. Frequency: Bimonthly. Number of issues published annually: 6. Subscription price $125.
Office of publication: USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.
Headquarters of General Business Office of Publisher: Same. Publisher: Same.
Editor: Rik Farrow; Managing Editor: Jane-Ellen Long, located at office of publication.
Owner: USENIX Association. Mailing address: As above.
Known bondholders, mortgagees, and other security holders owning or holding 1 percent or more of total amount of bonds, mortgages, or other securities: None.
The purpose, function, and nonprofit status of this organization and the exempt status for federal income tax purposes have not changed during the preceding 12 months.

Extent and nature of circulation	*Average no. copies each issue during preceding 12 months*	*No. copies of single issue (Oct. 2010) published nearest to filing date of 9/30/11*
A. Total number of copies	5184	4900
B. Paid circulation		
Outside-county mail subscriptions	3118	2981
In-county subscriptions	0	0
Other non-USPS paid distribution	1551	1489
Other classes	0	0
C. Total paid distribution	4669	4470
D. Free distribution by mail		
Outside-county	0	0
In-county	0	0
Other classes mailed through the USPS	77	57
E. Free distribution outside the mail	334	280
F. Total free distribution	411	337
G. Total distribution	5080	4807
H. Copies not distributed	104	93
I. Total	5184	4900
Percent Paid and/or Requested Circulation	92%	93%

I certify that the statements made by me above are correct and complete.
Jane-Ellen Long, Managing Editor

NOTES

USENIX Member Benefits

Members of the USENIX Association receive the following benefits:

Free subscription to *;login:*, the Association's magazine, published six times a year, featuring technical articles, system administration articles, tips and techniques, practical columns on such topics as security, Perl, networks, and operating systems, book reviews, and reports of sessions at USENIX conferences.

Access to *;login:* online from October 1997 to this month: www.usenix.org/publications/login/

Discounts on registration fees for all USENIX conferences.

Special discounts on a variety of products, books, software, and periodicals: www.usenix.org/membership/specialdisc.html.

Contributing to USENIX Good Works projects such as open access for papers, videos, and podcasts; student grants and scholarships; USACO; awards recognizing achievement in our community; and others: http://www.usenix.org/about/goodworks.html

The right to vote on matters affecting the Association, its bylaws, and election of its directors and officers.

For more information regarding membership or benefits, please see www.usenix.org/membership/ or contact office@usenix.org, 510-528-8649.

USENIX Board of Directors

2012 Election for the USENIX Board of Directors

USENIX is a member organization; its vision, focus, and initiatives are centered on serving the needs of its members and their various constituencies. The USENIX Board is responsible for the decision-making and proactive action that best serves the needs of the membership.

The biennial election for officers and directors of the Association will be held in the spring of 2012. People can be nominated for the USENIX board in two ways: either by contacting the Nominating Committee (nomcomm@usenix.org) or by submitting a written statement of nomination signed by at least five (5) USENIX members in good standing.

The Nominating Committee Report is now online at www.usenix.org/about/elections12/elections12nomcomm.html. The report includes a slate of nominees who, in the opinion of the Nominating Committee, would best serve the interests of the organization and the membership.

It is not too late to consider running for the USENIX Board. Nominations from the membership are open until January 6, 2012. To nominate an individual, send a written statement of nomination signed by at least five (5) members in good standing, or five separately signed nominations for the same person, to the Acting Executive Director at the Association offices, to be received by noon PST, January 6, 2012. Please also prepare a plain-text Candidate's Statement of up to 260 words and send both the statement and a 600 dpi photograph to jel@usenix.org, to be included with the ballots.

There are five distinct roles on the USENIX Board. The Board consists of eight directors, four of whom are "at large." The others are officers, including the president, vice president, secretary, and treasurer. Written statements of nomination should specify the role (president, vice-president, secretary, treasurer, or at-large director) to which the nomination pertains.

Ballots will be mailed to all paid-up members by January 31, 2012. Ballots must be received in the USENIX offices by March 12, 2012. The results of the election will be announced on the USENIX Web site by March 30 and will be published in the June issue of *;login:*.

The balloting is preferential: those candidates with the largest numbers of votes are elected. Ties in elections for directors shall result in run-off elections, the results of which shall be determined by a majority of the votes cast. Newly elected directors will take office at the conclusion of the first regularly scheduled meeting following the election, or on July 1, 2012, whichever comes earlier.

—Alva L. Couch, Chair, USENIX Board Nominating Committee

USENIX Remembers Dennis Ritchie (1941–2011)

Our community suffered a serious loss this past October with the passing of UNIX co-inventor and C programming language creator Dennis Ritchie. Dennis was a quiet man who left behind a far-reaching legacy. He was awarded the Turing Award in 1983, the National Medal of Technology in 1999, and the Japan Prize in 2011. While the world and the technical community mourn the loss of a true pioneer, USENIX lost that and much more—Dennis was one of us. He was a frequent attendee at USENIX events for over two decades, a mentor to many, and a friend to all.

If you have a favorite Dennis story, please share it with the community on our Facebook page (https://www.facebook.com/pages/USENIX-Association/124487434386)—and visit our online tribute page (http://www.usenix.org/about/Ritchie.html).

A Note from Dennis's Family

As Dennis's siblings, Lynn, John, and Bill Ritchie—on behalf of the entire Ritchie family—we wanted to convey to all of you how deeply moved, astonished, and appreciative we are of the loving tributes to Dennis that we have been reading. We can confirm what we keep hearing again and again: Dennis was an unfailingly kind, sweet, unassuming, and generous brother—and of course a complete geek. He had a hilariously dry sense of humor, and a keen appreciation for life's absurdities—though his world view was entirely devoid of cynicism or mean-spiritedness.

We are terribly sad to have lost him, but touched beyond words to realize what a mark he made on the world, and how well his gentle personality—beyond his accomplishments—seems to be understood.

USA Team Wins Big at International Programming Competition

The four-student USA team attending the 2011 International Olympiad in Informatics (IOI) delivered an impressive set of results, earning three gold medals and one silver. The IOI, widely regarded as the world championship programming competition at the high school level, was held this August in Pattaya, Thailand, where 302 of the world's best high school programmers from 78 countries tested the limits of their computational problem-solving abilities. The USA was among only four countries earning three gold medals (the others being China, Taiwan, and Croatia), and our point total was exceeded only by China and Russia, making this one of our best overall results ever.

The 2011 USA team includes:

- Wenyu Cao (gold, 6th place overall), from Phillips Academy in Andover, Massachusetts, who is now attending Princeton
- Johnny Ho (gold, 17th place overall), a sophomore from Lynbrook High School in San Jose, California
- Albert Gu (gold, 19th place overall), a senior from Saratoga High School in Saratoga, California, who is now attending Carnegie Mellon University
- Nathan Pinsker (silver, 42nd place overall), a senior from Palo Alto High School in Palo Alto, California, who is now attending MIT

The USA team is selected and trained by the USA Computing Olympiad (USACO), a national organization supporting the advancement of high school computing by identifying, motivating, and training top computing students across the country. The USACO offers six monthly programming contests in three divisions (bronze, silver, gold) throughout the academic year, a set of online training pages that have benefitted tens of thousands of students from nearly 90 countries, and

Left to right: Albert Gu, Wenyu Cao, Brian Dean, Johnny Ho, Nathan Pinsker. *Not pictured:* Deputy leader Neal Wu, who had to leave a day early to compete in the international finals of the Google Code Jam competition in Tokyo.

a rigorous invitational summer training camp for the top 16 students in the USA. USACO is sponsored by USENIX and IBM, and relies on a dedicated team of volunteer coaches for its operation; thanks are due to Jacob Steinhardt, Alex Schwendner, Richard Peng, Jelle van den Hooff, Neal Wu, Eric Price, and Rob Kolstad, as well as to a host of volunteer problem-solvers who help us in preparing our contests.

The problem lineup at the IOI this year was quite challenging, involving several problems that required students to think "outside the box." In addition to a challenging set of programming tasks, participants in the IOI were treated to a rich cultural program, including excursions into Bangkok, demonstrations of Thai martial arts, singing and dance, elephant rides, and authentic Thai cuisine. The entire USA delegation, including team leader Dr. Brian Dean, deputy leader and three-time former IOI gold medalist Neal Wu, and the four team members, all had the experience of a lifetime.

The USACO is indebted to our long-term sponsors, USENIX and IBM. Without the support of forward-looking organizations like these, USACO would never have the means of fulfilling its goals of elevating high-school computing nationwide. There are many alarming trends in high-school computing at the present time: undergraduate CS enrollments are at near 30-year lows, and only 1% of high-school seniors take the AP computer science exam. Our sponsors are among the few organizations that truly recognize that building the foundation for a future workforce to succeed in a high-tech global economy requires dramatic investments to promote excellence in high school computing education.

—Brian Dean, USACO Associate Director

Thanks to Our Volunteers

As many of our members know, USENIX's success is attributable to a large number of volunteers, who lend their expertise and support for our conferences, publications, good works, and member services. They work closely with our staff in bringing you the best there is in the fields of systems research and system administration. Many of you have participated on program committees, steering committees, and subcommittees, as well as contributing to this magazine. We are most grateful to you all. We would like to make special mention of some people who made particularly significant contributions in 2011.

Acting Executive Director
We would like to extend a special thank you to Margo Seltzer, Vice President of the USENIX Board of Directors, for her dedication to USENIX and for taking the reins with enthusiasm and commitment.

Program Chairs

Greg Ganger and John Wilkes: 9th USENIX Conference on File and Storage Technologies (FAST '11)

David G. Andersen and Sylvia Ratnasamy: 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI '11)

Anees Shaikh and Kobus Van der Merwe: Workshop on Hot Topics in management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE '11)

Christopher Kruegel: 4th USENIX Workshop on Large-Scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More (LEET '11)

Matt Welsh: 13th Workshop on Hot Topics in Operating Systems (HotOS XIII)

Michael McCool and Mendel Rosenblum: 3rd USENIX Workshop on Hot Topics in Parallelism (HotPar '11)

Jason Nieh and Carl Waldspurger: 2011 USENIX Annual Technical Conference (USENIX ATC '11)

Program Chairs (continued)

Armando Fox: 2nd USENIX Conference on Web Application Development (WebApps '11)

Ion Stoica and John Wilkes: 3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '11)

Irfan Ahmad: 3rd USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '11)

Sanjay Kumar, Himanshu Raj, and Karsten Schwan: 3rd Workshop on I/O Virtualization (WIOV '11)

Peter Buneman and Juliana Freire: 3rd USENIX Workshop on the Theory and Practice of Provenance (TaPP '11)

David Wagner: 20th USENIX Security Symposium (Security '11)

Hovav Shacham and Vanessa Teague: 2011 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE '11)

Sean Peisert and Stephen Schwab: 4th Workshop on Cyber Security Experimentation and Test (CSET '11)

Nick Feamster and Wenke Lee: USENIX Workshop on Free and Open Communications on the Internet (FOCI '11)

David Brumley and Michal Zalewski: 5th USENIX Workshop on Offensive Technologies (WOOT '11)

Ben Adida and Umesh Shankar: 2nd USENIX Workshop on Health Security and Privacy (HealthSec '11)

Patrick McDaniel: 6th USENIX Workshop on Hot Topics in Security (HotSec '11)

Alex Hutton: Sixth Workshop on Security Metrics (MetriCon 6.0)

Thomas A. Limoncelli and Doug Hughes: 25th Large Installation System Administration Conference (LISA '11)

Invited Talks/Special Track Chairs

John Wilkes: Work-in-Progress Reports (WiPs) and Posters Chair at FAST

David Pease: Tutorial Chair at FAST

Michael Walfish: Poster Session Chair at NSDI

Ajay Gulati: Poster Session Chair at USENIX Annual Tech

Irini Fundulaki: Local Workshop Chair at TaPP

Grigoris Karvounarakis: Workshop Organization and Proceedings Coordinator at TaPP

Sandy Clark, Dan Geer, and Dan Wallach: Invited Talks Committee at USENIX Security

Patrick Traynor: Poster Session Chair at USENIX Security

Matt Blaze: Rump Session Chair at USENIX Security

Æleen Frisch and Kent Skaar: Invited Talks Coordinators at LISA

Cory Lueninghoener: Workshops Coordinators at LISA

Chris St. Pierre: Guru Is In Coordinator at LISA

Matt Disney: Poster Session Coordinator at LISA

William Bilancio: Work-in-Progress Reports (WiPs) Coordinator at LISA

Other Major Contributors

John Arrasjid, David Blank-Edelman, Matt Blaze, Clem Cole, Alva Couch, Brian Noble, Niels Provos, and Margo Seltzer for their service on the USENIX Board of Directors

John Arrasjid, Jeff Bates, Stephen Bourne, Bryan Cantrill, Clem Cole, Æleen Frisch, Dan Klein, Thomas A. Limoncelli, Timothy Lord, Jim McGinness, Brian Noble, and Theodore Ts'o for serving on the USENIX Awards Committee

Brian Dean, Rob Kolstad, Don Piele, Richard Peng, Eric Price, Alex Schwendner, Jacob Steinhardt, Jelle van den Hoof, and Neal Wu, this year's directors and coaches for the USA Computing Olympiad, co-sponsored by USENIX

Dan Geer, Theodore Ts'o, Brian Noble, and Margo Seltzer for serving on the Audit Committee

Alva Couch for chairing the USENIX Board of Directors Nominating Committee

Eddie Kohler for his many cheerful and speedy responses to our requests for customizations of his HotCRP submissions and reviewing system

Jacob Farmer of Cambridge Computer for his sponsorship of the USENIX Education on the Road series and for organizing the Storage Pavilion and Data Storage Day at LISA

John Y. Arrasjid, Ben Lin, Raman Veeramraju, Steve Kaplan, Duncan Epping, Michael Haines, Haythum Babiker, Irena Nikolova, and Kiran Kumar Chittimaneni for writing the two Short Topics books published by USENIX in 2011

Matt Simmons for blogging about USENIX activities

Conference Reports

In this issue:

MULTIMEDIA at USENIX

Did you know that all USENIX conference videos and MP3s are now free and open to the public? Check out the videos and MP3s of these events: http://www.usenix.org/publications/multimedia/

Plus, don't forget to subscribe to the USENIX YouTube channel for the latest conference highlights and greatest hits: http://www.youtube.com/usenixassociation

20th USENIX Security Symposium (USENIX Security '11)

August 8–12, 2011
San Francisco, CA

Opening Remarks, Awards, and Keynote Address

Summarized by Rik Farrow (rik@usenix.org)

Tadoyoshi Kohno (University of Washington), the chair of Security '12, stood in for David Wagner, who was sick, and announced two Outstanding Paper awards: to Clark et al. for "Why (Special Agent) Johnny (Still) Can't Encrypt," and to Caballero et al. for "Measuring *Pay-per-Install*." He also announced that Security '12 would be held in Bellevue, Washington, an edge city of Seattle.

Network Security in the Medium Term: 2061–2561 AD

Charles Stross, Author of award-winning science fiction

Charlie Stross pointed out that, by 2061, networking will have been around about as long as steam engines have been today, but that we ourselves might not be around, having been wiped out by some global kernel panic or a nearby cosmic ray burst. And if we don't solve the energy crisis, we won't have a network to secure—there will be no power.

Stross covered many possible future scenarios. He decried the notion of the AI Singularity, the point of human-equivalent artificial intelligences, saying this was a fantasy akin to a steam-powered tin man. Reading his speech from his iPad (no graphics), Stross spoke eloquently, sometimes dancing closer to his supposed target, network security. I highly suggest listening to the MP3 of his speech on the USENIX Web site.

Stross posited that advances in both computing and bandwidth will allow complete lifelogging. Not only will cameras and microphones record everything we see, background processing will convert printed text and spoken voice into searchable text, and face recognition will identify anyone we come into contact with. Lifelogs would be an incredibly precious resource, one that would require protection, both while being transmitted and then later, when stored.

Stross made another point that I considered very significant in the near term. Currently, service providers cap our data transfers instead of supplying the networking infrastructure that would support practically unlimited access. He said that this expense of data transfer had pushed him into turning his iPhone into a dumb phone. This bandwidth-limiting by today's providers suggests that we need to keep computation local instead of moving masses of data into the cloud for computation (the network infrastructure model). With bandwidth caps, the cloud may remain just as distant as real clouds are to earthbound humans.

Web Security

Summarized by Tamara Denning (tdenning@cs.washington.edu)

Fast and Precise Sanitizer Analysis with Bek

Pieter Hooimeijer, University of Virginia; Benjamin Livshits and David Molnar, Microsoft Research; Prateek Saxena, University of California, Berkeley; Margus Veanes, Microsoft Research

Pieter Hooimeijer presented BEK, a formal language for defining browser input sanitizers and a back-end system for supporting that language. The work is prompted by the inability to make formal determinations about the behaviors of current Web input sanitizers; for example, it is not trivial to determine whether applying a sanitizer twice—or applying two different sanitizers—may result in unsafe output. Specifying a sanitizer via BEK allows one to check whether specific strings (e.g., XSS attack) are potential outputs of the sanitizer. In addition, BEK allows one to check for properties such as commutativity, idempotence, equivalence, and reversibility. The back-end of BEK is supported by a symbolic state transducer model of the sanitizer that can be used to run analysis or generate sanitizer code.

The authors evaluated BEK using 35 currently deployed sanitizers: 76% of tested sanitizers could be ported to the BEK language without modifying the language (90% with multi-character lookahead). The authors found that BEK could check for equivalence between sanitizers in under one minute. Lastly, the authors used BEK to determine whether or not the sanitizers were capable of allowing any XSS attack strings as sanitized output.

During the questions, Hooimeijer clarified that sanitizers were manually ported to BEK, but that the BEK language was designed to resemble the way that current sanitizers are written so that coders will be able to write sanitizers in BEK.

Toward Secure Embedded Web Interfaces

Baptiste Gourdin, LSV ENS-Cachan; Chinmay Soman, Hristo Bojinov, and Elie Bursztein, Stanford University

Elie Bursztein presented work analyzing the security of Web interfaces on embedded devices and subsequently developing a framework for secure Web interfaces. Bursztein discussed the prevalence of Web interfaces for customization of consumer electronics such as routers, printers, VoIP phones, and digital photo frames: there are at least twice as many Web interfaces on embedded devices as there are traditional servers hosting Web sites. Additionally, these Web interfaces tend to be custom-developed on a tight deadline and feature-driven, leading to many vulnerabilities. The authors audited the security of over 30 devices from a variety of brands and categories, and found vulnerabilities in all devices tested.

In an effort to improve the bottom line of security for embedded Web interfaces, the authors developed WebDroid, a framework built on Android for providing secure embedded Web interfaces. More specifically, WebDroid protects against the vulnerabilities revealed in the motivating security audits. The authors performed benchmark testing to evaluate the performance of WebDroid's security features, and found that WebDroid has a 10–15% loss in performance (requests per second and process time of requests) when using security features.

During questions, Bursztein clarified that most of these embedded devices could use WebDroid after installing Android, since they mostly have ARM processors. Bursztein was also asked whether the team looked at open source router firmware such as DD-WRT and OpenWrt; they did not.

Zozzle: Fast and Precise In-Browser JavaScript Malware Detection

Charlie Curtsinger, University of Massachusetts Amherst; Benjamin Livshits and Benjamin Zorn, Microsoft Research; Christian Seifert, Microsoft

Benjamin Livshits presented Zozzle, a low-overhead in-browser method for detecting malware. Zozzle, which does static, online analysis can be contrasted with Nozzle, which performs offline runtime analysis to detect heap sprays. Zozzle detects JavaScript malware via machine learning;it was trained using one thousand malicious samples and seven thousand benign samples. Zozzle uses hierarchical features and Naive Bayes classification; it deals with obfuscated code by unfolding the code using the JavaScript runtime in the browser, then reclassifying it.

When using 300 features, Zozzle has a throughput of 1 MB of code per second. When run over 1.2 million samples of JavaScript code, this resulted in four false positives and a false negative rate of 9%, both of which are comparable to the results given by antivirus engines.

One person asked if it is possible for an attacker to overwrite Zozzle's weight table in order to avoid detection; Livshits answered that this is a possibility but that Zozzle provides the benefit of online analysis for sites behind paywalls and other similar situations, due to its in-browser nature. In answer to another question, Livshits said that the team compared the ongoing results of Nozzle against Zozzle, and found that Zozzle identifies new malware threats before they are encountered by Zozzle. In another answer, Livshits clarified that Zozzle identifies threats beyond heap sprays.

Invited Talk

The Three Cyber-War Fallacies

Dave Aitel, CEO of Immunity, Inc.

Summarized by Nathaniel Husted (nhusted@indiana.edu)

Dave Aitel defined the three fallacies in the current understanding of cyberwarfare as being that cyberwar is asymmetric, non-kinetic, and not attributable. He gave examples of these fallacies from sites such as *The Economist* and CNAS. However, the Pentagon has defined "cyber" as a new warfare domain, thus making it a fact that can't be ignored, and modern hackers are now part of this domain.

Dave Aitel first attacked the fallacy that cyberwar is non-kinetic. The term kinetic, in this sense, is used to refer to bombs, ammunition, and other objects causing physical damage. For example, disabling a smart grid or the water pumps of New Orleans would have dire physical consequences. Another example of the kinetic nature of cyberwar is that it can change nation-state behavior. For example, sites like WikiLeaks can affect the policy and actions of a country as large as the United States. Also, considering the number of Fortune 500 companies that are most likely compromised in some way, shape, or form (Dave suggested many might not even know), it would be possible to affect their supply chain.

As for the second fallacy, attribution happens all the time in cyberwar. Dave mentions articles from McAfee, *The Economist*, and a number of other news sources. That organizations from China commenced "Operation ShadyRat" has been published in a large number of publications after McAfee's original statements. Such declarations lead to attribution.

The final fallacy is that cyberwar is asymmetric. In this case, Dave discussed the cost of both attacking and defending. The popular view is that attacking is cheap while defending is very expensive. The phrase, "An attacker only needs to find one hole while a defender has to defend many," is a prime example of this. But creating a worldwide strike capability not only requires a large network of computers but also the ability to maintain a large amount of up-to-date information regarding targets and exploits. Such a large network incurs large costs, and only an enormous organizations, like Google, Amazon, or the US government, has this sort of computing infrastructure. Moreover, creating a 0-day exploit for a piece of software takes roughly 450 hours, according to metrics obtained by Dave's company, Immunity. It takes 18 hours to run a modern exploit against a machine. If an exploited machine is discovered by the defender, i.e., cleaned up, the attacker must also assume that all their information on this machine is compromised and they must start over.

Defending against attackers is also cheaper than we have come to believe. Aitel says that the attackers are winning because they have a much better strategy. Defenders are hampered by the culture. For example, law enforcement is very successful against hackers with economic motives, but very bad about deterring anyone without a financial motive. The academic community is not a serious player in this area: many of their discoveries do not keep pace with reality. Defenders also continue consistently to underestimate their attackers' abilities. Finally, defenders more often than not continue to use software with serious vulnerabilities. Dave asked, "How many issues do you have to come up with before your company will stop using a product?"

Rik Farrow wondered about the ability of organizations to avoid using insecure software, as all software has some insecurities. Dave answered that there are relatively secure options, such as Google Chrome in the browsing market, for example, but companies don't choose them. How can organizations that completely misunderstand cyberwar use this new information to change their strategies? One of the biggest things they can do is run new purchases and products through a security team. If the security team says it isn't secure, don't use it or release it. Carson Gaspar (Goldman Sachs) said that businesses think that being secure is more expensive than being insecure and asked how this relates to Dave's talk. Dave replied that, viewed on a quarterly basis, they may be right. However, in the long run not being secure costs far more. Adam Drew (Qualcomm) asked Dave what advice he'd give to help students in academic research become more effective in this area. Dave replied that they need to be taught to think like attackers, but it's complicated. Many attackers are "crazy people" who have ingrained characteristics that make them very skilled at what they do but are not easily taught (or managed). However, he also said there are good people in academia doing good work.

The slides for Dave Aitel's talk are available at http://prezi.com/vunircise2q8/three-cyber-war-fallacies/.

Analysis of Deployed Systems

Summarized by Shane Clark (ssclark@cs.umass.edu)

Why (Special Agent) Johnny (Still) Can't Encrypt: A Security Analysis of the APCO Project 25 Two-Way Radio System

Sandy Clark, Travis Goodspeed, Perry Metzger, Zachary Wasserman, Kevin Xu, and Matt Blaze, University of Pennsylvania

➘ *Awarded Outstanding Paper!*

Matt Blaze presented this security analysis of the APCO Project 25 (P25) radio system. P25 is a digital radio standard used by law enforcement groups and the US Secret Service. It provides a radio system that is backwards compatible with existing analog solutions while also supporting encryption for sensitive communications. Blaze and his colleagues identified vulnerabilities in P25 radios, including susceptibility to tracking attacks and efficient denial of service. They also found that legitimate P25 users often unknowingly transmit in the clear because of usability issues that make it difficult to verify when encryption is in use.

Both active and passive tracking attacks are possible. Active attackers can "ping" a radio with a malformed frame to which it responds whenever in range, without the victim's knowledge. Radios can be passively tracked while in use, because they transmit a unique ID in the clear with each message, though the protocol specifies an option to encrypt the ID. Denial of service attacks can be launched with a 14 dB energy advantage given to the attacker. By jamming only a 64-bit subfield, an attacker can render an entire 1728-bit voice frame unreadable. The researchers prototyped a jamming device using a $15 child's toy. While a realistic attack would require an amplifier, the prototype highlights the simplicity of the attack.

Finally, Blaze addressed usability issues and mitigation techniques. The radios tested give users little feedback about whether outgoing traffic is being encrypted, and also demodulate and play any incoming traffic without giving the user an indication of whether the traffic is encrypted. The over-the-air rekeying protocol that the radios use also fails regularly, forcing users to communicate in the clear until their radios can be rekeyed successfully. The researchers purchased hardware to measure the sensitive voice traffic transmitted in the clear in several metropolitan areas. They observed an average of 20 minutes of sensitive cleartext per city per day. This sensitive cleartext included information such as confidential informant names. Eavesdropping attacks could be mitigated by using the over-the-air rekeying protocol less frequently and by preventing unencrypted voice traffic from mixing transparently with encrypted traffic.

Questions from the audience addressed government reaction to the research and further details about mitigation. Blaze responded that the researchers approached the government "very politely" and that the government employees they interacted with all understood that identifying an attack was not equivalent to launching one. He also pointed out that all of the passive attacks they identified could be effectively stopped by improving user awareness of radio state. According to Blaze, however, the active attacks that the researchers identified are fundamental to the protocol and require a redesign to mitigate effectively.

Dark Clouds on the Horizon: Using Cloud Storage as Attack Vector and Online Slack Space

Martin Mulazzani, Sebastian Schrittwieser, Manuel Leithner, Markus Huber, and Edgar Weippl, SBA Research

Martin Mulazzani presented this work on cloud storage system vulnerabilities. Mulazzani and his collaborators identified three attacks, all of which they launched successfully against the popular Dropbox cloud storage system.

The first vulnerability takes advantage of Dropbox's use of SHA-1 hashes for data deduplication. If a file hash already exists in the system, then the file is linked to the account rather than uploaded. An attacker can thus check for file existence or get a copy of a file, assuming knowledge of its SHA-1 hash. This attack is applicable to any cloud storage system that implements client-side data deduplication without requiring a client-side data possession proof. Attackers can also steal entire Dropbox folders if they steal a user's "Host ID," which is a unique credential used for authentication at setup time. It is stored in cleartext on the user's computer. Finally, an attacker can upload unlimited data not linked to an account by taking advantage of a vulnerability in the upload/download system. The data will eventually be reclaimed as garbage, but were reliably available for at least four weeks, according to Mulazzani's experiments. The researchers suggested several techniques to prevent the data deduplication attack by requiring client-side proofs. Since the researchers notified Dropbox of these attacks, the company has removed data deduplication, fixed the upload/download system vulnerability, and encrypted the Host ID. Mulazzani noted that the plaintext Host ID is still resident in RAM, so the folder stealing attack is more difficult, but not impossible.

During the Q&A, Ian Goldberg suggested that one approach to data deduplication is to challenge the client to compute a MAC of the file. Mulazzani agreed, but speculated that it might be slower in the case that the file already exists. Mark Seiden (Yahoo!) asked if the researchers had verified that

Dropbox computes the SHA-1 hash at the server for each upload. Mulazzani confirmed that the system does so.

Comprehensive Experimental Analyses of Automotive Attack Surfaces

Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage, University of California, San Diego; Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno, University of Washington

Cars are increasingly complex systems that typically contain 10+ electronic control units (ECUs), embedded systems. In this work, the researchers extracted firmware from several ECUs, reverse-engineered it to identify vulnerabilities, and finally created a series of attacks, all of which give the attacker complete control of the vehicle (brakes, engine, locks, etc.) without requiring physical access.

Stephen Checkoway described attacks using the car's media player, Bluetooth interface, and telematics unit (used for systems such as OnStar). The media player attack used a specially crafted WMA file, the Bluetooth attack used a trojaned Android phone, and the telematics attack could be triggered via the audio in a phone call. Checkoway played a video demonstrating a compromise via the telematics unit in which a remote researcher was able to unlock the car, disable the anti-theft system, and start the engine, allowing an onsite researcher to simply drive the car away. A second video demonstrated surreptitious tracking and audio recording via the onboard GPS and telematics systems. Checkoway attributed the proliferation of vulnerabilities mainly to a lack of past adversarial pressure and the heterogeneous multi-vendor development of modern car systems. Almost all the bugs they found appeared at component boundaries, often through incorrect assumptions made by suppliers. Finally, Checkoway said that relevant stakeholders such as SAE, USCAR, and the US Department of Transportation have been notified of the vulnerabilities and are taking action in response.

Bill Cheswick asked if modern military gear used the same equipment. Checkoway said that he did not know. J. Alex Halderman asked if a monoculture might be worse for security than the heterogeneous status quo. Checkoway responded that he is not sure and clarified that the researchers did not mean to suggest that one vendor should make all systems, but that fewer vendors should be used for each car. Mike Ryan (ISI) asked if the researchers could steer the car remotely. Checkoway said that they could not steer the car and did not test acceleration because of the risk involved. Rik Farrow asked how widespread the compromised telematic unit is. Checkoway answered that they only tested one make and model so he is unsure, but his understanding is that each manufacturer uses at least one unique telematics unit.

Forensic Analysis

Summarized by Lakshmanan Nataraj (lakshmanan_nataraj@umail.ucsb.edu)

Forensic Triage for Mobile Phones with DEC0DE

Robert J. Walls, Erik Learned-Miller, and Brian Neil Levine, University of Massachusetts Amherst

A typical crime scene investigation includes lots of digital evidence such as computers, mobile phones, etc., and it takes law enforcement agencies quite a while to extract data from these devices. In order to acquire evidence quickly and on-scene, Robert Walls proposed a system called DEC0DE for forensic triage of mobile phones. The authors chose mobile phones instead of computers since phones are not only ubiquitous but also contain key information (address books, images, etc.). For this work, the authors specifically dealt only with feature phones, which account for 60% of the phones used in the US. This system extracts digital information directly from the phone storage in, at maximum, around 20 minutes. The interesting point to be noted is that this system is agnostic to the file system and operating system of the phone. This is important, since it allows the possibility of handling phone models that have not been previously seen without any extra work (critical for triage).

The input to the system is raw storage (stream of bytes) from a phone. In order to remove unwanted bytes that need not be parsed, the raw storage is first filtered using a technique called block hash filtering (BHF), which preserves important fields such as timestamps and phone numbers. The system later locates these fields and interprets a combination of fields as a record. As the name suggests, BHF is carried out by dividing the storage into blocks and computing a hash on every block. Duplicate blocks are filtered out by comparing the hashes against a library of known hashes so that only important data is retained. Experimental evaluations on different phone models show that this filtering helps in removing lots of extraneous data (69% on average). There was also a lot of overlap between phones of the same model.

Once the filtering step is completed, the next step is inference. The system uses machine learning algorithms for this step, with the assumption that similar phone models have similar call logs. The formats are encoded using probabilistic finite state machines and then parsed using a dynamic programming algorithm (Viterbi). The state machines differ depending on the record of consideration. In the end, a decision-tree-based classifier helps to remove false positives. The whole system was evaluated by manually selecting known models from different manufacturers and known records and verified on models that closely match the former. Around 93% of the records were recovered. The main limitation of this system is that the authors assume raw storage can be acquired, which itself is a great challenge. Another limitation is that success depends on the quality of the state machine.

In Q&A, Rik Farrow mentioned that often service providers can transfer contacts from an old phone to a new phone with ease, so is it not a better idea to use those same tools? Robert replied that even their own tools don't do a good job. He illustrated this by describing how his lab mate had the service provider transfer contacts for him, but the process only transferred a small fraction of the contacts; his system, however, was able to extract all the contacts without any changes to his system. Ben Fuller asked about the poor performance working with two LG phones. Robert answered that the system could get better as more phone models become available.

mCarve: Carving Attributed Dump Sets

Ton van Deursen, Sjouke Mauw, and Saša Radomirović, University of Luxembourg

Sjouke Mauw started off by saying that there is a general feeling that MIFARE transportation cards can be easily hacked. In this work, the authors verified this by finding vulnerabilities on a Luxembourg-based transportation card called the e-go card. To kick-start the project, they first used standard data carving tools from digital forensics, which gave them a lot of dumps along with attributes such as identity of the card, purchase date, and number of rides left. They then posed a research question: given a series of dumps with many attributes, is it possible to map a dump attribute to the set of dumps? They made some strong assumptions, though, such as equal length dumps, same location of attributes in every dump, and that the encoding of an attribute is the same or deterministic. Although these assumptions seem very strong, they are acceptable given that they are dealing with dumps of transportation cards.

Sjouke discussed strategies for finding all possible mappings of an attribute to a dump set. The first strategy is based on commonalities. It is done by XORing the set of dump bits and finding the common indices. The second strategy is a little more complex and is based on dissimilarities. The whole methodology was validated by making several trips on a bus and manually noting known attributes such as date, rides left, etc. On applying the above algorithms, several attributes were found and matched with the manually noted set. And all this took only a few seconds per card. Sjouke concluded with future work such as automatic encoding and better algorithms to improve robustness. The current version of the tool is open source and available for download from http://satoss.uni.lu/mcarve.

Do the bits need to be consecutive for the dissimilarity algorithm to work? The consecutive bits were only shown for illustration and are not required by the algorithm. Someone asked about applications besides transportation cards and what knowledge was needed by the system about a card. Sjouke said they need contextual information about what is represented in the bit stream. Another application could be protocol reverse engineering, for which memory dumps are not needed but the communication can be inspected.

ShellOS: Enabling Fast Detection and Forensic Analysis of Code Injection Attacks

Kevin Z. Snow, Srinivas Krishnan, and Fabian Monrose, University of North Carolina at Chapel Hill; Niels Provos, Google

Code injection attacks are one of the most common methods of gaining control over a computer. Readily available exploit kits are making it very easy to deploy exploits. At a higher level, a code injection attack transfers the application control flow to a code supplied by the attacker. In most cases, though, it transfers the control to a shell code regardless of the method of exploitation.

Detecting the shell code itself aids in subverting a code injection attack. Lightweight emulation based on dynamic analysis is the state of the art in detecting shell code, but it is very slow. Emulators have also proven to be detectable, which makes lightweight emulation-based analysis weak. Kevin Snow described how the authors designed a new, faster, and more efficient dynamic analysis technique which is not based on emulation. For this sole purpose, they built an operating system, called ShellOS, that executes code streams. The entire OS consists of approximately 2500 lines of code written in C and assembly code.

The most interesting and useful part of this system is that it can run on a standard OS as a guest OS using hardware virtualization. When ShellOS first boots, it creates a suitable environment to execute shell code by allocating memory given by some user-supplied process memory snapshot. The host OS then supplies the ShellOS with a code stream to analyze through a shared memory region, which triggers ShellOS to execute the code stream from every triggered offset. ShellOS observes faults and timeouts in the code. In order to trace these to memory, ShellOS catches page faults at memory addresses that are defined by the above heuristics. To determine the effectiveness of their system, they conducted some experiments on throughput and detectability and compared them against the state-of-the-art shell code detection system called Nemu. The experiments showed ShellOS to be faster and more efficient than Nemu.

They did a case study on a real world scenario where shell codes are injected in PDF files. They used 374 suspicious PDF documents provided by Google that were collected between 2007 and 2010. They also had a benign set to test false positives, which consisted of 179 PDFs from previous USENIX conferences. The system was initially able to detect 325 PDFs from the malicious set and the remaining were detected after unpacking. None of the PDFs from the benign set was flagged as malicious. Although ShellOS is fast and detects shell code effectively, it does have some limitations. First, it is not easy to extract shell code. Second, hardware virtualization may also be detectable. However, future versions of ShellOS may not need hardware virtualization. The authors plan to release the source code of ShellOS soon.

What would be the effect if the shell code invokes an API call that affects the external environment, such as file manipulations or process creation? Is another process created in the virtual machine? The results of the API calls are simulated within ShellOS, and process call creation is not currently supported. The shell code will still be detected but their system will not be able to follow it in the diagnostics. What if an attacker has access to their system and keeps tweaking his code till the system does not detect it? That would certainly be a problem, as with all other approaches.

Invited Talk

Crossing the Chasm: Pitching Security Research to Mainstream Browser Vendors

Collin Jackson, Assistant Research Professor at Carnegie Mellon University

Summarized by Mihir Nanavati (mihirn@cs.ubc.ca)

Collin Jackson addressed some of the reasons why very few ideas proposed in academia for increasing browser security ever get adopted by the browser community. He discussed some of the fundamental differences in the goals of academia and publishing papers, and those in building browsers for mass market adoption. Using real-world illustrations, he also gave some "rules" to keep in mind to try to increase the possibility of getting a feature added to a browser.

Jackson started by quantifying what crossing the chasm to mass market adoption really means. Jackson argued that even very popular add-ons such as NoScript are only used by a small fraction of the total user base. To really make an impact, a feature needs to make it into the browser's main code. The easiest way to do this is to be picked up by at least one of the major browsers, which is usually followed by adoption by the other browser vendors.

Getting a few people interested in an idea is easy, and even getting several technically oriented early adopters on board is not too hard a feat. It is at the next stage, that of making it acceptable to ordinary users, that most flounder. This is a fundamental difference between browser add-ons and the core engine—add-ons are aimed at power users. Features that require extensive user interaction or break a large proportion of the Web are perfectly acceptable for add-ons but are impractical for core adoption.

While it is tempting to attribute lack of adoption to the inertia or general laziness of browser vendors, this is unfair. Browser vendors can move very quickly if the feature meets certain criteria and is deemed necessary to the community—clickjacking defenses like x-frame-options were implemented in every major browser in under two years, while history privacy has largely been adopted in under a year.

Jackson then considered the differences between ideas that had successfully been adopted and those that had failed to make the cut. Generally, browser vendors tend to favor small, simple features that fix something that is badly broken. Ones that can be implemented in a verifiable way, preferably across multiple browsers, and don't break existing Web pages are preferable. This is in contrast to academia, which tends to reward novel ideas that open new avenues for research and often involve complex and significant implementations.

He outlined a set of general guidelines on how best to select ideas for browser adoption. Ideally, features should attempt to make themselves indispensable by solving a real world problem, especially those that are receiving a fair degree of media coverage. Getting adopted by a single Web browser and championed by large Web sites are good ways of getting noticed. Same-origin policy is an example of such an indispensable feature, as is PostMessage, which was originally introduced by Opera to allow different browser windows to communicate without making a round trip to the server.

Second, sometimes even imperfect solutions may be preferred over more complex solutions if they are easily implementable in a standard way and can be deployed unilaterally. Features that require cooperation from Web sites often break a lot of functionality, since Web sites are very slow to react to changes in browsers.

Finally, low-risk proposals have the best chance of adoption. Generally, people are annoyed when Web sites do not load, and rather than understanding the reasoning behind it or filing a bug report, they tend to just switch to another browser. If it is imperative to break functionality, Jackson strongly suggested minimizing the impact by analyzing how necessary the break was, and whether the feature could be made opt-in and Web sites gradually be persuaded to use it.

Switching back to the theme of academia vs. industry, Jackson noted that feature evaluations in research tended to be far short of what is expected in industry. Simply verifying the front page of sites on the Alexa Top 100 or that the browser can still play YouTube videos is completely insufficient. Evaluations involving a deep crawl of the Web site and using client-side measurements are far more likely to reveal compatibility problems due to the addition of a feature.

Jackson then asked whether there was any point in pursuing bold and complex solutions, since the probability of them ever getting mainstreamed is so low. He concluded that even if such ideas are never mainstreamed, they push the boundaries of research and could be successful even if only a very small subset of the entire system they proposed gets adopted.

The questions revolved around whether browser vendors were being too conservative and trying to protect users too much. Couldn't Web browsers just leave the decision of trusting a Web site or not to the user? Jackson explained how Web developers would like to provide users with rich functionality, often using JavaScript, regardless of the user's trust perception of the Web site, making sandboxing and protection in Web browsers necessary. Jackson was then asked if a way to improve research evaluations would be to release better benchmarks to the community. While wholeheartedly approving the idea, he had doubts about whether this was possible, due to copyright issues. The session concluded with the observation that there was a disconnect between developers and users, and that a feature that required any amount of effort from a user would be unpopular and likely to be turned off. For this reason, the importance of keeping features simple cannot be overstated.

Static and Dynamic Analysis

Summarized by Samee Zahur (sza4uq@virginia.edu)

MACE: Model-inference-Assisted Concolic Exploration for Protocol and Vulnerability Discovery

Chia Yuan Cho, University of California, Berkeley, and DSO National Labs; Domagoj Babić, University of California, Berkeley; Pongsin Poosankam, University of California, Berkeley, and Carnegie Mellon University; Kevin Zhijie Chen, Edward XueJun Wu, and Dawn Song, University of California, Berkeley

Domagoj Babić introduced their new tool for dynamic symbolic analysis, MACE. Although many companies already use such tools for some of their software development projects, he noted that testing remains the most widely used means of weeding out software bugs and vulnerabilities. While most existing automated tools do not remember anything from one iteration to the next, MACE improves on this by learning an approximation of the application's state space and then using that approximation to guide further search. Its effectiveness was demonstrated particularly well for programs implementing various network protocols, such as Vino and Samba, where their abstract execution pattern naturally fits into the finite automaton structure used by their models here.

In the rest of the presentation, Domagoj outlined how dynamic symbolic analysis normally works, and how their approach differs. Normally, the process starts by running through the program with some concrete input sequence and collecting a trace of every single branch condition. This produces a set of constraints that the input sequence must satisfy in order to follow through the same path in the program. Other paths are then explored by negating the last constraint of each prefix of each constraint sequence. Their procedure, however, takes an additional input from the user: an output abstraction function. This groups all program outputs into coarse-grained categories or abstractions, and this is what determines the quality of the learnt program model. Using this, and a variation of the L* learning algorithm, they were able to deduce the sequences of inputs that cause significant program state changes. Thus MACE would iteratively build up a deterministic finite automaton (DFA) modeling internal state changes of the program, and also produce exact sequences of inputs that will cause transitions between the states. As new states and transitions of this DFA are discovered, they are fed back into the learning algorithm. Further exploration is then guided by this model, as MACE can now use the known input sequences for transitioning between states to start further exploration at any given state. It can also filter out redundant input sequences that cause the same transitions, making the analysis more tractable.

This enabled them to find a number of vulnerabilities in network programs, ones Domagoj called "deep vulnerabilities," that is, those hard for an unguided analyzer to find. He explained this by showing a graph that demonstrated how an unguided analyzer quickly loses its ability to explore deeper states. At the end, however, one of the audience members noted that the improvement in code coverage over an unguided search seemed to vary—6.5% for Vino versus 59% for Samba—and asked why it was so. Domagoj conjectured that since Vino implemented a simpler protocol, it is likely that the baseline was already able to explore a large part it. Session chair Sam King asked what was the most surprising discovery they made during development. Domagoj answered, "It was surprising to see that it works!".

Static Detection of Access Control Vulnerabilities in Web Applications

Fangqi Sun, Liang Xu, and Zhendong Su, University of California, Davis

Fangqi Sun presented their work on static analysis of Web sites to detect access control violations. Even large companies like Bloomberg and Disney often have such vulnerabilities on their Web sites, where they make implicit assumptions about access control policies and simply forget to place guards on sensitive Web pages. This often allows an attacker to gain access to restricted parts of a Web site by just typing a URL in a browser. She pointed out that frequently used methods of code review are neither comprehensive nor efficient, and automated detection of access control vulnerabilities is often hard in the absence of formal specifications. Their approach, instead, was to automatically explore hyperlinks produced by PHP scripts to produce role-specific sitemaps. Once they obtain such sitemaps for normal users, sysadmins, etc., their tool can automatically attempt to explore pages that should be accessible by one role and not another (e.g., by sysadmins and not normal users). If it succeeds, the tool flags a vulnerability.

Rationales behind various design choices were given. For example, Fangqi described how static analysis provided better code coverage compared to dynamic analysis techniques, but also required the use of context-free grammars to approximate various links produced by PHP scripts. The presentation ended with evaluations and some limitations of their tool. It was able to find vulnerabilities in both traditional and Web 2.0 applications. The evaluation also demonstrated the usefulness of a specialized tool: it can scan through 12,000 lines of code in two minutes.

Someone asked how their tool explores privileged pages not found in the code. Fangqi said developers can manually specify additional Web pages to explore. Session chair Sam King asked how dynamically generated links were being handled. Fangqi said that JavaScript-generated links are indeed a limitation, as she had already pointed out, and they intend to address it in the future.

ADsafety: Type-Based Verification of JavaScript Sandboxing

Joe Gibbs Politz, Spiridon Aristides Eliopoulos, Arjun Guha, and Shriram Krishnamurthi, Brown University

As advertisements and other mashup components in today's Web applications often require the use of third-party JavaScripts, they also require sandboxing to provide safety guarantees. Arjun Guha said that the authors' work tries to verify safety properties of various trusted sandboxing libraries often employed, for which they specifically focus on Yahoo!'s ADsafe library as a typical example. Arjun showed how common such third-party scripts are, how hard it is to verify sandboxing libraries that are trusted to provide safety, and how even a single mistake can provide attackers with the upper hand. The rest of the presentation consisted of the details of how a type-based static checker can provably guarantee the safety of a sandboxing library, and how they used it to verify the ADsafe library.

Arjun described, one by one, how their type system guarantees each of the claimed safety properties, e.g., not being able to load arbitrary code at runtime, not being able to affect DOM outside a designated part, etc. One of their key observations was that ADsafe already requires JavaScript codes to pass an existing static checker, JSLint. They could therefore design their type system to be a superset of everything JSLint accepts, allowing them to significantly simplify their design of the type system.

Arjun ended with several bugs they found in ADsafe with their automated system, as well as a bug in JSLint. Sam King asked whether we should move to a better language that is easier to check, or to a subset of JavaScript. Arjun answered that JSLint and other static checkers already do require code to be rewritten in a restricted subset of JavaScript, to the point where it is almost a new language. David Evans (University of Virginia) noted that the use of a keyword whitelist (as opposed to the keyword blacklist of identifiers, as used here) may be more effective in filtering unsafe attributes. Since browsers are adding new features and keywords all the time, some of them may be unsafe. Arjun replied that if the external environment cannot be relied on, even a whitelist-based filter can be defeated, pointing out that redefinition of built-in keywords by the hosting page is always a problem. A questioner sought clarification on which bugs were empirically found and which proven by their type system. Only the JSLint bug was empirically found.

Invited Talk

I'm from the Government and I'm Here to Help: Perspectives from a Privacy Tech Wonk

Tara Whalen, Office of the Privacy Commissioner of Canada

Summarized by Julie Ard (julieard@gmail.com)

The Canadian Office of the Privacy Commissioner (OPC), established in 1983, has a mandate to oversee compliance with the Canadian Privacy Act, covering governmental privacy, and its expansion in 2000 to protect and promote the privacy rights of individuals. The OPC acts as an ombudsman. Its powers include investigation, audit, and the ability to pursue court actions, publicly report on information handling practices, promote public awareness, and to support research on privacy issues (they awarded $350,000 in grants last year for privacy research and public education projects). The Technology Analysis Branch in the OPC supports investigations, among other duties. Technologists undertook two major technical investigations which Tara discussed in detail, emphasizing the importance of government employees and investigators having technical expertise. One investigation concerned Facebook's privacy policies and possible conflicts with Canadian privacy law. The second investigation was into Google's inadvertent collection of data from WiFi networks while taking pictures for their Street View service.

Data protection authorities exist in over 40 countries (predominantly in Europe). The most similar governmental organization in the US is the Federal Trade Commission. A very active discussion included questions regarding how both Facebook and Google responded to the OPC's complaints. Over 30 countries were involved in the Google complaint. Many simply requested that data associated with their citizens be deleted. Some (including Canada) requested access to the data so that they could perform their own investigation. Canada's investigation was performed on-site; copies of the data in question were not made. American lawsuits are ongoing. A question from the audience initiated a discussion about secure deletion and technical assurance.

Another member of the audience observed that companies tend to push the line on privacy, characterizing Facebook's privacy policies as a moving target and suggesting that they tend to beg for forgiveness rather than ask permission. For example, the Facebook CEO stated publicly that our notions of privacy are obsolete. The audience recognized that the choices these companies make affect society and privacy standards. Governmental data protection authorities expend vast resources investigating, and making complaints and recommendations. This process may take several months to a year, depending on complexity. The OPC does not think that it is a losing battle or a "done deal" that privacy is obsolete.

Someone asked whether it's possible to use location services for oneself but not share that information with Big Brother."One can disable location services altogether, but it would be more desirable for the individual to choose which applications can use location data. This preference can vary based on the application. For example, in the US government employers do want information from BlackBerry to track their government employees. A member of the audience asked whether that happens in Canada; Tara said that such actions are covered by established legislation.

Another topic presented was that of lawful intercept based on a case study covered in the presentation of a German who published his own mobile data in order to see what could be determined from that data. A discussion ensued on the topic of lawful intercept for law enforcement and national security. In Canada, numerous bills have been proposed but none have been passed. Someone asked about data crossing borders into the US, for example, where organizations are required to retain Patriot Act data. Tara said that to established legislation covers those situations.

Tara encouraged researchers to make their work presentable, visualizable, and accessible so that it can influence the world of politics. Evidence is vital for informing the policy debate. She applauded the creation of tools like Tor to empower citizens. Canada's role in establishing the facts of the Google and Facebook cases heavily influenced their outcomes: Facebook's initial reaction was to implement Canada's requests, and Google's press release essentially mirrored elements of Canada's complaint by promising to implement changes that the OPC suggested. Tara reiterated that these debates have the power to shape company policies, as evidenced by Google and Facebook's reactions to the complaints, and that Canada values its role in fostering global cooperation.

Understanding the Underground Economy

Summarized by Robert Walls (rjwalls@cs.umass.edu)

Measuring **Pay-per-Install***: The Commoditization of Malware Distribution*

Juan Caballero, IMDEA Software Institute; Chris Grier, Christian Kreibich, and Vern Paxson, University of California, Berkeley, and ICSI

Chris Grier began the session with an in-depth look at the ecosystem that has built up around pay-per-install (PPI) services. PPI services provide a way for clients to quickly install their malware on a large number of pre-compromised hosts by simply purchasing installs from these services. To measure the PPI ecosystem, they infiltrated four programs and set up a number of hosts, in geographically diverse locations, to automatically download the malware provided by the PPI service. This allowed the team to perform real-time monitoring, infer the types of clients using PPI services, and estimate the financial impact of a botnet takedown.

They drew three major conclusions from their study. First, they found that PPI services are popular: 12 of the 20 most common malware families are at least partly distributed by PPI services. Second, they found that malware regularly performs repacking to avoid detection, every 11 days on average. Third, clients target specific geographic locations for their malware, resulting in differing demand and therefore different install rates for each country. Chris attributed this demand to the client's ability to monetize their malware in each particular country. For example, their measurements indicate that spambots tend to be installed uniformly across different countries, while click-fraud binaries largely focus on Western countries. Finally, Chris mentioned that they observed instances of PPI arbitrage, where individuals would exploit price differences between PPI providers by buying installs from one provider and selling them to another.

Dan Farmer asked if someone could exploit the PPI system by cheaply acquiring virtual hosts through cloud services, selling them to the PPI providers, and then turning off the hosts. Chris replied that this is probably possible, but the PPI services will eventually detect the abuse and withhold payment. One audience member asked how the volume of PPI installation of malware compares to other distribution mechanisms. Chris said they are working on expanding their study to include this data, but they do know that most of the popular malware uses multiple installation vectors. Another attendee inquired about the specific payment mechanisms. Chris explained that most programs advertised payouts using WebMoney and some mentioned PayPal. Finally, Jelena Mirkovic (ISI) wanted to know about the implications for researchers. Chris suggested that researchers should be aware of how malware fits in the PPI ecosystem. He reiterated that malware using PPI services may not include any infection mechanisms. Anecdotally, Chris said they found PPI loaders that were misclassified as another family of malware. Such loaders might exhibit different behavior each time they are run. Co-author Vern Paxson helpfully added to Chris's comments by pointing out that the PPI loaders can be used as a source to acquire new samples of malware.

Dirty Jobs: The Role of Freelance Labor in Web Service Abuse

Marti Motoyama, Damon McCoy, Kirill Levchenko, Stefan Savage, and Geoffrey M. Voelker, University of California, San Diego

Marti Motoyama continued the session with his work on the role of freelance labor in abusing Web services. He argued that scammers, spammers, and other Internet denizens can leverage the large labor pool provided by sites such as freelancer.com and Amazon's Mechanical Turk to cheaply and effectively abuse free Web services. Marti used Gmail and spamming as an example, pointing out how one can use outsourced human labor to circumvent many of Gmail's technological protections against creating bulk accounts. They estimate that about 30% of the jobs on freelancer.com are abusive. Marti primarily covered three different job types commonly submitted to freelancer.com: account registration, online social network linking, and search engine abuse. For each of these types, he commissioned his own job on freelancer.com to measure the quality of the workers' responses.

The goal of the first job type, account registration, is to obtain access to a large number of accounts on a target Web service. After looking at seven years' worth of job data on freelancer.com, Marti and his colleagues found that Gmail and Craigslist were the most targeted Web services for this job type. Marti commissioned the task of creating Web-based email accounts to 10 workers, the majority of whom delivered valid accounts. He observed that the accounts in many of the delivered sets were fairly old, indicating that they were stockpiled and not created on demand. The second job type he analyzed was online social network (OSN) linking. He defined OSN linking as buying friends, followers, or subscribers on sites such as Facebook, Twitter, and YouTube. They found that the commissioned workers were generally unable to deliver high-quality social links, since many of the delivered links came from fake OSN accounts. Finally, Marti covered search engine abuse, specifically jobs for creating written content that contains certain links or keywords. He observed that 10% of the jobs seen on freelancer.com fall into this category. The freelancer.com workers delivered mixed results for this task, with some doing very well and others ignoring job requirements.

Marti concluded that the large, cheap labor pool available to abusers changes the threat model to Web services and that traditional security mechanisms are not sufficient to stop abuse. However, he claimed it is possible for outsourcing sites to detect and remove abusive jobs. During the Q&A, Tyler Moore asked whether sites like freelancer.com are actually interested in filtering abusive jobs, given that they earn revenue from these jobs. Marti responded that they do very little enforcement, especially when compared to similar sites such as Amazon Mechanical Turk. He then added that freelancer.com is a legitimate business, so they might be willing to address the issue if the scope of the problem is brought to their attention. Finally, another attendee commented that even if freelancer.com is taken down, it is likely that another site will be created to provide the same abuse service. Marti agreed that this is a possibility.

Show Me the Money: Characterizing Spam-advertised Revenue

Chris Kanich, University of California, San Diego; Nicholas Weaver, International Computer Science Institute; Damon McCoy and Tristan Halvorson, University of California, San Diego; Christian Kreibich, International Computer Science Institute; Kirill Levchenko, University of California, San Diego; Vern Paxson, International Computer Science Institute and University of California, Berkeley; Geoffrey M. Voelker and Stefan Savage, University of California, San Diego

Chris Kanich started his presentation by pointing out two questions he is commonly asked about spam: who buys this stuff and how much money do the spammers make? In beginning to address these questions, Chris said that, at its core, spam is about advertising goods. The spammer—"affiliate marketer," in spam parlance—earns a commission on every sale they can provide to their affiliate program. Chris found that the order IDs for many affiliate programs appeared to be sequential and thus he was able to measure the IDs over time and estimate the sales throughput for those programs. Overall, the throughput varied from as low as 49 to nearly 900 orders per day. By combining the throughput with an estimated cost per order, Chris was able to calculate the average revenue per month for each program. This revenue varied from $200,000 per month to as high as $2,400,000 per month for the larger spam pharmacies.

Chris explained that they inferred information about the purchasers using the Web logs of a compromised image hosting server. The spam sites received views from all across the world, but sales were concentrated in the United States and Western Europe. Chris estimates 91% of all customers to be located in Western countries. While the vast majority of purchases are for recreational drugs such as Viagra, 29% are for non-recreational pharmaceuticals. US-based customers are four times more likely to buy non-recreational drugs than other Western customers.

Mark Seiden (Yahoo!) questioned the legitimacy of the drugs. Chris said that the drugs they tested contained the active ingredient in the correct amount, but they couldd not make any claims about other aspects of the drug. Another attendee asked about the percentage of purchases that arrived. Chris said that most arrived; the ones that did not were likely due to errors on his part. John Spring wanted to know to what extent this becomes a public health problem. Chris commented that their goal is to bring this issue to light and they are currently in contact with the FDA. Finally, an attendee brought up the issue of credit card fraud, pointing out that selling these drugs is already illegal, so why don't the programs go that extra step? Chris replied that these are businesses, and it is trivial for customers to contact the credit card company to cancel orders. In fact, the customer service for these programs tends to be very good.

Invited Talk

Privacy in the Age of Augmented Reality

Alessandro Acquisti, Associate Professor of Information Technology and Public Policy at Heinz College, Carnegie Mellon University

Summarized by Nathaniel Husted (nhusted@indiana.edu)

Alessandro Acquisti started his talk by discussing Cincinnatus, a Roman consul who, after being called back to service, defended Rome from northern invaders. A photograph of his statue was shown, in which the general is returning a symbol of military power with one hand and retrieving his agriculture tools with the other, dramatizing his choice to return to private life after his military victory. This story was used to indicate the importance of private life in ancient Rome. Alessandro then retold a story regarding a man who destroyed the legendary Temple of Artemis in Ephesus so that his name would be recorded for all history. The individual was captured and killed by the citizens of Ephesus. His captors also attempted to purge his name from history, but failed. We know the individual as Herostratus. Together, the two stories illustrate that throughout the course of human history we have been concerned with both our public and our private lives, as well as with controlling the information about us in the public sphere.

Alessandro's talk revolved around four major research experiments performed by Alessandro and his co-authors: the inconsistency of privacy evaluation, the paradoxical nature of privacy control, humans' ability to discount past information, and the use of social networks and face recognition for individual re-identification. These experiments try to look at how technology affects our privacy decisions and how privacy decisions affect our technology; how we make trade-offs and decisions regarding what information we want to keep private and make public; and what are the cost-benefit trade-offs in revealing private information.

The first experiment focused on whether people's evaluations of privacy can be manipulated. The experiment contrasted the willingness to accept cash to reveal personal data versus the willingness to pay cash to protect personal data. The results showed that participants' valuations of privacy changed significantly based on the priming and framing of the offer. If they started with less privacy, they valued privacy less; if they started with more privacy, they valued it more.

The second experiment focused on the paradoxical nature of control and its relation to privacy. Traditionally, control over personal information is believed to be a means of protecting privacy. Their experiment investigated whether more control can lead to less privacy. For this experiment, more than 100 students were asked to perform an online survey where a portion of the questions asked were sensitive in nature. One version of the survey stated that the answers to the survey, if provided by the subjects, would be published by researchers; the second version of the survey allowed individuals to choose what answers would be published. When allowed explicit control via the added box, individuals not only answered more questions but also allowed more answers to be made public. The results of the experiment show that making people feel more in control over their privacy can lead to more public disclosures of sensitive information.

The third experiment focused on how we judge individuals for past and present behavior, both good and bad. The experiment consisted of a survey in which individuals were asked to read a story about Mr. A, who either found a purse and kept a large sum of money or returned the purse with the money. This event either occurred five years ago or 12 months ago. Individuals formed very negative impressions of Mr. A when he was presented as having kept the money, no matter how long ago that event happened. However, when Mr. A was presented as having returned the money, individuals thought positively of him—but only if his good deed happened recently. If the good deed happened five years ago, there was no positive impact.

The fourth, and most recent, experiment concerned Alessandro's work on combining Facebook, personal information facial recognition software, and data mining. It is this portion of the talk where Alessandro's group is bridging the gap between science fiction and modern society. In this project they were able to compare images from Facebook's searchable profiles with head shots taken manually on the CMU campus or with images coming from dating Web sites, with the goal of identifying individuals online and offline. The culmination of this project was a sample augmented reality application that can perform the transfer from face to personal information on a mobile smartphone.

Alessandro discussed how these new technologies will affect our views on privacy. We can view our social network profiles as real IDs. In fact, social networks have, in some ways, turned into an inadvertent national ID. The convergence of various technologies also creates a "democratization of surveillance," because in a world where all personal information can be gathered from a face, we all become each other's big brothers with the aided use of a mere smartphone.

Some questions focused on whether younger people value privacy less; on whether the amount of value a person places on privacy changes if the loss is more concrete; on whether gender affects our decisions regarding discounting individuals' behavior; and on what the take-home message from the talk was. Alessandro tackled the first question by mentioning that what is most likely to happen is that views on what should be private and what public will change with time. Alessandro found that the concreteness or abstractness of the reward did not affect a person's behavior in the first experiment. He also found that, in the current studies, gender did not have a significant effect. Finally, the positive message from this talk was the hope provided by research advances in privacy enhancing technologies (PETS).

Defenses and New Directions

Summarized by Ben Ransford (ransford@cs.umass.edu)

Secure In-Band Wireless Pairing

Shyamnath Gollakota, Nabeel Ahmed, Nickolai Zeldovich, and Dina Katabi, Massachusetts Institute of Technology

Shyamnath Gollakota presented a protocol that allows a user to pair two wireless devices. When two devices are paired, they share a secret and can authenticate each other's transmissions. Some consumer-grade wireless devices, such as WiFi routers and Bluetooth audio equipment, establish a shared key via Diffie-Hellman (DH) key exchange when the user presses a button on both devices within a certain time window. However, on a wireless medium, simple DH is vulnerable to man-in-the-middle (MITM) attacks. Past academic work has proposed pairing protocols that require trustworthy out-of-band channels to bootstrap mutual trust, but Gollakota argued that out-of-band channels are difficult to incorporate in devices such as medical and home sensors, for reasons of both cost and size.

Gollakota described a new pairing protocol, Tamper-Evident Pairing (TEP), that is secure against MITM attacks and uses only in-band communication. The key idea is that, because an adversary cannot create radio silence by transmitting, pairing devices can reliably detect MITM tampering. TEP surrounds DH packets with a long leading synchronization packet and a specially constructed trailing hash. These modifications make TEP secure against adversarial message alteration, message hiding, and channel hogging. Because TEP messages are tamper-evident, if each pairing device receives exactly one untampered-with pairing request during the designated time window, they have successfully authenticated each other and can pair. The authors implemented TEP in the driver of a mainstream 802.11 card and evaluated it on a network test bed at MIT.

An audience member suggested that an attacker could selectively overpower the silent hash bits. TEP uses a balanced hash with an equal number of ones and zeroes to ensure that every one corresponds to a radio-silence zero. Zack Weinberg asked whether users would be willing to wait for the timeout period. A user with physical access can press a hardware button to preempt the timeout. Carson Gaspar (Goldman-Sachs) asked whether an attacker could overpower both the synchronization packet and the balanced hash. Such behavior would be detectable. Someone pointed out that TEP devices might interpret cross-technology interference from other products (e.g., microwave ovens) as TEP synchronization packets; Gollakota responded with some empirical data to demonstrate that only certain devices would interfere, and that those interfering devices would delay the pairing by only a few minutes. If a device persistently interferes, it breaks the paired devices' ability to communicate at all.

TRESOR Runs Encryption Securely Outside RAM

Tilo Müller and Felix C. Freiling, University of Erlangen; Andreas Dewald, University of Mannheim

Tilo Müller described a Linux kernel patch that allows AES operations to occur entirely outside of RAM. The patch addresses a well-known shortcoming of most cryptography implementations targeting microprocessors: secret keys are stored in RAM, where they are vulnerable to attacks that allow a miscreant to dump system memory—for example, the “cold boot” attacks presented at USENIX Security in 2009. Even popular full-disk encryption (FDE) implementations store secret keys in vulnerable memory. TRESOR implements AES in such a way that key material is stored in processor registers rather than RAM. It uses the large debug registers that are available only to processes running at the highest privilege level. Under normal operation, these registers are unused; they are available for breakpoints and rarely accessed configuration information. To circumvent RAM storage of keys, the authors implemented AES in x86 assembly, avoiding putting runtime variables in data segments and using 2-kilobit x86 SSE registers for intermediate states.

Müller compared TRESOR to an AES implementation using only generic x86 instructions, which was too slow, and to an implementation using Intel’s new AES-NI instruction set, which provides fast AES instructions but stores round keys in insecure RAM. TRESOR, in comparison, generates round keys on the fly and does not store them in RAM. Müller noted that the kernel’s normal context switching stomps on the debug registers, inspiring the authors to make TRESOR run in an atomic section. In the authors’ evaluation under QEMU, their search of emulated RAM with the open-source aeskeyfind tool failed to find the key under TRESOR but succeeded under the alternative schemes. In future work the authors plan to store keys in the Trusted Platform Module (TPM) or x86 machine-specific registers (MSRs). TRESOR is open source software available at http://www1.cs.fau.de/tresor.

An audience member noted that TRESOR keeps secrets in RAM briefly before they are moved to the debug registers, and asked whether the secrets had to be in RAM at all. Müller replied that passwords on Linux can be much larger than any available register. Frank Stajano asked about the performance impact of TRESOR and whether off-the-shelf FDE is usable under TRESOR; Müller said that TRESOR’s overhead compared to AES-NI was not huge. John Criswell noted that an attacker could change the kernel in memory or on disk to attack TRESOR, which Müller acknowledged. Another audience member asked whether the authors had studied other side channels; Müller reported that TRESOR should be resistant to timing attacks because the code does not use input-dependent branches; he pointed out that the authors had not yet considered power side channels.

Bubble Trouble: Off-Line De-Anonymization of Bubble Forms

Joseph A. Calandrino, William Clarkson, and Edward W. Felten, Princeton University

Bubble forms are machine-readable pieces of paper on which people place marks to indicate their preferences or opinions. Will Clarkson pointed out that bubble forms are used for voting in some precincts and then posted online, making them an attractive target for malefactors who wish to know how certain people voted. Clarkson’s group used a set of 92 anonymized surveys to train a classifier on several features of the markings such as shape, radius, center, and color distribution. They trained the classifier on 12 (out of 20) filled bubbles per person, then tested the classifier’s performance on the remaining eight. The results invalidated the common assumption that people cannot be identified by their markings on bubble forms: their classifier ranked the true respondent over all others more than half the time on 1200 dpi scans. Clarkson reported that the scans were robust against downsampling, with 45% of respondents correctly identified at only 150 dpi. Clarkson suggested several applications of bubble-form de-anonymization, such as the detection of cheaters on standardized tests. He concluded by suggesting several ways of making bubble forms more robust against their attacks; for example, making bubbles’ borders thicker decreased the classifier’s accuracy.

An audience member asked whether the authors’ chosen features were robust against changing the environment in which a person filled in the form; Clarkson explained their attempts to avoid overfitting by their classifier and remarked that people seem to be consistent as conditions vary. Another person asked about varying the scanner; Clarkson said that their blurring step normalized for scanner variations. Adrian Mettler suggested that users could use felt-tip pens to confound de-anonymization. An audience member asked whether stress affected bubble-form filling, and Clarkson acknowledged that it might. Another audience member asked whether the authors’ techniques could de-anonymize users from a much larger set; Clarkson agreed that further testing was necessary but said that voting, for example, often occurs in smaller precincts in which their classifier could work. Peter Neumann (SRI) suggested that a malicious party could de-anonymize voters using other techniques, such as marking ballots with invisible ink. Clarkson clarified the authors’ assumption that the parties that receive the bubble forms are not tampering with them.

Securing Search

Summarized by Ed Gould (summary@left.wing.org)

Measuring and Analyzing Search-Redirection Attacks in the Illicit Online Prescription Drug Trade

Nektarios Leontiadis, Carnegie Mellon University; Tyler Moore, Harvard University; Nicolas Christin, Carnegie Mellon University

Nektarios Leontiadis described their work measuring and analyzing specific attacks against search results, namely those used by illicit online pharmacies. This work seeks to determine the size, effectiveness, and weak points of the attacks. The specific choice of drug sales was motivated by the potential dangers of improper use of the drugs. As a form of illicit advertising, email spam is inefficient. Social network and blog spam are better, but Search Engine Optimization targets users more directly. A Google search for "no prescription cialis" will produce some odd results; some are legitimate, some are malicious.

To collect data, the team issued more than 200 queries daily during the experimental period. They note that SEO attacks are growing, while blog and email spam are declining. The number of pharmacies (both legitimate and illicit) is constant. Infections on the attacked systems tend to be long-lived, and seem to persist the longest on .edu sites. The researchers identified 34 connected components of the attack. They noted seven organized groups, loosely connected, that represent 50% of the infected nodes. Eleven ASes hosted most of the redirect servers.

They conclude that there is one major group of affiliates perpetrating these attacks, and that .edu sites are popular to attack.

Neil Schwarz asked why .edu domains take longer to disinfect. It is difficult to notice the infections. Lucas Ballard asked about query selection: when looking for good sites, how often do bad sites outperform good ones? They only have aggregated results, not differentiated by type of query. Lucas followed up by asking why the domain count is rising. Is the count limited to domains (domain rotation) or does it also include IPs? They just counted domains, not IPs. Stefan Savage pointed out a source of possible bias by using page access as an estimator. Some sites use on-site billing, others off-site. Off-site billing involves an extra redirect, and their data do not include payment sites.

deSEO: Combating Search-Result Poisoning

John P. John, University of Washington; Fang Yu and Yinglian Xie, MSR Silicon Valley; Arvind Krishnamurthy, University of Washington; Martín Abadi, MSR Silicon Valley

John described a tool, called deSEO, to combat Search Engine Optimization (SEO) attacks. The "malware pipeline" is, roughly, find vulnerable servers, compromise them to host malware, and spread links via search results that point to the malware. A Google search for "flintstones pictures myspace" yields a "scareware" link as the first result, claiming that the user's computer is infected by one or more viruses. About 40% of popular search terms are infected (changing, as what's popular changes). There is an estimated $150M profit in the scareware market.

John described how the mechanisms work, what the research can show, and the development of deSEO. Their analysis of attacks was carried out from August to October 2010. E-commerce sites often contain credit card information, which offers an additional incentive to compromise them. Files are uploaded to the compromised servers, and use PHP to generate malware pages. They were able to download a PHP script from a buggy server, even though this is not usually possible, and were thus able to analyze the script.

The general pattern is that there is a dense link structure on the pages generated, linking to more than 20 other sites with some 40 million pages generated, poisoning 20,000 popular search terms. Ninety-five percent of Google Trends terms are poisoned, targeting 100,000 victims over 10 weeks. It is difficult to detect and blacklist these pages, because they are typically cloaked to crawlers and search analysis, as the PHP scripts detect crawlers and produce benign results. Often it takes user interaction (e.g., mouse movement or clicks) to produce the malware. Features that can cause a site to get noticed are diverse behavior before compromise and similar behavior among pages after compromise.

The deSEO tool uses history-based filtering, cluster analysis of suspicious domains, and similarity analysis. They found about 1000 domains, with 15,000 URLs infected.

Alva Couch pointed out that John had just told us how to defeat his technique: use sparsity. Do they have any fallback mechanism? John replied that relevant keywords are still necessary, and clustering is needed to get the rankings. It is possible that the bad guys could use this information, but it seems unlikely.

Someone asked how the malware keeps lists of pages to link with up-to-date, and John answered that the malware server includes a list of sites to link to, and the PHP script selects from this list.

Securing Smart Phones

Summarized by Italo Dacosta (idacosta@gatech.edu)

A Study of Android Application Security

William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri, The Pennsylvania State University

One of the reasons for the increasing popularity of smartphones is the great number of mobile applications available. For example, Google Android, one of the most popular mobile OSes, has hundreds of thousands of mobile applications available in the Android market. However, these applications are not security certified, due to their large numbers and the lack of a common definition for security. As a result, malicious applications can be found in the Android market. This paper describes a breadth of security properties in a large set of popular Android applications to characterize their security and provides a better understanding of mobile applications' and developers' behaviors.

The sample set used in this study consisted of the top 10 most popular applications in each of the Android market's categories—a total of 1,100 applications. To analyze the security and behavior of the applications, access to their source code was required. Android applications are written in Java but use a different bytecode (.dex files) and runtime (Dalvik virtual machine); therefore, existing Java decompiler tools cannot be used directly. Hence, the authors built a Dalvik decompiler, ded, which takes the application's .dex files as input and returns the corresponding Java source code. The ded decompiler works as a multistage process (retargeting, optimization, and decompilation) and it is available on the project's Web site. Using ded to decompile the selected applications produced a total of 21 millions lines of code. The authors performed static and manual analysis of this code to look for dangerous behavior and vulnerabilities and to understand how applications handle sensitive information. The static analysis used Fortify SCA, a commercial tool for Java vulnerability analysis. The authors created custom rules to analyze the applications' source code, using different techniques such as control flow, data flow, structural analysis, and semantic analysis.

This study reports 27 findings which provide insight into the applications' and developers' behavior. In the area of phone identifiers, the authors found that 33 applications leak phone IDs. Regarding location data, 13 applications were found with location data flows to the network. The authors also found that 51% of the applications include an ad or analytics library. In many cases, applications have more than one third-party library. In addition, this study shows evidence of the use of developer kits, which hide the identity of the original developers and may include some dangerous functionalities. Also, several Android-specific vulnerabilities were detected. William Enck described some of the limitations of this study, such as the focus on popular applications, code recovery failures, limitations of the static analysis tool, and obfuscated code in some applications. Finally, Enck noted that this study offers the opportunity for a more automated security certification process for mobile applications.

Bill Soley (Oracle) asked if there are other implications besides privacy regarding phone identifiers such as IMEI numbers. While other malicious activity is possible, right now the main concern is privacy. David Evans (University of Virginia) said that developers are not being malicious but are just making mistakes and that a possible solution could be to generate unique IDs that do not leak privacy. Enck agreed and pointed out some recent research that follows this approach. Another participant asked if native code was found during the decompilation process. Yes, around 70 or fewer applications had native code; this is an area malware developers are beginning to push. Finally, someone asked how Enck would change the application sample set if he could. Enck would probably obtain a list of all available applications and select applications randomly. Also, the lists of recently added and paid applications can be interesting to study.

Permission Re-Delegation: Attacks and Defenses

Adrienne Porter Felt, University of California, Berkeley; Helen J. Wang and Alexander Moshchuk, Microsoft Research; Steve Hanna and Erika Chin, University of California, Berkeley

In modern client platforms such as browsers and mobile operating systems, applications are untrusted and isolated from each other by using IPC and specific communication mechanisms. They also require explicit permission to access resources such as camera, microphone, and user location data. These permissions are assigned per application to reflect user needs and level of trust in the application. However, a system that uses IPC and per-application permissions can be vulnerable to permission re-delegation attacks, where an application that lacks permissions gains access to additional privileges by communicating with another application (a special case of the confused deputy problem).

Adrienne Porter Felt described how they analyzed the permissions of 872 Android applications to find candidates that could facilitate this type of attacks. They found that 37% of the applications meet the required conditions for a candidate: a dangerous permission and a public interface. To discover the attacks, the authors built an automated tool that uses call graph analysis, and they manually verified the attacks found. The authors found 15 vulnerabilities in 5 system applications; however, other vulnerable applications may not have been detected.

Felt presented IPC Inspection, an OS or browser mechanism to prevent permission re-delegation attacks. When a deputy application (the privileged application) receives a message, the system reduces the deputy's permissions for the length of the session to the intersection of the deputy's previous permissions and the requester permissions. Also, to prevent DoS attacks, the deputy can specify who can and cannot send it messages. IPC inspection was implemented for Android OS and ServiceOS (Microsoft's research browser). The evaluation focused on determining whether IPC Inspection does not break applications and whether it effectively blocks permission re-delegation attacks. The evaluation results showed that 11 out of 20 randomly selected Android applications (from the set of 872) may require minor changes or additional permissions. In addition, the evaluation showed that IPC Inspection prevents all of the permission re-delegation attacks described in this study.

William Enck asked about the case where, in install-time systems, an intentional deputy attenuates authority, which can lead to permission bloat. You could add a time-of-use check, specifically for this case, that does not need to be necessarily a permission prompt. Felt also noted that people in her research group are working on "user driven access control with access control gadgets" to give the OS a way to know that an action is being approved by the user. Also, someone asked about when a singleton application is used. In this case, the deputy needs to declare itself as a singleton, because otherwise the application could crash. This problem does not happen on the Web, only in the Android OS. Adrian Mettler (UCB) asked about the possibility of escaping from the stack introspection protection option. This is possible, but developers could end up using this for all their messages. However, the option can be added, to make sure it does not break the application and to help application developers.

Quire: Lightweight Provenance for Smart Phone Operating Systems

Michael Dietz, Shashi Shekhar, Yuliy Pisetsky, Anhei Shu, and Dan S. Wallach, Rice University

Android protects applications from each other by using OS security mechanisms. In this model, applications should be slick (i.e., minimal permissions). Instead, however, most applications are complex, due to the use of third-party libraries such as mobile ads and mobile payments. Third-party libraries typically require additional permissions not originally required by the application and that can introduce bugs that affect the application's stability. Also, applications and third-party libraries mutually distrust each other. A simple solution to this problem is to split apart third-party libraries into separate applications. However, this approach introduces a new problem—it increases the risk of confused deputy attacks. In this type of attack, an application lacking a particular permission sends a request through another application that has this permission (confused deputy). The second application then forwards the request to the OS, allowing the first application to evade the permission mechanism.

Dietz described QUIRE, a mechanism that enables the separation of libraries from applications and protects data provenance and integrity, while preventing confused deputy problems. For this purpose, QUIRE introduces the idea of provenance-carrying IPC, where an application can protect itself by quoting the call chain that called it. Quoting only reduces the privileges of the application that chooses to quote the call chain; therefore, confused deputy attacks will not work even if a malicious application lies about the call chain. In addition, QUIRE provides verifiable communication between applications by using simple cryptographic mechanisms to protect data moving over IPC and RPC channels. Moreover, QUIRE does not require changes to the Dalvik virtual machine.

The QUIRE implementation consists of four components: the authority manager OS service, the network service provider OS service, the IPC stub/proxy code generators, and the trusted UI. To evaluate QUIRE, the authors built two demo applications: a secure mobile payment system and a mobile ad service. Through these applications, the authors demonstrated the security benefits and practicality of QUIRE. In addition, the performance evaluation showed that QUIRE overhead is small (80 microseconds per IPC).

Arjun Guha (Brown University) asked what applications' installation looks like under the QUIRE model. Dietz responded that applications will need to use dependencies at installation time to learn what other applications need to be installed. William Enck asked if provenance happens on intents. Dietz explained that QUIRE hooks to the service binding IPC at this point. He has not looked at intents yet but anything using Binder should work well. Paul Pearce asked about ad networks functionality not supported by QUIRE prototype implementation. Dietz responded that he could not think of one at the moment. QUIRE was designed as a system that application developers can use to build a policy on top and, in some cases, it may break functionality. Dave Evans asked what happens when applications use the network instead of IPC for communications. Dietz responded that this will prevent provenance, but it has not been an issue yet. It will be something to consider when the boundary between mobile and Web applications blurs.

Invited Talk

Deport on Arrival: Adventures in Technology, Politics, and Power

J. Alex Halderman, Assistant Professor, Computer Science and Engineering, The University of Michigan

Summarized by Adam Bates (amb@cs.uoregon.edu)

J. Alex Halderman presented stories from three strands of his research—early digital-rights management attempts in audio CDs, security analysis of voting machines in the United States, and security analysis of voting machines in India. Through these stories, Halderman explained the risks for researchers whose work leads them to a stand-off with politically or economically powerful parties. He also demonstrated the importance of being able to explain highly technical security issues in a manner that is palatable to the public.

Halderman's work in digital rights management began at Princeton University as a graduate student in 2003. At the time, companies like Sony were trying to secure their intellectual property that was being distributed in a legacy format, the compact disc. In an early generation of this technology, Halderman discovered that Sony was leveraging the Windows Autorun feature to install software that interfered with the CD Driver. Using the Freedom to Tinker blog as a mouthpiece, he posted that the DRM software could be avoided by holding down the Shift key as the CD was inserted. The “DRM is defeated by the Shift key” story caused the responsible company’s stock to drop by 80%. Halderman also spoke out against Sony’s infamous DRM-as-rootkit attempts, going back and forth with the company in a “delightfully public” manner. Communicating these issues via a blog helped the Center for Information Technology Policy (CITP) to speak directly to the public. The negative publicity eventually forced Sony to abandon the initiative.

Halderman next related his history with the Diebold voting machines. The move to electronic voting systems was motivated by the voting fraud vulnerabilities of bulky, lever-based machines. Unfortunately, the early generation machines were rushed to market without much regard for computer security. Companies like Diebold had not voluntarily subjected their machines to any kind of independent analysis. In 2006, the CITP lab at Princeton was able to acquire a machine and reverse engineer the hardware for thorough analysis. They discovered and published a number of easily deliverable vulnerabilities, including the ability to infect a machine with malicious software without leaving a trace. It was also possible to create a virus that could spread from machine to machine. This led to another public standoff, where Diebold touted the importance of overlooked security features such as the need for a key in order to gain physical access to machine hardware. However, only one key was used universally and it was easily obtainable commercially.

Halderman went on to work on California’s “top-to-bottom” voting machine analysis. California was one of the first states to recognize the threat that insecure electronic voting posed. Under threat of decertification, voting machine manufacturers were required to share their code with the study. However, the fact that the study was being called for by politicians exposed a potential conflict of interest. For this reason, it was important to this research team that they had permission to share their results with the public.

The efforts of Halderman and his students at the University of Michigan also helped to draw attention to very serious issues in Washington DC’s prototype Internet voting system. The system was about to go live for an actual election when they were opened for security probing. Halderman’s group launched attacks that altered ballots and broke the confidentiality of legitimately cast votes. In spite of the fact that they added the “Hail to the Victors” audio track to the vote confirmation page, their penetration went unnoticed for several days. This work eventually helped to derail Washington DC’s use of the online system.

More recently, Halderman became involved in an analysis of India’s voting machines. In spite of solid design and effective deployment, fraud was rumored to have occurred in Indian elections. The study found two serious vulnerabilities: (1) installing a dishonest display board by replacing the LED component; (2) designing a device that modified the votes while in storage on the EEPROM. As a result of these findings, Halderman and his colleagues fell out of favor with the Indian Election Commission and with local law enforcement. After Hari Prasad, one of the Indian collaborators, promoted these findings on television, he was detained by the police. The Commission finally accepted the need for change, but on a subsequent trip to India Halderman was barred from entering the country for 24 hours. Stalling for as long as possible to avoid Halderman’s deportation, the Election Commission was able to speak on his behalf and get him into the country as their guest. The Commission, now prototyping a paper trail add-on, is seen as a model for developing democracies.

Lessons learned included the power of being technically correct, the importance of effective communication with the public, and the dire threat to democracy posed by insecure electronic voting systems. Halderman concluded by charging the audience to continue to change the world through computer security.

Poster Session

First set of posters summarized by Michael Z. Lee (mzlee@cs.utexas.edu)

IMD Shield: Securing Implantable Medical Devices

Shyamnath Gollakota and Haitham Al Hassanieh, Massachusetts Institute of Technology; Benjamin Ransford, University of Massachusetts Amherst; Dina Katabi, Massachusetts Institute of Technology; Kevin Fu, University of Massachusetts Amherst

Benjamin Ransford (ransford@cs.umass.edu) presented this work. The goal is to counter a set of attacks on implantable medical devices (IMDs) published in 2008. The primary issue is that some devices are susceptible to passive and active attacks. However, invasive surgery to retroactively fix these issues is expensive and carries risk, so the authors sought another solution. Their proposal is a wearable device, called the IMD Shield, that uses friendly jamming to block messages to and from an IMD. This device blocks incoming active attacks as well as outgoing messages. The radio configuration employs two antennas, which allows them to simultaneously receive the sensitive signal from the IMD and jam the signal so that eavesdroppers cannot decode it. The IMD Shield’s random jamming signal works like a one-time pad; it is the only device that is able to decode the new signal.

Using GPUs for OS Kernel Security

Weibin Sun and Robert Ricci, University of Utah

Security can be computationally expensive, but some operations can be parallelized and would benefit greatly from using the computational power of a GPU. Weibin Sun (wbsun@cs.utah.edu) presented KGPU, a kernel driver that leverages the GPU to offload expensive but easily parallelized operations such as encryption and AV signature matching. Because the current interface to GPUs is through a proprietary driver, they use a helper program to translate between KGPU requests and CUDA calls. Although this requires extra memory copying from kernel to user space, it seems to provide a nice speedup.

The Art of War Applied to Intrusion Detection in Wireless Ad-Hoc Networks

Stefan Stafrace and Bogdan Vrusias, University of Surrey

When working with intrusion detection systems in wireless ad hoc networks, the efficient use of resources is key, because nodes in ad hoc networks are resource-constrained. In a traditional network, you're able to deploy intrusion detection systems in strategic choke-points, but in wireless ad hoc networks this is not possible, due to the use of the shared medium and node churn. Stefan Stafrace (s.stafrace@surrey.ac.uk) suggests applying risk-based military principles to efficiently detect intrusions in wireless ad hoc networks. The authors offered a case study in which systematic route patrols were conducted by squads of agents to detect a sinkhole attack. The results show that high detection precision can be obtained while also conserving resources and limiting the data packet loss due to the attack.

A Digital Forensics System Using a Virtual Machine Monitor Integrated with an ID Management Mechanism

Manabu Hirano and Hiromu Ogawa, Toyota National College of Technology, Japan; Takeshi Okuda, Nara Institute of Science and Technology (NAIST), Japan; Eiji Kawai, National Institute of Information and Communications Technology (NICT), Japan; Youki Kadobayashi and Suguru Yamaguchi, Nara Institute of Science and Technology (NAIST), Japan

Manabu Hirano (hirano@toyota-ct.ac.jp) presented this poster. When performing digital forensics, one can run into the problem of unattributed data tampering, which can lead to false accusations and other bad outcomes. The authors propose a system called BitVisor, a hypervisor-based solution that employs user ID management to securely record who is accessing and modifying data. The end result is that the VMM is able to securely store the ID outside the reach of the guest operating system, translating actions in the guest OS with a helper program installed by the hypervisor. They use TPM and SecVisor-like (Cylab/CMU) properties to guarantee that an attacker cannot statically replace the VMM or tamper with its data dynamically during runtime.

Automated Model-based Security Management of Web Services

Rajat Mehrotra and Qian Chen, Mississippi State University; Abhishek Dubey, Institute for Software Integrated Systems, Vanderbilt University; Sherif Abdelwahed, Mississippi State University; Krisa Rowland, US Army Engineer Research and Development Center

Rajat Mehrotra (rm651@msstate.edu) presented an autonomic performance and security management framework for Web services. The goal is to integrate system control, optimization, and security analysis into a common model-based framework. It enables distributed Web services to efficiently adapt to varying load requirements and identify and mitigate potential security incidents. In modeling the behavior of a system, the authors wish to efficiently estimate system behavior and make adjustments as necessary. Using various inputs from security, performance, network, and system measurements, they can differentiate between different safe and unsafe system scenarios.

NotiSense: An Urban Sensing Notification System to Improve Bystander Privacy

Rob Smits, Sarah Pidcock, Ian Goldberg, and Urs Hengartner, University of Waterloo

Although crowd-sourcing data collection using mobile devices is de-anonymizing for the participant, bystanders should be notified so that they can preserve their privacy while protecting the identity of the data collector. Sarah Pidcock (snpidcoc@cs.uwaterloo.ca) presented NotiSense, a service to help notify such bystanders. The authors accomplish this by collecting enough information about the data collector, hashing and filtering the locations, and then having the collector's mobile device rebroadcast information. Bystanders in the area can see these broadcasts, check whether they're affected, and notify users. In a field test, they find that it is effective enough to cover a reasonable area around a data collector.

Secure Computation with Neural Networks

Brittany Harris and Jiamin Chen, University of Virginia

Brittany Harris (bjh3ev@virginia.edu) and Jiamin Chen (cjmyezi@gmail.com) presented this work. Oblivious computation can be used to jointly compute values while preserving each party's privacy. This work applies Yao's Garbled Circuit to enable the joint computation of the weights of a neural net.

This allows two parties to jointly train a neural net using data from both parties, without exposing their private training data or intermediate weight results to the other party. Alice first computes the weights using her training data directly, and then Alice and Bob execute a garbled circuit protocol where the inputs are Alice's learned weights and Bob's training data, to obtain the final weights without revealing either the intermediate results or training data.

SPATor: Improving Tor Bridges with Single Packet Authorization

Rob Smits, Divam Jain, Sarah Pidcock, Ian Goldberg, and Urs Hengartner, University of Waterloo

Tor is used for anonymity but is susceptible to some kinds of attacks. Rob Smits (rdfsmits@cs.uwaterloo.ca) and his colleagues are addressing an attack on Tor clients who have opted to become Tor bridges. The adversary assumes that the correct IP address for his victim is contained in one of the bridge descriptors. He can perform aliveness checks on the Tor bridges he has collected and then take an intersection of bridge IP addresses that were detected as online to de-anonymize this Tor client. The authors propose that, as clients receive bridge descriptors, an additional time-limited key be included. From this, clients derive a ConnectionTag—a 32-bit field, encoded in the initial sequence number and TCP timestamp of the initial SYN packet. If it does not validate, the Tor Bridge can drop the request before revealing aliveness.

Vulnerabilities in Google Chrome Extensions

Nicholas Carlini, Adrienne Felt, Prateek Saxena, and David Wagner, University of California, Berkeley

Adrienne Felt (apf@cs.berkeley.edu) presented this poster. Chrome allows users to install extensions that run with elevated browser privileges. Bugs in extensions can leak privileges to malicious Web sites or active network attackers. To help mitigate this, Chrome's extension platform includes several security features. However, in analyzing the top 50 Chrome App Store extensions and 50 randomly selected extensions, the authors found that 42 have vulnerabilities. For example, the Google Voice Chrome extension automatically searches for strings that look like phone numbers and converts them into links that, upon click, will make a call. Thus, a malicious site can use JavaScript to click pay-per-call numbers. Isolated worlds successfully reduce the number of vulnerabilities that a malicious Web attacker can leverage, but there are numerous bugs that active HTTP modification can attack. In general, privilege separation is not effective, because developers circumvent privilege separation, either intentionally or accidentally.

Unifying Data Policies across the Server and Client

Jonathan Burket, Jenny Cha, Austin DeVinney, Casey Mihaloew, Yuchen Zhou, and David Evans, University of Virginia

Web applications currently take a decentralized and ad hoc approach to security. Austin DeVinney (adevinney@radford.edu) and Yuchen Zhou (yz8ra@virginia.edu) presented a unifying framework applied to specific security policies once and then automatically enforced throughout the application. On the server side, they provide GuardRails, an additional layer on top of Ruby on Rails which allows an author to specify certain server security properties that are automatically enforced throughout the application. In addition, they modified the Chromium browser to interpret the generated attributes and enforce policies that protect private content from untrusted scripts running in the browser.

Improved XSS Protection for Web Browsers

Riccardo Pelizzi and R. Sekar, Stony Brook University

Riccardo Pelizzi (rpelizzi@cs.stonybrook.edu) presented this poster. Chrome and IE have implemented detection for reflected cross-site scripting (XSS) attacks from GET and POST parameters. However, the two browsers do not detect partial XSS injection attacks that take advantage of existing scripts already in the Web page. The authors found that 8% of the Web sites surveyed are vulnerable to this type of attack. Their approach to this problem is to improve filtering by parsing the input from GET and POST requests into parameters, and to use approximate substring matching to cover a wider range of Web application sanitization logic. Their implementation and evaluation shows that they cover more cases than Chrome's own filtering with an acceptable overhead as compared to Chrome.

Challenges in Deployment and Ongoing Management of Identity Management Systems

Pooya Jaferian, University of British Columbia; Kirstie Hawkey, Dalhousie University; Konstantin Beznosov, University of British Columbia

Pooya Jaferian (pooya@ece.ubc.ca) presented this poster, which tries to answer the question, how do people (corporations) do ID management? Their preliminary results from collecting and analyzing support logs show that, overwhelmingly, issues arise during installation. The process of troubleshooting is a close collaboration among consultants, support staff, and users employing a variety of content and debug methods such as interactive debugging, screen shots, and calls through many rounds of communication over a variety of channels.

An Arithmetic Operation Implementation Strategy for Privacy-Aware Role-Based Access Control

Yoonjeong Kim, Hyun-Hea Na, and Ji-Youn Lee, Seoul Women's University; Eunjee Song, Baylor University

Role -based access control (RBAC) is a model that effectively limits security vulnerability by controlling access to a specific role. However, the model is incomplete—the intent of a user is equally important when trying to enforce least privilege. Yoonjeong Kim (yjkim@swu.ac.kr) presented this work whose goal is to add arithmetic operations to allow easier specification of purpose, obligation, and conditions of access. To this end, the authors use XPath and XML specification to port Java applications to allow for arithmetic operations.

Second set of posters summarized by Christian Rossow (christian.rossow@gmail.com)

AdSentry: Comprehensive and Flexible Confinement of JavaScript-based Advertisements

Xinshu Dong, National University of Singapore; Minh Tran, North Carolina State University; Zhenkai Liang, National University of Singapore; Xuxian Jiang, North Carolina State University

Xinshu Dong presented AdSentry, a framework to reliably execute JavaScript Web advertisements. The system is based on a shadow JavaScript engine that is used as a sandbox to run untrusted ads in parallel to the normal JavaScript execution. The sandbox monitors accesses, and access control policies help to mitigate the insecurity of malicious JavaScript code. AdSentry was implemented as a prototype for Mozilla Firefox.

The Socialbot Network: When Bots Socialize for Fame and Money

Yazan Boshmaf, Ildar Muslukhov, Konstantin Beznosov, and Matei Ripeanu, University of British Columbia

Many users of social networks make their personal data private and only accessible to their friends. Yazan Boshmaf (boshmaf@ece.ubc.ca) presented work in which the authors created more than 100 artificial Facebook accounts and analyzed how users reacted to friendship requests from these accounts. More than a third of the friend requests were accepted. As a consequence, attackers were able to gain significantly more personal data about other users than is accessible via public profiles.

DETER Testbed: New Capabilities for Cyber Security Researchers

Terry Benzel, John Wroclawski, Bob Braden, Jennifer Chen, Young Cho, Ted Faber, Greg Finn, John Hickey, Jelena Mirkovic, Cliff Neuman, Mike Ryan, Arun Viswanathan, Alefiya Hussain, and Stephen Schwab, USC Information Sciences Institute (ISI); Brett Wilson, Cobham, Inc.; Anthony Joseph and Keith Sklower, University of California, Berkeley

DETERlab, the DETER Testbed, is an Emulab-based cluster testbed for cyber-security research which has been operated for many years by the DETER group. This poster showed new capabilities of the testbed, particularly that DETER is able to scale up to emulating an entire Internet infrastructure, including an autonomous-systems network.

An Analysis of Chinese Search Engine Filtering

Tao Zhu, Independent Researcher; Christopher Bronk and Dan S. Wallach, Rice University

Tao Zhu (zhutao777@gmail.com) presented work that analyzes the extent to which Chinese search engines censor search results for specific keyword groups. The authors found that pornographic terms and names of politically important persons are commonly filtered. A long-term analysis shows temporal changes in the censorship, presumably caused by extending filter blacklists. The authors also observed that some search engines maintain whitelists of presumably safe Web sites for specific search keywords.

More Efficient Secure Computation on Smartphones

Sang Koo, Yan Huang, Peter Chapman, and David Evans, University of Virginia

Yan Huang (yhuang@virginia.edu) presented this work on efficient and privacy-conforming data calculations on smartphones. The authors explored a protocol to find common contacts between two mobile phone users without sharing any contacts. Efficiency is gained by using a garbled circuit framework. The authors also show other privacy-preserving use cases for their framework, e.g., to determine geographical proximity between two mobile devices.

Understanding Attacks

Summarized by Robert Walls (rjwalls@cs.umass.edu)

SMS of Death: From Analyzing to Attacking Mobile Phones on a Large Scale

Collin Mulliner, Nico Golde, and Jean-Pierre Seifert, Technische Universität Berlin and Deutsche Telekom Laboratories

Collin Mulliner began his talk by pointing out that previous work on mobile phone security has largely neglected the more common feature phone in favor of smartphones. In fact, feature phones still dominate the market, with some estimating that only 16% of mobiles are smartphones. For this work, Collin set out to test the security of feature phones by looking at the SMS implementations across a variety of different phones. Since it is infeasible to test all models, he focused on phones from the most popular manufacturers. Collin claimed that due to the reuse of phone platforms, a bug found on one phone model is likely to translate to all other models that share the same platform. Unfortunately, it is very difficult to actually analyze feature phones, because the many platforms are all closed source. Collin's solution was to look outside of the phone and perform his analysis using his own custom GSM network and fuzz-based testing.

Collin then covered the results of his SMS fuzz testing for a select set of phones. For most phones, the found bugs crashed the phone, causing it to disconnect from the network and reboot. Many of the bugs can be triggered without direct interaction by the user of the target phone: merely receiving the message will cause the crash. Interestingly, some of the bugs caused the phone to crash before it could send an acknowledgment to the provider. Collin suggested that this behavior could be used to amplify the attack's effect, because the provider will repeatedly retransmit the attack message. Collin went on to discuss a number of possible large-scale attacks, including targeting all of the customers of a specific provider or manufacturer. He noted that existing bulk SMS operators can provide the necessary SMS throughputs to make such attacks possible. Finally, Collin discussed a few possible countermeasures, including patching the firmware and filtering SMS messages. However, he pointed out that both techniques are poorly suited to addressing this problem.

Rik Farrow suggested that it might be possible for a specially crafted SMS attack to modify and effectively gain control of the phone. Collin remarked that they saw at least one bug that could possibly be used this way, but such attacks are infeasible; it is a tremendous amount of effort to exploit a single phone, and even then only that particular model would be affected. Dan Farmer wondered if there was a way to fingerprint phones to identify the specific model. Collin replied that there are some methods that rely on MMS implementations, but they found it was only possible with a small number of providers. Collin concluded the Q&A by showing a video demonstration of his attacks crashing a number of mobile phones.

Q: Exploit Hardening Made Easy

Edward J. Schwartz, Thanassis Avgerinos, and David Brumley, Carnegie Mellon University

Modern OS defenses are designed to make exploiting binaries more difficult. Edward Schwartz questioned the true effectiveness of these defenses. In his talk he focused on hardening exploits against two common defenses: data execution prevention (DEP) and address space layout randomization (ASLR). DEP prevents memory from being both writable and executable at the same time, thereby preventing an exploit's shellcode from executing. DEP can be bypassed by using return oriented programming (ROP), which utilizes instructions, or gadgets, that are already present in the target binary. ASLR seemingly makes ROP difficult to use by randomizing the location of those instructions; however, modern ASLR implementations actually leave small amounts of code unrandomized in memory. Edward's solution to evade OS defenses is called Q. Q searches the unrandomized program image to automatically build the gadgets needed for ROP, arrange gadget types such that they implement the desired computation, and assign compatible gadgets to the arrangement.

Edward then discussed how Q can automatically modify existing exploits to bypass DEP and ASLR. Q uses trace-based analysis to identify the execution path of the exploit. Using the resulting path constraints along with a set of exploit constraints, Q can automatically create a modified exploit that is unaffected by DEP and ASLR. Edward demonstrated this in a video showing Q hardening an exploit which was then successfully used on a machine with DEP and ASLR enabled. Edward went on to explain how Q was able to successfully harden a number of real exploits for both Windows and Linux. Further, he claimed that Q is able to create ROP payloads for most programs that are larger than 100 KB. He also discussed a number of limitations of Q. First, it currently only uses single path analysis, and this prevents Q from finding certain exploits. Second, Q's gadgets are not Turing-complete. Third, Q does not support conditional gadgets. Edward concluded by saying that even small amounts of unrandomized code makes DEP and ASLR completely ineffective.

John Grizzle (Illinois) asked if control flow integrity would prevent these types of attacks. Edward replied that it would. They chose to investigate DEP and ASLR because they know they are not perfect and they wanted to gauge how good they actually are. Dave Melski (GrammaTech) pointed out that a lot of vulnerabilities will place constraints on the type of inputs, e.g., no null bytes. He wondered how Q handled this. Edward responded that this is partially addressed by the path constraints; if a payload violated path constraints, Q would not find an exploit. For generating payloads, the user can specify the type of bytes that are allowed. Joe Werther (MIT) asked about the prevalence of non-ASLR images in modern operating systems. Edward responded that they did not have widespread statistics, but there is a report referenced in their paper which claims that many popular software packages have at least one module that is not marked as randomized. Finally, Karl Koscher (U. Washington) suggested that you could come up with a subset of gadgets to locate libc and subsequently use all of the gadgets provided by libc. Edward agreed that this would be interesting and said there is another paper, “Surgically Returning to Randomized libc,” which discusses locating libc, but for a different application.

Cloaking Malware with the Trusted Platform Module

Alan M. Dunn, Owen S. Hofmann, Brent Waters, and Emmett Witchel, The University of Texas at Austin

Trusted computing aims to provide a secure environment for computation. It attempts to accomplish this by creating a hardware root of trust, most commonly using a trusted platform module (TPM). Interestingly, Alan Dunn argues that the same security properties provided by a TPM can be used to provide a hardware cloak for malware. Alan said that malware can, for example, use TPMs to store secret keys, prevent monitoring by security analysts, and ensure that only unmodified malware is executed. More concretely, the TPM can be used with special processor instructions to provide secure execution via a non-analyzable late launch environment that is separate from system software on the platform. To do this, the malware writers must first make sure that sensitive computations are separated and encrypted such that they can only be decrypted by the TPM within the late launch environment. This is accomplished through use of TPM binding keys and remote attestation. When a remote malware distribution platform is satisfied that the conditions are met, it returns the encrypted payload for execution on the compromised host. Alan and his colleagues implemented three examples of malware using the TPM.

Alan described some possible defenses against TPM malware. The first defense is whitelisting late launch binaries. This defense is largely satisfying; however, it requires a hypervisor, which may be troublesome for home users to install. Additionally, it might be difficult to maintain the whitelist. The second defense is manufacturer cooperation, in which the manufacturer breaks TPM security guarantees to allow a security analyst to impersonate a legitimate TPM. The last defense is based on physical compromise of a TPM. However, the industry has incentives to fix existing physical attacks in order to maintain meaningful TPM security guarantees. Alan argues that strengthening TPMs against physical attacks actually makes TPM malware more resilient.

Bryan Parno (Microsoft Research) questioned whether an analyst would really have a problem analyzing this type of malware, given that they have sufficient resources to physically compromise the TPM. Alan questioned Bryan’s assertion that the TPM’s physical protections are only there to protect against low-capability attackers like common laptop thieves. Alan then suggested that there might be a range of adversaries between laptop thieves and the NSA. Another attendee expanded on Bryan’s question by asking if the industry actually has any incentives to fix TPM’s physical compromises. Alan conceded that this claim was closer to an opinion than a fact.

Invited Talk

The (Decentralized) SSL Observatory

Peter Eckersley, Senior Staff Technologist for the Electronic Frontier Foundation, and Jesse Burns, Founding Partner, iSEC Partners

Summarized by Italo Dacosta (idacosta@gatech.edu)

SSL/TLS is the most popular cryptographic system. It allows establishment of a secure communication channel between a client and a server by relying on X.509 certificates signed by a certificate authority (CA). SSL/TLS robustness is as good as its ability to authenticate the other party. However, as has been shown recently, there are several problems with the CA trust model. Certifying identities on the Internet is a hard job with odd incentives. CAs often make mistakes resulting in vulnerabilities, there is (circumstantial) evidence of governments compelling CAs to sign rogue certificates, and there are a great number of CAs, all equally trusted. In addition, the X.509 standard has a history of implementation vulnerabilities, and its extreme flexibility and generality have created a large number of disparate certificates.

The goal of the SSL Observatory is to investigate the problems associated with CAs, the types of certificates they are signing, and the size of the PKIX (public X.509) attack surface. In 2010, the SSL Observatory collected all available X.509 certificates on the Internet by scanning the IPv4 address space (3 billion IANA-allocated addresses) for port 443/TCP. They found 16.2 million IP addresses listening on port 443, 11.3 million SSL handshakes, and 4.3+ million valid certificate chains, with only 1.5+ million distinct certificates (leaves). The results are publicly available for anyone interested in analyzing them. The approach used with IPv4 will not work with IPv6, due to its larger address space, so new approaches will be required once IPv6 is fully deployed. The new version of this project is the Decentralized SSL Observatory, a browser extension that will allow the SSL Observatory to collect certificates from different network viewpoints. This approach is important because most attacks against SSL/TLS are only visible in the network path between the victim’s client and the server (i.e., localized attacks).

Eckersley and Burns then explained their findings. First, the results confirmed that there are a lot of CAs on the Internet: 1,482 CAs trustable by Microsoft or Mozilla from 651 organizations. Second, CAs are located in approximately 52 countries, which means exposure to many jurisdictions. Third, several vulnerabilities were detected: around 30,000 servers were using broken keys or valid certificates with generic names (e.g., localhost). Fourth, several problems associated with certificate revocation were found—for example, a large number of revoked certificates (~1.96 million revocations), the lack of revocation support (683 certificates without revocation information), and lack of a clear reason for revocation. Fifth, they found several configuration errors: violations of Extended Validation (EV) rules, CA certificates with keys from expired certificates, 512 and 1024 EV certificates, and certificates with huge list of names. They concluded that the attack surface includes not only CAs and target server but also the DNS infrastructure and anywhere in the network path between the client and the server.

Next, they discussed some proposed solutions to the SSL/TLS problems. (1) the consensus measurement approach (e.g., Perspectives and Convergence.io) attempts to get certificate information from different network vantage points to detect any anomaly; a disadvantage of this approach is the possibility of false positives. (2) More vigilant auditing, such as the SSL Observatory project, could be done. (3) The DNSSEC+DANE solution uses the existing relationship with the domain registrar to get the certificates for a Web site without requiring a CA, but this solution requires DNSSEC to be fully deployed. Also, DNSSEC+DANE defends against attacks to CAs but does not protect the rest of the attack surface. (4) Certificate pinning could be done via HTTPS headers: "whoever used to be domain.com should stay domain.com." This idea is simpler than DNSSEC and provides better security if implemented correctly (it protects the whole attack surface except for the first request). Eckersley described use of a private CA per domain in parallel to PKIX to cross-sign pinned certificates. However, X.509 certificates do not support cross-signatures. He suggested possibly using a second leaf certificate signed by the pinned "private CA" key or using an X.509 extension with a cross-signature.

A member of the audience commented on client certificates as another possible solution to the attacks against SSL/TLS and noted that federated login could help to deploy client certificates. The presenters responded that while the use of client certificates prevents the theft of authentication credentials, many other attacks are possible and additional solutions are still required. To the question of how pinned certificates are revoked, Eckersley commented that he has several ideas in mind, such as adding a timestamp, but more discussion is needed. Someone suggested having a hierarchical structure like DNS, where CAs are limited to sign certificates for particular domains. The speakers agreed and noted that an X.509 extension, name constraints, allows such functionality. The problem is that this extension is not widely supported, and it introduces some operational problems. Another person added that limiting the scope of CAs is also against their financial model and therefore will be difficult to implement. Adam Langley (Google) talked about the many problems with current certificate revocation mechanisms (e.g., performance and privacy issues) and asserted that new approaches are required. Stephen Kent (PKIX WG chair) commented that PKIX is not in itself bad—it is the way it is implemented in browsers. Eckersley noted the semantic problem caused by the increasing number of top-level domains. Finally, Burns mentioned that more transparency is needed regarding the sub-CA information of each root CA.

Dealing with Malware and Bots

Summarized by Lakshmanan Nataraj (lakshmanan_nataraj@umail.ucsb.edu)

Detecting Malware Domains at the Upper DNS Hierarchy

Manos Antonakakis, Damballa Inc. and Georgia Institute of Technology; Roberto Perdisci, University of Georgia; Wenke Lee, Georgia Institute of Technology; Nikolaos Vasiloglou II, Damballa Inc.; David Dagon, Georgia Institute of Technology

Manos Antonakakis said that Internet Protocol (IP) address-based blocking techniques can no longer keep up with the number of IP addresses the command and control (C&C) servers use. Also, there is a time delay between the day a malware is actually released in the wild and the day security researchers analyze that malware. Furthermore, the daily DNS lookup signal for malware-related domain names is different from that of normal Web sites. Hence, the authors propose a system, called Kopis, that statistically models the DNS lookup signal by utilizing the data in the upper part of the DNS hierarchy and builds an early warning system to detect malicious domain names. It leverages the fact that since DNS is a distributed hierarchical database, there must be a place in the DNS hierarchy that enables one to have global visibility from the point of view of who is looking up the domain names. Based on this observation, the system detects malicious domain names.

An interesting and important point to be noted here is that the system does not need a malware binary to detect malware domain names. The system can analyze large volumes of DNS messages at AuthNS or TLD servers. It also introduces an alternative IP reputation classification signal for DNS due to which botnets can be identified several weeks before the malware is actually found.

The basic building block of the system is an authoritative domain name tuple with two components: the resource record, which is a mapping from the domain name to its IP address, and the requester. Features such as requester diversity, requester profile, and resolved-IPs reputation are used. The requester diversity feature identifies whether the machines that query a given domain name are localized or globally distributed. Using this feature, the authors show that malicious domain names are more widespread than benign domain names. The requester profile feature allows one to see whether a certain IP has historically had lookups for some specific malicious domain names. This enables one to know if a network has been well protected or not. Using these features, the authors perform a long-term evaluation where they show their system can reliably detect malicious domain names with a low false-positive rate. Manos concluded the talk with some case studies on some of the botnets their system had discovered. Kopis can be incorporated as an early warning system that can detect malicious domain names well before the malware reaches a network.

Marc Eisenbarth (HP) asked if their system would work from lower tiers in the DNS hierarchy. Manos answered that their system works as long as you have enough visibility.

BotMagnifier: Locating Spambots on the Internet

Gianluca Stringhini, University of California, Santa Barbara; Thorsten Holz, Ruhr-University Bochum; Brett Stone-Gross, Christopher Kruegel, and Giovanni Vigna, University of California, Santa Barbara

Gianluca began by noting that 85% of worldwide spam is through botnets and it is important to locate those spambots responsible for sending most of the spam. A simple approach to track spambots would be set up spam traps with fake email IDs and use the spam received in these fake IDs to track the bots. However, spam traps suffer from some limitations, since only a subset of spambots which are trapped using the spam trap can be detected. Also, the implementation of spam traps may not be easy, since some spambots operate only in certain countries and send spam only within those countries. Based on the premise that bots within a botnet share similarities, the authors propose a system called BotMagnifier, which observes a portion of a botnet and identifies more bots belonging to it.

The system builds on two inputs. The first is a set of IP addresses of known spam bots called seed pools. These are the ones that participate in a specific spam campaign (emails with similar subject) and are obtained by setting spam traps. The second is a log of both benign and malicious email transactions called a transactions log. This log was obtained from a Spamhaus mirror. The system is operated periodically where, at every instant, a set of seed pools (minimum of 1000 IPs) are supplied as input, and at the end of each observation period (typically a day), the IP addresses of bots in the magnified pool and the botnet name are generated as output. The system considers an IP address as behaving similarly to bots in a seed pool if three conditions are satisfied: that address has sent emails to at least a finite number of destinations in the target set, that it has never sent an email to a destination outside the target set, and that it has contacted at least one destination in the characterizing set.

They validated their approach by studying the Cutwail botnet, for which there was direct data available about the IP addresses of the infected machines. The C&C servers that were analyzed accounted for 30% of the botnet, and the validation experiment was run for 18 days. During this period, the spam campaigns were identified using the spam trap, and the seed and magnified pools were generated. Most of the original IP addresses were identified, indicating a good detection rate. Finally, the system ran for a period of four months, during which it tracked close to 2 million IP addresses. Of these, nearly half were from the magnified pools and the rest were seed pools. In an experiment where they use network logs to identify spam bots, the authors showed that their system is data-stream independent .

Vern Paxson (UCB) wondered about the rules in the paper that a bot should have sent a message to an IP address that is not in the seed pool. Paxson asked why that should be the case when a bot can send to a unique destination not in the seed pool. Gianluca answered that this was because the current system did not support that case; in the in future they would make it more general. Jelena Mirkovic (USC/ISI) asked if they had tried dropping that criterion, and Gianluca said that they had not.

Jackstraws: Picking Command and Control Connections from Bot Traffic

Gregoire Jacob, University of California, Santa Barbara; Ralf Hund, Ruhr-University Bochum; Christopher Kruegel, University of California, Santa Barbara; Thorsten Holz, Ruhr-University Bochum

Gregoire Jacob began the talk by presenting a system, called Jackstraws, that will identify command and control connections from bot traffic. Existing techniques for detecting botnets are either host-based (traditional malware detection, signature generation, behavioral monitoring) or network-based (IP blacklists of C&C severs). However, both these techniques are difficult to automate. This is because these techniques require clean C&C logs of system calls or traffic. But getting these logs is difficult, since the traffic could be encrypted. Furthermore, not all the traffic in a bot is associated with C&C activity.

In order to address these issues and identify C&C traffic in bot traffic, the authors propose a system called Jackstraws. The basic rationale behind the system is that C&C traffic results in observable activity at the host such as system modifications, critical information accesses, etc. Hence, the authors combine the network traces with the host-based activity, i.e., they use both a host-based model (system call graphs with data dependencies) and network-related links (every graph associated with a network connection). Another observation is that similar commands will result in similar core activities even if the bots are different. These similarities can be learned using machine learning to identify and generalize C&C-related host activity. This is done using graph mining over known connections and then clustering these graphs to identify similar activities. These graphs are then merged into a template and template matching is carried out to detect C&C activity over unknown connections.

Gregoire then focused on a more detailed explanation of the above basic steps. The system was evaluated on a malware dataset of around 37,000 malware samples comprising over 700 families. After further processing, over 400 templates were generated. They tested these over labeled connections, for which they got a detection rate of close to 80% with a very low false-positive rate but a rather high false-negative rate. Gregoire mentioned that the high false negatives were due to some incomplete graphs. The system was then tested with over 66,000 unknown connections, out of which over 9,000 connections were identified. Among these, over 190 connections were new and not covered by any network signatures. Gregoire concluded by saying that they proposed an automatic system to separate C&C traffic from noise traffic. Their system, which is protocol agnostic, could give more information to analysts and also uncover new malware families that were not present in training.

Rik Farrow was curious why the system picks up families that were not included in the training set. Gregoire answered that, on the network side, the C&C may have a completely different protocol, but that is not the case on the host side. The botnets use the same system calls, in most cases. Also, new botnets are usually created by reusing parts of codes from old botnets. Hence, these behaviors can all be captured from a given template. Christian Kreibich (ICSI) asked how these malware samples were executed in a sandbox. Gregoire answered that the samples were executed using Anubis for four minutes to make sure that they were establishing the connections to the C&C server.

Panel

SSL/TLS Certificates: Threat or Menace?

Moderator: Eric Rescorla, Skype

Panelists: Adam Langley, Google; Brian Smith, Mozilla; Stephen Schultze, Princeton University; Steve Kent, BBN Technologies

Summarized by Nick Jones (najones@cs.princeton.edu)

Each panelist spoke briefly before taking questions from the audience. Schultze argued that there are many fundamental problems with the existing CA model, including that too many people are trusted, trust can be delegated almost infinitely, and accountability is difficult to achieve. He said it's a fundamental problem that users' perception of security is very different from the security they actually have. Additionally, the expansion of the existing Web architecture onto mobile devices exacerbates this problem, by having even fewer UI indicators and a much longer patch cycle.

Adam Langley from Google discussed the mindset of browser vendors who are considering the implementation of new features. At Google, when considering deployment of new features, they consider the possible security gain multiplied by the number of users affected. Specifically, in the case of CA controls for Android, Langley argued that only a small number of users would take advantage of the feature and that it didn't make sense to spend significant development resources implementing it. Langley then discussed several up-and-coming technologies for increased security, such as HTTP strict transport security (HSTS), blocking mixed scripting, and DNSSEC signed certificates. He said that features like strict transport security take priority over fixing the certificate model, because they pose a larger risk and are easier to fix.

Brian Smith from Mozilla's Firefox team pointed out that, from a browser vendor's perspective, there are several requirements which must be met before a new security feature can be deployed: new features must not confuse users, must be fast, and must not be prone to misconfiguration by server admins. He acknowledged that many security-enhancing features cannot meet these requirements, and he endorsed Firefox's extension architecture as a model for testing new security features. Smith discussed DANE, one technique Mozilla is considering implementing for increased certificate security. DANE is currently an IETF draft standard, which proposes using DNSSEC to associate certificates with domain names.

Steve Kent of BBN Technologies advocated the "Mao Zedong approach to PKI," arguing that the fundamental requirement of any CA is to establish and maintain an accurate binding of public key to identity attributes. Kent favors a model with lots of CAs, with a focus on organizational and proprietary CAs. In his model, a proprietary CA serves applications tied to the name space for which the CA is authoritative. Similarly, an organizational CA would serve entities associated with that organization.

During the Q&A, one person asked if the panel would be happy if they lived in an ideal world where all of the technical infrastructure problems were solved. Schultze said that even with the technical problems fully solved, there are still real-world security problems, such as typo squatting, which have to be addressed. Kent said that solving the technical problems is a good first step, but not the entire solution.

Another person asked how DANE can be enforced outside the US. Kent responded that below the DNS root, there are lots of country TLDs. Thus, if a user goes to a URL containing a country TLD, then that TLD will be part of the DNSSEC hierarchy.

Nick Weaver (ICSI) asked about situations in which people choose not to run SSL. He wanted to know if there were any ways to enforce integrity over HTTP without encryption. Langley responded that it is technically possible to do so, but that industry doesn't think anyone would use it in practice.

Diana Smetters (Google) asked about building user interfaces that convey the right security message to users. Specifically, she asked how users should deal with expired and misconfigured certificates, and how normal users should understand what those warnings mean. Schultze responded that user desensitization comes from users seeing too many errors. He argued that browsers should just fail whenever they see a misconfigured cert, because that would force site administrators to be more proactive about fixing these errors. Langley responded that one of the attractive aspects of DNSSEC is its hierarchical delegation, which could reduce the number of errors users see.

One person asked about "trust agility," specifically regarding a Firefox plugin where users decide via consensus whether to trust a certificate. Langley responded that the consensus model places too much burden on users, and that normal users shouldn't be expected to think. Smith responded that users shouldn't have to choose which notaries they trust, because that can devolve into the same problem as choosing which CAs to trust.

Someone asked about browser warnings, and why the browser might not warn a user if Bank of America was using a certificate issued by a Romanian CA. Langley responded that no matter how big the warning, user design studies show that users will bypass them.

Privacy- and Freedom-Enhancing Technologies

Summarized by Ben Ransford (ransford@cs.umass.edu)

Telex: Anticensorship in the Network Infrastructure

Eric Wustrow and Scott Wolchok, The University of Michigan; Ian Goldberg, University of Waterloo; J. Alex Halderman, The University of Michigan

Eric Wustrow presented Telex, a system designed to circumvent blacklisting censors by steganographically hiding requests to prohibited sites in requests to permitted sites. Two observations drove the design of Telex: first, oppressive governments tend to favor IP address blacklists; second, those governments often do not control all intermediate routers. The authors propose inserting Telex stations at intermediate ISPs that are not under censorship. These stations inspect TLS handshake traffic looking for encrypted requests that Telex clients have placed in the TLS nonce field. Upon finding such a request, the station proxies it on the client's behalf and injects responses back into the return traffic.

To a censor shallowly inspecting the client's traffic, the client appears to be connecting only to the permitted site, the path to which contains the Telex station. To the prohibited site, the client's request appears to come from the Telex station. Each client needs Telex client software to run, which would pose a problem for online-only software distribution, but Wustrow optimistically described a system of out-of-band channels (e.g., USB flash drives) through which the software could be passed. Although Telex is not ready for general use—the only "permitted" site is currently a single server at Michigan—Wustrow reported that several of the paper's authors had been using the system full-time for months. He closed with several open questions about how to deploy Telex on the open Internet. Telex software is available at http://telex.cc.

Matthew Green (Johns Hopkins) asked whether the presence of Telex stations in a country provides incentive for censoring nations to attack it. Wustrow responded that the additional motivation provided by Telex was minimal and noted that the US has funded proxy services for oppressed users since 2003. Zack Weinberg asked how Telex would work under routing asymmetry, in which traffic follows one path to a destination and a different path back. Wustrow remarked that putting Telex stations sufficiently close to a desirable prohibited site could probably ensure that the station had access to traffic in both directions; he also suggested that multiple Telex stations on different paths could communicate out of band.

PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval

Prateek Mittal, University of Illinois at Urbana-Champaign; Femi Olumofin, University of Waterloo; Carmela Troncoso, K.U.Leuven/IBBT; Nikita Borisov, University of Illinois at Urbana-Champaign; Ian Goldberg, University of Waterloo

Prateek Mittal described PIR-Tor, a modification of Tor to improve its scalability. When Tor clients join the onion-routing network, they contact a Tor directory server and download a full list of thousands of potential relays through which the client can route traffic. From the full list, the client selects only three relays. Clients currently download the full list in order to prevent malicious directory servers from directing Tor clients to chosen compromised relays. Mittal cited a study showing that directory-listing traffic will soon exceed data traffic on Tor. Noticing that clients use at most 18 middle and exit relays per three hours of Tor use, the authors developed PIR-Tor, a modification of Tor that uses private information retrieval (PIR) techniques to allow clients to fetch a subset of available relays without revealing to the directory server which ones were fetched. In PIR-Tor, a client chooses three candidate guard nodes (initial relays) from the list of directory servers, downloads a small amount of signed meta-information from each, and performs 18 PIR queries to choose its relays.

Crucially, none of these relays learns which relays the client attempted to select, so a malicious relay cannot simply give the client a list of servers with which it colludes. Mittal showed some plots to demonstrate that PIR-Tor exchanges one to two orders of magnitude less directory data with directory servers, arguing for its scalability over the current implementation of Tor.

Matthew Green asked whether PIR-Tor would ever become part of the Tor codebase, to which Mittal replied that PIR-Tor is open source. Roger Dingledine expressed a concern that the authors had not taken all the subtleties of Tor security into account; Mittal directed him to the paper for more information.

The Phantom Tollbooth: Privacy-Preserving Electronic Toll Collection in the Presence of Driver Collusion

Sarah Meiklejohn, Keaton Mowery, Stephen Checkoway, and Hovav Shacham, University of California, San Diego

Sarah Meiklejohn presented Milo, a protocol for privacy-preserving transit payments that, unlike previous systems, enables fine-grained road pricing without revealing to drivers the locations at which their presence is recorded. Meiklejohn gave an overview of previous approaches. In both vPriv and PrETP, presented at USENIX Security in 2009 and 2010, respectively, drivers upload logs of their driving activity to a central authority; the tolling authority performs audits to keep drivers honest. Meiklejohn pointed out a flaw in the previously proposed approach to auditing, wherein the tolling authority sends drivers photos of their cars in certain places and asks them to pay for having been there: drivers can learn the locations of traffic cameras and cheat en masse to avoid them.

Milo, the authors' modified version of PrETP, uses several cryptographic primitives to maintain driver privacy, driver honesty, and audit secrecy. A driver records location/time pairs and forms Pedersen commitments to segment prices. The driver sends the price commitments to the tolling authority along with zero-knowledge proofs of their correctness (as in PrETP) and additionally sends a blind identity-based encryption (IBE) of the commitment's opening. In an audit, the tolling authority's parent organization sends an IBE request to the driver, who responds without learning which segments were audited. Meiklejohn presented performance measurements from an implementation on both Intel and ARM architectures and showed that the time required to audit a driver's activity scales linearly with the number of segments driven.

Matthew Green asked whether audits could be made faster by performing encryptions on the car in advance; Meiklejohn said that it could. Diana Smetters asked how drivers could be given any confidence about their privacy; Meiklejohn acknowledged that that was a fundamental question perhaps more easily addressed in European nations, where citizens trust their governments to take privacy seriously.

Invited Talk

Pico: No More Passwords!

Frank Stajano, University of Cambridge

Summary by Ed Gould (summary@left.wing.org)

Frank Stajano presented a design for a system that would eliminate the need for and use of passwords in interactive authentication protocols. He began by reminding us that, in the past, passwords worked acceptably well. We had only one or two to remember, and 8-character passwords were beyond the scope of brute-force attacks. This is no longer true, and has not been for quite some time. See http://www.fastword.me for some related work.

Users have been told that passwords must have many properties (unguessable, un-brute-forcible, all different, memorizable and memorized), but the intersection of all these requirements yields the null set. Thus, passwords are both unusable and insecure—the worst of both worlds.

The goals of Pico, the author's design for a no-password system, include

- no more passwords, pass-phrases, PINs, etc.;
- scalable to thousands of verifiers;
- no less secure than passwords;
- increased usability (no searching or typing, continuous authentication);
- increased security (no guessing, phishing, eyelogging, etc.).

Two non-goals were mentioned as well:

- zero cost;
- backwards compatibility.

Pico includes a device that is somewhat like a smartphone (although it may be very much smaller), with a few buttons and a display. It has a radio communication facility as well as a camera. It can be shaped like a key fob, a watch, a MP3 player, or jewelry, for example. Importantly, it is a dedicated device, not something running on a multi-purpose device.

The authentication process using Pico involves the Pico device capturing a visual image from the verifier app to which one is authenticating, and a multi-step confirmation of identity, which is different the first time, when the user "pairs" with the app. The app is able to repeatedly communicate with Pico to ensure continued authentication.

Mechanisms for disabling the use of a Pico when it's not in the possession of its proper owner, as well as mechanisms to recover from loss or damage to the Pico, were described as well. Several possible ways to avoid being coerced into using one's Pico were described. A method using "Picosiblings" was described to enable the Pico to operate at all.

Frank pointed out that there are some passwords, e.g., file decryption keys, that do not fit the user-ID/password model, and thus are not addressed by Pico. He also mentioned that optimizing for backwards compatibility may be necessary to get to a critical mass, and quoted Roger Needham: "Optimization is the process of taking something that works and replacing it with something that almost works, but costs less." You can find the paper related to this talk at http://www.cl.cam.ac.uk/~fms27/.

Alan Sherman (UMBC) asked if Frank would comment on the resistance to man-in-the-middle (MITM) attacks. Frank explained that there is little leverage for a MITM attack after the initial pairing. If there were a MITM present during pairing, it might be able to fool Pico. However, it would have to be present for all future interactions as well, or Pico would notice its absence. The multi-channel protocol (camera and radio) makes it harder to do a MITM. If the visual part is hard to use, it is more vulnerable. Carson Gaspar pointed out that nested authentication will be critical for things like command-line administration, allowing for context-valid credentials.

Applied Cryptography

Summarized by Adam Bates (amb@cs.uoregon.edu)

Differential Privacy Under Fire

Andreas Haeberlen, Benjamin C. Pierce, and Arjun Narayan, University of Pennsylvania

Andreas Haeberlen presented work on eliminating covert channels in differential privacy systems. Andreas described several attacks on existing differential privacy implementations (PINQ & Airavat) and presented Fuzz, a new system that addresses these problems. Many companies would like to share their potentially useful data without violating the privacy of the subjects of that data. Anonymizing that data has been shown to be an insufficient defense in the face of adversaries with outside information. Differential privacy solves this problem through a querying mechanism that adds noise to results and the concept of privacy budgets, a method of limiting the number of answerable queries.

Although several attacks are detailed in the paper, Haeberlen focused on one timing attack for the purposes of the presentation. In a properly implemented system where the database is being remotely accessed, an attacker can only observe the query's response, completion time, and her remaining privacy budget. While existing systems secure the query response, it is possible for an adversary to design a query that leaks data through its completion time or privacy budget deduction.

Fuzz is both a programming language and a runtime environment that closes all three of these channels. It employs static program analysis to determine query cost without relying on the database as an input. Predictable transactions ensure that all microqueries take the same time to execute. The runtime environment isolates microqueries, preempts microqueries to execute timeouts, and returns a default value in the case that a microquery cannot complete. The overhead of these defenses is minimal aside from the padding that is imposed by predictable transactions. With ample knowledge of the database, this can be parameterized to reduce this effect. Fuzz is available at http://privacy.cis.upenn.edu/.

Ian Goldberg asked if the system would be susceptible to network-probe timing attacks. Haeberlen responded that the computer was fully busy during processing, and that the machine could be configured to not respond to probes. Ben Fuller drew attention to the overhead imposed by predictable transactions in a large enough database; Haeberlen agreed that the defense comes at a price. Another attendee pointed out that it is necessary to know the machine's hardware configuration to properly set the timeout, and he asked for a clarification regarding the early termination attack.

Outsourcing the Decryption of ABE Ciphertexts

Matthew Green and Susan Hohenberger, Johns Hopkins University; Brent Waters, University of Texas at Austin

Matthew Green presented this work on expediting the decryption of attribute-based encryption (ABE) ciphertexts through outsourced computation. ABE extends identity-based encryption by allowing data to be encrypted to a set of attributes. This is of use in data-sharing environments where records can only be shared with certain groups of people. ABE requires the creation of a ciphertext policy that can grow complex based on the number of attributes. However, ABE's use on mobile devices is limited, due to the rapid growth in ciphertext size and decryption time as the size of the attribute policy increases.

Green presented new versions of Ciphertext-Policy and Key-Policy ABE that allow for outsourcing this decryption to an untrusted cloud service, avoiding the need to share a private key. These new versions introduce a transformation key that is sent to the cloud to perform partial decryption. The secret key is still required to recover the plaintext, so the cloud is not part of the trust model. The performance of this new ABE allows for practical use scenarios on devices with limited computational power. Often, decryption of ciphertexts on a more powerful machine remains an easy task. This new system was evaluated in the wild with an Amazon EC2 proxy. In one test with a complex attribute policy, decryption time was reduced from 17.3 seconds to less than 1.2 seconds. The partial decryption also reduces the size of the plaintext, reducing the cost of transmission. Green identified smart cards and trusted code base reduction as other possible applications of this new system.

Diana Smetters asked Green to elaborate on the key-sharing scheme in his model, pointing out that revocation is difficult. Green explained that every user received their own transform key and that the cloud proxy can act as a reference monitor. Bryan Parno asked if this scheme could be thought of as a regular proxy encryption scheme. Green replied that they are very similar and that both schemes are selectively secure.

Faster Secure Two-Party Computation Using Garbled Circuits

Yan Huang and David Evans, University of Virginia; Jonathan Katz, University of Maryland; Lior Malka, Intel

Yan Huang presented this work on an efficient garbled circuit used for two-party environments. Garbled circuits are a method of making privacy-performing computations; the circuit generator encodes the plain wire signal of 0's and 1's with data-independent nonces, encrypts a truth table, and sends it to the circuit evaluator, who can decrypt one and only one entry in the truth table. The circuit outputs a table of values, only one of which the circuit evaluator will be able to decrypt. Traditionally, garbled circuit execution is slow and scales poorly. Huang presents a new method of garbled circuit generation that is scalable and faster, as well as a library of pre-compiled circuits.

Fairplay, a popular system for secure function evaluation, is impractical for larger circuits, due to speed and memory constraints. This work demonstrates significant improvement through pipelining the circuit creation process—gates are evaluated as they are generated, dramatically improving memory and time efficiency without sacrificing security guarantees. The system is evaluated benchmarking the hamming distance, edit distance, and AES performance problems against previous implementations. Hamming distance experienced a speed-up of several orders of magnitude, and an AES s-box was implemented with a 30% improvement in the number of non-free gates.

Huang concluded that the pipelining technique, along with circuit-level optimization, allowed for garbled circuits to scale to large problem size. This framework and Android app demos are available at MightBeEvil.com. Ian Goldberg commented that he loved this work and hopes to see a trend of people realizing that garbled circuits can be efficiently implemented. He asked if this work can be applied to multi-party problems. Huang responded that much of what was learned in this work can be applied to the multi-party scenario. Diana Smetters inquired about the slow-down of Huang's circuits compared to a native run. Huang replied that it was still several orders of magnitude slower but that this could be a worthwhile cost in security-critical situations.

Invited Talk

The Cloud-y Future of Security Technologies

Adam O'Donnell, Co-founder & Director, Cloud Engineering Immunet

No report is available for this session.

4th Workshop on Cyber Security Experimentation and Test (CSET '11)

August 8, 2011
San Francisco, CA

Opening Remarks

Sean Peisert and Stephen Schwab, CSET '11 Program Co-Chairs

Summarized by Sean Peisart (peisart@cs.ucdavis.edu)

The 4th Workshop on Cyber Security Experimentation and Test (CSET) was held on August 8, 2011. In its first three years, CSET's focus was largely on testbeds and experimentation relating to testbeds, reflecting its origins as the DETER Community Workshop. In its fourth year, the focus was broadened to equally emphasize the nascent science of cybersecurity, i.e., measurement, metrics, data, simulations, and models, as those subjects will also strongly influence the theory and practice of experimental security research. Additionally, the chairs sought to make CSET more of a workshop in the traditional sense. Depending on subject matter, some talks were set up to be highly interactive, 45-minute discussions between presenter and audience. In some cases, similarly themed papers were presented in sessions in which the three talks were presented in 20 minutes each without questions and then all three papers were discussed for 30 minutes together.

Overall, based on reactions from both presenters and audience members, the new scope and format for CSET was a success. Out of 30 submissions, 12 papers were accepted and were well received by the audience.

Security Experimentation and the Real World

Summarized by Peter A.H. Peterson (pahp@cs.ucla.edu)

Should Security Researchers Experiment More and Draw More Inferences?

Kevin S. Killourhy and Roy A. Maxion, Carnegie Mellon University

The rhetorical title of this title was answered immediately by Killourhy with a resounding "Yes!" Explaining, Killourhy stressed that not all empirical work should be considered an experiment, per se, and that experimental practice in security research could be improved. Only 54% of studies in keystroke dynamics were comparative (evaluating a matrix of tools and data sets for comparison on the same grounds), and only 7.5% were inferential (drawing statistical inferences from data, rather than simply reporting results).

A particularly troubling trend in security is the "one-off" evaluation, where a new technology is evaluated against a home-grown dataset. The researcher performs the evaluation, finds a benefit, and declares victory. Unfortunately, in these cases it is impossible to know how well the technology compares to others, because no comparative evaluation was performed (and often neither the dataset nor the tool is public). Comparative experiments—standard in other sciences—show the differences between pairs of techniques and workloads, so as to show their differences.

In addition to comparative studies, Killourhy stressed the importance of statistical inference for experiments, such as comparative experiments. Reporting only which tool performed the best on which data set is not enough, because "security technologies don't have an error rate, they have many error rates, depending on the factors [in the experiment]." In contrast, statistical inference can show not only which tool is best for which data set, but by how much, what the confidence of that result is, what relationships exist between tools and data sets, and more.

A lively discussion ensued, focusing on how to improve security experimentation. Availability and functionality of "research code" hinders good experimentation. Roy Maxion (CMU) suggested adopting "structured abstracts" as used in medicine, where specific language about methodology and result are included, allowing papers to be quickly surveyed. Terry Benzel (ISI) noted that repeatability is often difficult to achieve, even for the original investigator, because of the changing software and hardware environment (and suggested use of testbeds to help mitigate this problem). Cynthia Irvine (NPGS) followed up by pointing out that, often, a sponsor is looking to solve a problem (not to perform comparative studies). The room generally discussed challenges with performing comparative studies with older tools and workloads in the face of rapidly changing threats. Others voiced a need for sponsors to prioritize enabling testing by others.

From there, the discussion turned to facilitating comparative studies, asking whether it would be better to share detectors, or data sets, as well as the difficulties of doing either, which range from proprietary concerns to sensitivity of workload information and more. Killourhy stressed that "what we're doing isn't working," and that almost anything we could do would be an improvement, saying that "the problem doesn't go away because it is inconvenient." The room agreed—but no silver bullet was evident.

No Plan Survives Contact: Experience with Cybercrime Measurement

Chris Kanich, Neha Chachra, and Damon McCoy, University of California, San Diego; Chris Grier, University of California, Berkeley; David Wang, Marti Motoyama, Kirill Levchenko, Stefan Savage, and Geoffrey M. Voelker, University of California, San Diego

Testbeds enable research that could not easily be performed in the real world. However, some kinds of research must necessarily be performed in the real world. This talk described the goals, procedures, and results from some real-world research involving cybercrime.

Two necessary things for engaging with and observing cybercriminals are verisimilitude and scale. Verisimilitude is the quality of being authentic—an important quality when performing research of this kind. If one is pretending to be a customer, it is important to appear to be an authentic customer, regardless of whether you are purchasing end-user goods offered through spam or purchasing computational or other resources offered on the underground market. This can have challenging repercussions; the researchers found it necessary to use a native speaker of Russian in order to navigate the forums and communicate in a natural way. Scale is another important issue; the underground market is a large organization, and it is difficult to see the big picture without a significant and broad effort.

This inspired a long discussion on basically two points. First, people considered the work from an experimental perspective, wondering how researchers could best identify how representative their data was. Geoff Voelker said that when possible, researchers would try to measure similar things from various vantage points in order to try to determine how well the data matched. Related challenges arise due to being blocked or having data "poisoned" by criminals who "got wise" to the investigation. Kanich underscored that they try to be upfront about claims relating to the data and state that the observations are limited by many practical concerns.

The second major topic was about the ethics of this kind of research. One participant said, "Your papers are usually great. How the hell do you get the ethical backing to do this stuff?" Kanich responded that first, funding did not come from government sources, and that they tried to consider whether they truly defrauded the parties involved. They considered that those parties who purchased goods did not need to purchase through them, and they did not keep their money. Furthermore, rather than creating new spam from scratch, the researchers used "double-agent" machines to modify instructions for downstream spam bots that would already have sent spam to potential customers.

A number of people asked about IRB oversight and posited that IRBs are currently, by and large, medically oriented and are not sensitive to cybercrime issues. The researchers described their relationship with IRBs, lawyers, and funding, and stated again that they take a consequential approach, asking whether they would do harm in the course of the research. Additionally, the papers for their major studies each include a section on the ethics of their methodology and actions. During this discussion, Doug Maughan (DHS) highlighted the forthcoming Menlo Report on ethical principles for ICT research and suggested that ICT researchers should, like Dave Dittrich, find their way onto IRBs for the future.

Ultimately, important, timely, and fascinating data resulted from—and will continue to come from—this ongoing research. At the same time, the inevitable debates about representivity, ethics, legality, and funding will continue alongside them.

Experimental Methodology

Summarized by Peter A.H. Peterson (pahp@cs.ucla.edu)

Salting Public Traces with Attack Traffic to Test Flow Classifiers

Z. Berkay Celik, Jayaram Raghuram, George Kesidis, and David J. Miller, Pennsylvania State University

George Kesidis argued for greater "statistical hygiene" in security experimentation. Their work focused on identifying, discussing, and attempting to mitigate the way in which the timing characteristics of sample botnet traces used for evaluating flow-classifiers can inadvertently affect the results.

Due to the lack of publicly available traces of attack traffic, many researchers construct synthetic attack traffic as training and testing targets for their flow classifiers. One technique is to combine benign background traffic from a corporate network and traffic from a defanged botnet running in a testbed. In this way, the background traffic is salted into the botnet traffic, providing an ostensibly realistic trace.

However, when flow characteristics of the botnet traffic are statistically distinct from the background traffic, they may be identified by machine learning algorithms as meaningfully identifiable characteristics of botnet traffic, even though they may be artifacts of the trace synthesis process. In turn, this can make the flow classifiers appear to be more successful in evaluations than they may be in real life, because the synthetic trace can be artificially straightforward to classify. In the paper, the authors worked to investigate and overcome these issues, including comparing how various scenarios and machine-learning algorithms combined to produce various results. Researchers in this area would do well to consult the paper for more details.

The focus of the talk and discussion was more about these kinds of issues in general, and expanded beyond the paper itself into issues of experimentation and statistical forensics. For example, another classic issue affecting research results is "double dipping," such as when training sets (or sets used to derive ground truth) are used as targets for testing. This is a specific problem for machine learning, but also affects any research where the evaluation phase can be unintentionally (or intentionally) biased toward the solution.

Discussion for this session was combined with the next two papers.

Beyond Simulation: Large-Scale Distributed Emulation of P2P Protocols

Nathan S. Evans and Christian Grothoff, Technische Universität München

This paper was presented by Matthias Wachs and Bart Polot.

When test requirements grow larger than even significant testbeds can handle, researchers often turn to simulation. However, the fidelity of the simulation can be poor, because of inadvertent mistakes when the simulation is constructed from the real-world counterpart. And, in any case, the process of building a simulation can have a large cost in terms of time and human resources.

While simulation allows great scale, it has a high translation cost. On the other hand, the scale of emulation solutions may be limited, but allows the experimenter to acquire data that directly reflects the original implementation of the tool in question. Accordingly, Wachs and Polot presented the GNUnet framework, which is a scalable emulation framework for peer-to-peer protocols, capable of accurately supporting many emulated hosts through judicious sharing of local resources. They described the resource-sharing design of GNUnet, which includes the use of shared memory and fast messaging techniques as well as centralized management of peers. They also described their experience testing an 80,000-peer emulation of a Kademlia DHT on a small cluster as a test case and example of the frameworks.

GNUnet makes some tradeoffs in order to achieve its goals. For example, it does not support timing control, so it may not be suitable for latency-sensitive tests. There are also other characteristics of the design that may affect its suitability for particular purposes, such as interference from the underlying OS, scheduling, similarity of peers, etc. And, of course, test code must be written using the GNUnet framework, which has its own cost. Interested readers should see the paper or presentation audio for more information.

Automating Network Monitoring on Experimental Testbeds

Michael Golightly, Princeton University; Jack Brassil, HP Laboratories

Jack Brassil presented this work on Netflowize, a prototype tool for Emulab-type topologies that is able to automatically add experiment-wide instrumentation nodes to the topology. This ability, along with a set of tools, allows for flexible monitoring of resource consumption across the test environment. Netflowize uses existing NetFlow probes and collectors available on Emulab and DETER; while these tools are available on these testbeds, they are typically used by testbed operators. Netflowize allows researchers to leverage these tools in a straightforward and accurate way.

Netflowize automatically determines where in the experimental topology to place the probes. Where a naive approach could add too many probes, Netflowize minimizes this. Netflowize has two modes: "lightweight" mode uses existing infrastructure to perform monitoring in a transparent way; "heavyweight" is also transparent, but deploys additional hardware resources in the experiment to serve as the probes. This has the benefit of not creating load on experimental nodes, but requires more hardware.

Netflowize is under active development. Future work includes better error and redundancy handling, more user-accessible "knobs," and efficiency improvements. Other developments may include extending beyond NetFlow to use other tools, and more. Brassil pointed the participants to a URL where prototype code was available (contact him if you are interested).

Discussion

Following this presentation, the floor was opened to questions from the workshop participants to the authors of all three papers.

Stephen Schwab (ISI) asked George Kesidis about best practices for the application of machine learning in experimentation. Kesidis responded that you don't need an expert, but "good statistical hygiene" is a must. I took that to mean that new users of ML techniques may not recognize the necessity of separating testing and training sets, even though they might not make those kind of "double dipping" mistakes in other experimental areas.

Roy Maxion (CMU) asked all three presenters what they felt made rigorous experimentation with high-confidence results hard or easy. Jack Brassil suggested that if our community (like others) would centralize to common, shared tools, it would be easier to verify results and insist on more rigorous experimentation. Kesidis suggested that reproducibility is achievable, but that it is too hard to adequately specify experimental conditions within the confines of a conference paper. Kesidis did say that we can point readers to online resources. Kesidis also said that if you're doing research, your goal should be to prove your results and enable them to be accepted. This might include open sourcing the work and making sure that others are able to recreate it (subject to confidentiality concerns, etc.). He said, "It's not that other fields are that much better than we are," but we should still be doing it.

Jelena Mirkovic (ISI) suggested that recreating experiments isn't always very straightforward—even if the code is open source and available. Matthias Wachs suggested that this could be solved with better communication between researchers, and accordingly invited the participants to email them directly with any questions regarding GNUnet.

Bots and Overlays

Summarized by Kevin Killourhy (ksk@cs.cmu.edu)

Challenges in Experimenting with Botnet Detection Systems

Adam J. Aviv and Andreas Haeberlen, University of Pennsylvania

Adam Aviv described a hypothetical situation in which a researcher evaluates a new botnet detector. Ideally, the researcher conducts a series of representative tests, both on her own network and also on others'. She compares the results to prior work, and she makes the detector and data available so that other researchers can reproduce her results. A survey of 20 research papers on botnets suggested that this ideal is often sacrificed to practicalities. Challenges include establishing ground truth, creating a production-quality detector, and obtaining permission to deploy it.

Behind these challenges, Aviv said, the big concern is privacy. His statement prompted a lively discussion about whether privacy is the biggest problem—compared to lack of ground truth, standard methodology, and statistical analysis—and whether those other problems can be solved without tackling the privacy issue. Aviv explained that if privacy were not a problem, researchers could share data and do apples-to-apples comparisons of different detectors. It was suggested that, if nothing else, privacy is a good excuse for not sharing data.

Returning to the survey of botnet-research papers, Aviv showed that the majority of papers used an overlay methodology for their evaluation. For this method, two separate network traces are needed: botnet traffic from a simulation or sandbox, and standard traffic from the researchers' network or another source. Then the two traces are integrated into a data set for detector evaluation. A participant observed that one cannot know that the standard traffic is clean (i.e., untainted by bot traffic). Aviv agreed and raised a host of other concerns, including the introduction of artifacts by overlaying traffic from different networks.

Inspired by how PlanetLab helped distributed-systems researchers, Aviv asked, "Can we do better together?" To start the discussion, he sketched out a straw-man solution: operators of various networks give researchers access to a box on their network. The boxes would be fed NetFlow data, and researchers could install their detectors on boxes across many networks. Detector output would be vetted by the network operators; when free of sensitive data, the results would be returned to the researcher for further analysis. Participants discussed whether incentives such as access to bleeding-edge detectors would encourage network operators to participate. In the end, Aviv offered this solution as a place to start; improvements are welcome.

ExperimenTor: A Testbed for Safe and Realistic Tor Experimentation

Kevin Bauer, University of Waterloo; Micah Sherr, Georgetown University; Damon McCoy, University of California, San Diego; Dirk Grunwald, University of Colorado

Kevin Bauer introduced Tor as being, simultaneously, a production-quality public service and an ongoing research project. Tor is a low-latency overlay network on which users can send and receive TCP traffic anonymously. At the same time, researchers are constantly modeling, testing, and improving the network. These two aspects of Tor sometimes come into conflict, causing researchers to adopt a range of research methods, with mathematical models at one end and worrisome use of the production Tor network at the other. Simulators, like Emulab, and PlanetLab make up the middle of this range.

A good experimental testbed for Tor research would scale to Tor-like size, ensure reproducibility, have realistic traffic properties, and be otherwise ethically sound and easy to use. With PlanetLab, one quickly runs into scalability and reproducibility issues; resources are limited, and node assignments change from one allocation to the next. With the Tor network itself, scalability is not a problem but reproducibility and ethical issues are; Tor users expect privacy—-the very reason they use Tor.

Bauer explained the design of the ExperimenTor testbed (http://crysp.uwaterloo.ca/software/exptor). Realistic routers were modeled using publicly available data. Clients were modeled by studying aggregate real-world traffic (e.g., amount of traffic per service and number of clients per country). The ModelNet emulation framework was chosen since many applications can be run unmodified on the testbed just by linking with the ModelNet libraries. Early experience with the testbed is promising. An emulator has been deployed across four institutions and used in two ongoing research projects: effects of link-based router selection, and a redesign of Tor's congestion control.

Discussion focused primarily on two issues: the general utility of the testbed and the reproducibility of Tor research. One participant noted that this approach seemed like a procedure for network-based research in general, not just Tor research. Bauer agreed that the ModelNet emulation framework is very general, but his present efforts are focused on Tor. Participants also reacted to the issue of reproducibility with experiments on real networks and other testbeds. One participant wondered whether results that cannot be reproduced are worth reporting.

Methodology and Getting Real Data

Summarized by Kevin Killourhy (ksk@cs.cmu.edu)

On the Design and Execution of Cyber-Security User Studies: Methodology, Challenges, and Lessons Learned

Malek Ben Salem and Salvatore J. Stolfo, Columbia University

Malek Ben Salem explained how work on masquerade detection has been hindered by a lack of masquerade data. For instance, she wanted to test the conjecture that an attempt to steal information often manifests as extensive and abnormal search behavior. To test such a claim, one needs data not only from legitimate computer usage but also from attempts to steal information. Observing attack-like behavior under laboratory conditions can be a challenge, and this talk concerns her experiences trying to add rigor to the process of gathering such data.

Ben Salem enumerated steps for designing and conducting a user study: state a hypothesis, identify experimental variables, establish a control group, choose a sampling procedure, and estimate the necessary sample size. She emphasized the need to control variability and reduce bias among users. In practice, these steps require some careful thought. For instance, to ensure that subjects acted as they might if they were participating in a real attack, she provided them with scenario narratives. Subjects were told to imagine themselves at a co-worker's unattended computer, trying to find sensitive financial information.

For discussion, she offered several recommendations and lessons learned. Get IRB approval early, since the process can be slow. For data that is released publicly, subjects should sign waivers regardless of the planned data sanitization; subjects do not always have the understanding or foresight to thoroughly sanitize their data. Conduct pilot tests and post-experiment questionnaires to identify any unforeseen issues and provide additional insight. As an example, the preliminary narrative did not provide a name for the imaginary co-worker. Talking with pilot subjects revealed that they would have used the name when searching for information, and so a name was added.

Experimental Challenges in Cyber Security: A Story of Provenance and Lineage for Malware

Tudor Dumitras, Symantec Research Labs; Iulian Neamtiu, University of California, Riverside

Tudor Dumitras led with an anecdote about what happens when researchers ignore issues of experimental validity: the IROP Keyboard,a piece of hardware satirically proposed by Zeller, Zimmermann, and Bird, does not have the I, R, O, or P key because studies have shown that most coding errors involve those keys. More serious issues of validity arise when tracing the lineage and establishing the provenance of a malware artifact.

When analyzing malware samples to determine which one came first and how they evolved, several methodological questions arise. Is the reconstructed lineage correct (i.e., what is ground truth)? What am I really measuring, and are my data representative? How well does this work now, and, more important, how well will it work tomorrow? These questions assess the validity of an experiment. Threats to validity come in several forms—*construct validity*, whether the metrics measure the right concept; *internal validity*, whether causal inferences can be drawn; *content validity*, whether all data relevant to the concept are used; and *external validity*, whether the results generalize beyond the experiment.

Within the domain of tracing lineage and establishing provenance of malware artifacts, Dumitras explores these threats to validity and offers some solutions. For instance, ground truth is somewhere between hard and impossible to establish for malware lineages, but the same tools can be used to reconstruct the lineage of open-source software (e.g., the Linux kernel). Dumitras also offered Symantec's WINE (Worldwide Intelligence Network Environment) as a helpful service for promoting experimental validity (http://www.symantec.com/WINE).

WINE provides researchers with real-world malware data, gathered as part of Symantec's own security and anti-virus efforts. The data enables research on both static and dynamic properties of malware. Metadata and contextual information are provided for the artifacts (e.g., collection times and infection reports). WINE makes possible reproducible experiments with representative malware samples. Dumitras fielded several questions on the logistics of using the service and whether it might be abused by malware authors or competitors. Users must be on-site at a Symantec Research Lab and under NDA. Dumitras and his colleagues are exploring what IRB-related issues arise for researchers using the data. According to NSF sources, the cost of using this service could be included in a grant budget.

Education

Summarized by Kevin Killourhy (ksk@cs.cmu.edu)

Active Learning with the CyberCIEGE Video Game

Michael Thompson and Dr. Cynthia Irvine, Naval Postgraduate School

Michael Thompson describes CyberCIEGE as a cyber-security game for a broad audience. The game enables instructors to cover a wide variety of security concepts without requiring a lot of prior knowledge of the students. Players are able explore a scenario, approach it in different ways, and even fail as part of the learning process. A scenario-definition language can be used to add new scenarios to the game. Quantitative assessments can also be included, leading one participant to wonder if the game might be used to test the competence of his organization's IT department.

Scenarios involve a simplified network simulation including assets, users trying to access the assets, and attackers trying to compromise them. Players can add computers, routers, and firewalls to the network. They can change ACLs, apply patches, and even adjust aspects of physical security. However, as Thompson explained, students get the experience of configuring a VPN without first going through a CISCO training course. As a demonstration, Thompson walked through one game scenario. The player helps a user access the Web, installs a network filter, and protects trade secrets by isolating another computer from the network.

While there have been no formal assessments, game scenarios are used in Intro to Computer Security at the Naval Postgraduate School, and the game is being piloted at other institutions. Informal feedback suggests that students enjoy the hands-on aspect of exploring networking concepts through the game. One of the lessons learned is that different students approach a scenario in vastly different ways, and the game must provide a lot of feedback to students as they explore a scenario.

Investigating Energy and Security Trade-offs in the Classroom with the Atom LEAP Testbed

Peter A.H. Peterson, Digvijay Singh, William J. Kaiser, and Peter L. Reiher, University of California, Los Angeles

Peter Peterson presented Atom LEAP, an energy-measurement platform, and described his experience using it in an undergraduate research seminar. LEAP is open source, inexpensive, and easy to build. The "energy calipers" provide overall and component-level energy usage (e.g., CPU, RAM, and USB) at very fine granularity. User scripts make it easy to start and stop monitoring.

In the seminar, students used LEAP to investigate security/energy tradeoffs. Instructors presented students with topic areas; the students organized into groups within each area and developed research plans. After the first two weeks, class time was used for group meetings rather than lectures. Through the projects, the instructors intended to have students learn about performance measurement and analysis.

One project measured energy consumption of an AV product scanning a home directory. Another project compared the energy consumption of various compression/encryption schemes, finding that gzip was the most energy efficient. Peterson observed that the projects and the style of the course resembled what a student would encounter in grad school, and that this experience was beneficial for undergrads.

In retrospect, supporting many different topic areas was a lot of work for the instructors. Future iterations of the class might have multiple groups tackle one topic, so groups could red-team other groups' plans. Because the quality of the final reports was uneven, additional coverage of experimental design and statistical analysis is also planned. Nevertheless, students got the message that evaluation is tricky. Peterson encouraged interested instructors to pilot the LEAP technology in their own courses.

Experiences in Cyber Security Education: The MIT Lincoln Laboratory Capture-the-Flag Exercise

Joseph Werther, Michael Zhivich, and Tim Leek, MIT Lincoln Laboratory; Nickolai Zeldovich, MIT CSAIL

Joseph Werther posed the question: How do we get more smart students involved in cybersecurity? He described an effort by MIT Lincoln Laboratory to conduct a capture-the-flag (CTF) event intended to educate and promote interest in security; 53 students from six universities participated. The Wordpress content-management system was chosen as the target; it is a realistic target, and its extensible nature allows students to become comfortable with the base system. Additional components can be tackled incrementally.

The two-day CTF event was preceded by a week of lectures and laboratory exercises. Underlying the effort was the belief that education in "offensive security" enables students to understand the mechanics of a vulnerability and how a system can be exploited. Werther identified three components of learning: reading, building, and experience. All three were incorporated into the lessons. The five classes included coverage of Web applications, Wordpress, server security, and Web-application security. The final class was a work-through of the Google Gruyere hacking exercises. For the CTF event, teams had to defend a Wordpress instance running in a virtual machine and to attack the other teams' instances. A team's score incorporated offensive success, based on how many other teams' flags were captured, and defensive success, based on how few of their own flags were compromised.

After the exercise, participants filled out a voluntary survey. While acknowledging that the results were not scientific, Werther noted that respondents reported increased interest as well as skill in security work. The organizers plan to conduct more CTF exercises in the future. They hope to expand to more colleges and to improve the robustness and extent of their game monitoring. In the meantime, they are investigating the best way to encourage learning and assess knowledge-gain.

Discussion Panel

Michael Thompson, Peter A.H. Peterson, and Joseph Werther

The panelists were asked what each of their educational efforts replaced. Werther noted a dearth of capture-the-flag exercises, especially with education, not competition, as the principal goal. Thompson explained that CyberCIEGE is used in introductory labs but did not know what exercises they replaced. Peterson explained that, prior to the seminar using Atom LEAP, no such course was offered.

When asked how they measured their learning objectives, all three panelists acknowledged that they were reporting on pilot-stage experiences. They were thinking about how to improve assessments the next time around. One participant suggested that they look to the scientific literature on education and learning.

The panelists were asked whether their experience showed that anyone can learn cybersecurity. Thompson answered that what one needed more than anything was interest. Without an interest in the subject, learning about cybersecurity would be very difficult. For those with interest, Thompson's experience suggested that one could improve success rates by accommodating different learning styles.

In the end, the discussion turned to *Star Trek* and the Kobayashi Maru. Should cyber-security education include an unwinnable scenario? Would facing such a situation be a valuable lesson to students of security? Perhaps.

USENIX Workshop on Free and Open Communications on the Internet (FOCI '11)

August 8, 2011
San Francisco, CA

[Note: For the remainder of the workshop program, including full papers and presentation slides, see http://www.usenix.org/events/foci11/tech/.]

Measuring Censorship

Summarized by Nick Jones (najones@cs.princeton.edu)

Three Researchers, Five Conjectures: An Empirical Analysis of TOM-Skype Censorship and Surveillance

Jeffrey Knockel, Jedidiah R. Crandall, and Jared Saia, University of New Mexico

Knockel began by introducing TOM-Skype, a modified version of Skype produced by TOM Group in China. When Skype users in China attempt to download Skype, they are automatically redirected to TOM-Skype. Since TOM-Skype is interoperable with regular Skype, it uses the Skype network for all of its communication. Due to this, TOM-Skype performs censorship locally on users' computers. TOM-Skype performs this monitoring using keyfiles, which are lists of encrypted words to monitor for.

In this work, the authors reverse engineered the cryptographic algorithm used to encrypt the keyfiles. They approached this problem by using known blocked words in conversation, and monitoring the program's behavior. Notably, the latest version of TOM-Skype (5.1) contains two separate keyfiles. One keyfile triggers a surveillance message which is sent to TOM Group, while the second keyfile triggers both surveillance and censorship of the user's conversation. From this work, the authors propose five conjectures which they believe are a useful model for studying Internet censorship: (1) censorship is effective, despite attempts to evade it; (2) censored memes spread differently from uncensored memes; (3) keyword-based censorship is more effective when the censored keywords are unknown and online activity is, or is believed to be, under constant surveillance; (4) the types of keywords censored in peer-to-peer communications are fundamentally different from the types of keywords censored in client-server communications; (5) neologisms are an effective technique for evading keyword-based censorship, but censors frequently learn of their existence.

One audience member asked if the authors retained copies of the sets of blocked keywords that TOM-Skype has used over time. Knockel said that the keywords were retained, and that they may analyze the changes in future work.

Fine-Grained Censorship Mapping: Information Sources, Legality, and Ethics

Joss Wright, Oxford Internet Institute; Tulio de Souza, Oxford University Computing Laboratory; Ian Brown, Oxford Internet Institute

Wright argued that every country engages in censorship at some level, and that it is useful to examine censorship at a more fine-grained level than national borders. In this work, the authors used DNS servers across China to check for blocked Web site addresses. They tested 278 DNS servers, and performed a DNS query for each of the top 80 blocked Web sites. Different servers within China provided different results for the same blocked Web sites. Some servers listed the site as non-existent, others returned no results, and some redirected a user to Beijing before returning no result.

In addition to the censorship study, Wright discussed the challenges inherent to studying censorship problems. He talked about the legal and ethical implications of asking end users to access blocked Web sites, specifically when doing so may place these users at some risk. He stressed the importance of getting informed consent from participating users, and this inspired a discussion about best practices for communicating risks to users.

During Q&A, one audience member asked about the difference between a user requesting access to a Web site and a user successfully accessing the Web site. Wright responded that this depends on which country the user resides in, and that it is necessary to examine the laws of each country. Another person asked about building censorship detection tools which incorporate plausible deniability. Wright responded that the problem has numerous ethical and legal challenges that must be addressed before we could build such tools.

CensMon: A Web Censorship Monitor

Andreas Sfakianakis, Elias Athanasopoulos, and Sotiris Ioannidis, Foundation for Research and Technology, Hellas

Sfakianakis began by discussing the dynamic nature of censorship and the drawbacks of existing tools for detecting and reporting censorship. Sfakianakis then introduced CensMon, a distributed system for detecting censorship at a global level. CensMon is designed around a central server, which uses a network of agents to report censorship. CensMon agents monitor multiple systems including Twitter, Google Alerts, and Google Trends in order to extract URLs for censorship checking by the agent network. Additionally, CensMon checks for filtering at different protocol levels, including DNS filtering, IP blocking, and changes in accessibility to known censored Web sites. The initial deployment of CensMon was tested over 14 days with agents in 33 countries: 86% of the agents reported no filtering, with China's agent node reporting 176 censored domains.

In addition to domain blocking, CensMon can detect partial content censorship, such as news articles which have been changed by a censor. In their initial study, while 3% of the URLs tested saw some content changes, no partial content filtering was detected. Sfakianakis argued that one major advantage of the CensMon system is that monitoring multiple streams of information in multiple locations provides an ability, under certain circumstances, to detect content that has been modified but not completely blocked.

Audience members asked whether CensMon can handle dynamic content. Sfakianakis replied that CensMon currently only handles Web pages that have an "article-like" format. Can CensMon handle syndicated content such as newspaper stories? CensMon does not have any special handling of this type of content. Is it possible to use Tor exit nodes as CensMon agents? This is technically possible, but not implemented in the current CensMon software.

Work-in-Progress: Automated Named Entity Extraction for Tracking Censorship of Current Events

Antonio M. Espinoza and Jedidiah R. Crandall, University of New Mexico

Espinoza presented this study, which analyzes censorship in China by performing named entity extraction on Chinese-language sources to pick out people, places, and other relevant terms from news texts. The authors trained their named entity extractor on the Chinese-language version of Wikipedia, using part of Wikipedia as a training set and part as a test set. The authors then queried Chinese search engines with phrases extracted from their training data. They repeated these searches every 12 hours, looking for changes in the returned results, as well as GET request censorship. Several sensitive terms were discovered, such as "nobel prize," "norway," and "jasmine flower."

Espinoza said that the list of censored terms used for GET request censorship is relatively static and slow-changing. In the future, the authors hope to improve their named entity extraction and to support other languages. Additionally, they would like to add other input sources into their monitoring study, such as the list of censored words used by TOM-Skype.

One audience member asked about the time frame during which these experiments were run. Espinoza responded that they were conducted during a two-month period around the beginning of 2011. Another question addressed the possibility that the queries might return different results when executed within China. Espinoza acknowledged this and said that they have not yet been able to test that, but would like to do so in future work.

5th USENIX Workshop on Offensive Technologies (WOOT '11)

August 8, 2011
San Francisco, CA

Attacks on Networks and Networking Equipment

Summarized by Rik Farrow (rik@usenix.org)

Media Access Control Address Spoofing Attacks against Port Security

Andrew Buhr, Dale Lindskog, Pavol Zavarsky, and Ron Ruhl, Concordia University College of Alberta

Andrew Buhr explained how enabling port security increased the chances of success for an attacker when spoofing MAC addresses. Port security means giving a higher precedence in a switch-based lookup table over non-secure MAC addresses. In their experimental setup there are three switches, with two edge switches set up with port security and a third switch used to connect the edge switches configured without port security. Cisco advises that configuration, for several reasons.

Andrew described two of the three attacks that appear in their paper. In each described attack, the attacker is connected to the same switch as one of the victims. The second victim is connected to the other edge switch. When the first victim sends an ARP request to the second victim, the attacker can replay the same ARP reply. Because the second victim's reply comes via the relaying switch, the response is considered non-secure. So the attacker's spoofed ARP reply results in associating the second victim's MAC address with the attacker's switch port, allowing the attacker to impersonate the second victim.

Andrew suggested several techniques as defense strategies, with the preferred method being segregating trusted and non-trusted nodes into their own broadcast domains. Enabling port security on the interconnecting switch is not recommended by Cisco, because it disables channel bonding and dynamic port reconfiguration.

Mike Ryan (ISI) wondered if the second attack would work if there were two attackers, each connected to a different edge switch, with a private backchannel. Andrew expected this attack would work, as long as the attackers could forward frames quickly enough that there were no retransmissions of frames. David Brumley asked if the version of IOS made any difference to the attacks, and Andrew said that it did. David then asked about availability of code, and Andrew replied that no code is necessary, as all the attacker needs to do is change the MAC address for the attacking system's interface.

Fragmentation Considered Vulnerable: Blindly Intercepting and Discarding Fragments

Yossi Gilad and Amir Herzberg, Bar Ilan University

The researchers took a new look at an old problem. In the 1990s, there were several well-known DoS attacks that relied on problems with IP fragmentation: Teardrop, Rose, and the Ping of Death, all based on implementation mistakes. In this work, the authors rely more on specification issues.

IP fragmentation is best avoided, but still occurs today. With IPv4, any router can fragment packets, and in IPv6, only the sending host can fragment packets. ICMP is used to determine Path MTU to avoid fragmentation, but fragmentation can still occur with UDP and when packets are tunneled.

The key to their attack is to determine the IP ID. This is trivial with Windows, which uses a monotonically increasing IP ID. Linux uses a per-destination IP ID, which makes determining the IP ID more difficult. In their attacks, they make use of a sandboxed script, PuZo, on the victim's network, to watch for fragments that do not show up. The missing fragments must have had a valid IP ID, and thus are not directed to PuZo. Their attack requires O(square root of N) probe packets.

Mike Ryan asked if this attack worked against firewalls, and the speaker said it does. Someone else pointed out that Linux can be patched to use randomized IP IDs, and the speaker replied that this can cause collisions and be as bad as an attack.

Killing the Myth of Cisco IOS Diversity: Recent Advances in Reliable Shellcode Design

Ang Cui, Jatin Kataria, and Salvatore J. Stolfo, Columbia University

Ang Cui presented this paper, which describes a very effective exploitation of Cisco routers. Internet infrastructure is highly reliant on Cisco routers,and cannot be defended against attacks like this as the use of a firewall or IDS is, in many instances, not possible.

Felix Linder (FX) has pointed out that ASLR (address space layout randomization), as well as the many different versions of IOS, make successful exploitation of Cisco routers difficult. The authors estimate that there are over 300,000 binary versions of IOS. Yet IOS is a functional monoculture: in any router you will see the same behavior when you enter the enable command. FX created a disassembling shellcode that relies on finding a known string, then searching for the address of this string in code, and, finally, replacing the instruction that reports the receipt of the correct password with an instruction that always reports success.

The problem with that attack is that it takes a long time. IOS includes a watchdog timer, to guard against runaway processes, with a two-second limit. The two linear searches through memory used by FX's exploit often trigger the watchdog timer, killing the process.

The authors' approach relies on a single search of a more limited amount of memory. Interrupt handlers use the eret (exception return) instruction, and their shellcode searches for these instructions and replaces them with hooks into their own rootkit. As this search and replace occurs quickly and future execution is distributed across many processes, because it relies on interrupt handlers, the attack does not get caught by the watchguard timer.

The first stage rootkit monitors packets punted to IOS—any that cannot be handled by line cards. If these packets contain a 32-bit magic number, the next four bytes are used as an address, the following byte a flag, and the rest of the packet is loaded into memory as executable code. Using these packets, a more full-featured rootkit can be loaded into the router's memory and controlled using punted packets labeled with the magic number. But before this can be done, the first stage rootkit must return the locations of the eret instructions, as these provide a fingerprint that identifies the specific version of this binary instance of IOS. The second stage rootkit is tailored for this binary instance.

Ang described a possible defense against their attack, the creation of "symbiotes." The symbiotes run checksums on the invariant portions of IOS to detect the installation of rootkits, and this is future research for the authors. A video (http://www.hacktory.cs.columbia.edu/ios-rootkit/) of a successful attack was shown, eliciting applause.

Adam Drew (Qualcomm) asked Ang to explain the principle behind their proposed defensive software. Ang said that their defense was designed to work on blackbox systems, like IOS, where the internal workings are unknown. They can intercept a large number of returns, perform checksumming, and rely on having enough symbiotes to make it very difficult for an attacker to disable or avoid them all. Amiz Herzberb (Bar Ilan University) wondered if increasing the number of IOS versions might help, but Ang said that their attack relies on a database of versions, and increasing the version space makes little difference to their attack. Someone else wondered how successful the first stage attack needs to be to collect fingerprints, and Ang answered that that depends on the particular attack. They used a synthetic attack for their demo. Someone else asked how they created the final rootkit using the fingerprints, and Ang answered that this can be done automatically once the binary version is discovered.

SkyNET: A 3G-Enabled Mobile Attack Drone and Stealth Botmaster

Theodore Reed, Joseph Geis, and Sven Dietrich, Stevens Institute of Technology

Theodore Reed described, and later demonstrated, the use of an inexpensive drone to enlist participants in a botnet. A collection of such drones would be called a SkyNET, but the authors built only a single drone, using an off-the-shelf AR.Drone quadracopter platform, a TS-7550 single board computer with 3G, GPS, and two WiFi cards. The drone is intended to fly around an area searching for WiFi networks. Breaking WEP and WPA encryption is offloaded to the cloud (EC2), and an OpenVPN connection over the GSM link is used for communication back to the command and control (C&C) system.

Their demonstration model can fly 2.7 kilometers. During testing in New York City, they found over 1700 access points near the Empire State Building and another 1100 in a residential area. Ted commented that "just flying the drone attracts victims," who came up to them as they flew the drone from city parks.

They included a couple of mechanisms to protect against the accidental loss of the drone, and potentially the information against the systems it had compromised. The drone includes a list of pairs of asymmetric keys, and the keys are randomly assigned to bots with the ID of the keys kept at the C&C system. Ted concluded by saying that without any engineering background, they had built a usable drone for about $600 that could carry out attacks against WiFi-enabled systems.

David Brumley asked how noisy the drone is. When the drone was demonstrated, its four rotors were about as noisy as a vacuum cleaner; as Ted said, its noise detracts from its "stealthiness."

Crossing into the Real World: Beyond IP-based Attacks

Summarized by Karl Koscher (supersat@cs.washington.edu)

Getting the Face Behind the Squares: Reconstructing Pixelized Video Streams

Ludovico Cavedon, Luca Foschini, and Giovanni Vigna, University of California, Santa Barbara

Ludovico Cavedon presented this paper, which looks at the effectiveness of pixelization filters for video. These filters are often used to obscure private or censored information (e.g., faces and license plate numbers) while remaining somewhat aesthetically pleasing and keeping the broader image context intact (as opposed to using a black box, for example). While it is often assumed that these filters cannot be inverted, this paper demonstrates that in many cases, close approximations of the original images can be reconstructed.

The paper makes the following assumptions: first, the image being pixelized must be approximately fixed (e.g., a license plate). Second, the pixelized area of the image must be fixed between frames. Finally, there must be some small motion between frames of the image.

Since pixelization is a linear operation, a naive approach is to simply build a system of equations describing the pixelization and solve it approximately. However, this produces unsatisfactory results (see figure 4(c) in the paper for an example), due to quantization error. Instead, the approach presented uses the maximum a posteriori method, which takes advantage of the fact that the solution is not arbitrary but represents an actual image.

Ludovico concluded his talk with several impressive demos of their technique, which are also shown in the paper. He also discussed issues with real-world video, such as compression artifacts, image rescaling, and moving subjects.

Heat of the Moment: Characterizing the Efficacy of Thermal Camera-Based Attacks

Keaton Mowery, Sarah Meiklejohn, and Stefan Savage, University of California, San Diego

Sarah Meiklejohn presented this paper, which looks at the effectiveness of using thermal imaging technology to recover codes entered into code entry devices (e.g., ATM PIN pads). While previous work demonstrated that this type of attack was feasible against a particular type of safe, this research dives deeper and evaluates the effectiveness along four different axes: different types of material, different types of people and their code entry techniques, different scales of attack (e.g., automatically recovering hundreds of PINs versus manually identifying one), and different degrees of success (e.g., recovering the code with multiple attempts).

The attack works poorly against metal, which reflects and dissipates heat rather well. Therefore all of the results presented were for plastic and rubber keypads only. For their experiments, they recruited 21 people to enter a total of 27 codes (seven of which had repeated digits). They discovered a wide variance in the amount of heat transferred by different people. In all of their experiments, automated recovery outperformed manual analysis. Recovery of perfect codes was rather low (~15% for manual analysis, ~35% for automated analysis), even immediately after code entry. Recovery of codes with one error (a substitution or transposition) was also low (~25% manual, ~50% automatic). Recovery of perfect key combinations (but not necessarily order) fared significantly better, at over 80% right after entry.

Several people asked which would be the safest ATM to use. A busy one with metal keys. Adam Drew wanted to try recording ATMs, to see how well this works, but wondered about the legality.

Packets in Packets: Orson Welles' In-Band Signaling Attacks for Modern Radios

Travis Goodspeed, University of Pennsylvania; Sergey Bratus, Ricky Melgares, Rebecca Shapiro, and Ryan Speers, Dartmouth College

Travis Goodspeed presented a new technique that allows an attacker to inject an arbitrary layer one packet into many wireless networks. The attack can be performed on many digital, unencrypted wireless networks where packet lengths are variable and an attacker can cause a higher-layer packet with some arbitrary data to be sent. The idea is simple: if you embed a layer-one packet (including the preamble, sync, and other metadata) in a higher-layer packet and the receiving radio does not detect the outer layer-one packet (e.g., if the sync is corrupted), that receiving radio will often detect and decode the inner layer-one packet crafted by the attacker.

Travis pointed out ways the attacker gets more of an advantage. For example, for power management reasons the outer preamble might be shortened. However, the attacker can generate a significantly longer preamble, increasing the odds that a receiver will lock onto the inner packet. In systems where the sync field is dependent on the recipient, the attack is always successful. Finally, if one receiver has better reception than the other, it's possible to target the receiver with weaker reception without the other receiver noticing.

One proposed countermeasure is to encrypt all wireless links, even if they offer no protection against local attackers. During the Q&A period, Karl Koscher pointed out that using sync patterns that can't be generated by normal data would also be an effective countermeasure. Travis responded that perhaps having different speeds, such as one and six Mbps, would allow you to inject into a network of a different frequency. Someone else asked whether the connection remains misaligned after a successful attack. Travis said no, that the attack works only on a per-packet basis.

Finally, there was a short discussion of how applicable this technique is to Ethernet. While packet corruption is extremely rare over wired Ethernet, Travis hypothesized that finding a source of noise (such as intentionally injecting collisions on an unswitched network) would allow this technique to work.

Targeting the Cloud and Commodity Computing Devices

Summarized by Mihir Nanavati (mihirn@cs.ubc.ca)

Energy Attack on Server Systems

Zhenyu Wu, Mengjun Xie, and Haining Wang, The College of William and Mary

Zhenyu Wu described an attack that forced servers to perform more expensive computations and consume more power. Having made the observation that power consumption was a large percentage of the cost of ownership, he showed that while recent advances in energy-efficient computing had led to significant decreases in the idle power consumption of servers, peak load power consumption has remained more or less unchanged from a few years ago. An attacker could significantly increase operating costs by getting the servers to constantly execute at full load.

This attack was simulated against a local MediaWiki server. Profiling the server showed that less frequent requests were absent from the object cache and took approximately six times as much power to satisfy, compared to requests that could be satisfied from the object cache. By generating such requests, they were able to force a 6–40% increase in power consumption. To achieve stealthiness, the requests were capped at a level that would make them appear to originate from a human user. The number of malicious requesters were also limited, so as not to degrade latency significantly.

Adam Drew compared the work to John Cleese's "How to Irritate People," where the attacker is just doing enough damage to be an annoyance, but not enough to severely hamper operation. He then asked whether the increased load could lead to greater failure rates for hardware. David Brumley wondered about the cost compounding, where an increase in computation increased the heat and cost of cooling required as well. Wu said that they had started exploring this direction a bit, but it was still in its early stages.

Putting Out a HIT: Crowdsourcing Malware Installs

Chris Kanich, Stephen Checkoway, and Keaton Mowery, University of California, San Diego

Chris Kanich presented this analysis of the economics of attracting and exploiting the systems of Amazon Mechanical Turk workers. Unlike normal use of Mechanical Turk to solve problems that are hard for computers to solve, this focuses on exploiting the systems of the Turkers and monetizing them in pay-per-install markets. For any monetary benefit, enough Turker systems would need to be vulnerable to outweigh the costs of actually running these tasks.

Kanich described the tasks used to determine the vulnerability of the Turkers. One asked responders questions about their antivirus, and for an additional payment to run a JavaScript program that collected information about their system and reported it back. Over 80% of the people, spread across geographical regions, were running a vulnerable configuration. Vulnerability was approximated by the existence of published CVEs—an actual available exploit was not necessary. Running another task showed that while over 90% of the respondents had antivirus, only a small fraction of them had up-to-date signatures.

Kanich then said that Amazon Mechanical Turk allows tasks to be offered on the basis of geographical regions, which is a bonus, because compromised systems can be sold for significantly varying amounts on pay-per-install programs on the basis of geographical location of the clients. Any such attack however, is only successful if there is significant uptake among users. Kanich noted that they had around 400–500 people attempt their task, most in India and in the US, within five days.

The questions revolved around whether the pay-per-install community was honest about paying the rates they advertised, and expressing general incredulity over the high percentage of vulnerable systems and the willingness of users to run untrusted code on their systems. Kanich observed that most people may believe that they are protected against any code by their antivirus. In reference to the payments, he replied that the payment rates corroborated other pay-per-install research and were within an order of magnitude of what others had observed.

All Your Droid Are Belong to Us: A Survey of Current Android Attacks

Timothy Vidas, Daniel Votipka, and Nicolas Christin, Carnegie Mellon University

Daniel Votipka presented this work, which surveys the landscape of attacks on Android smartphones and some of the possible mitigations. The smartphone market has seen huge growth, and Android, being both open source and the fastest-growing platform, with approximately 500,000 daily activations, is an important player in the ecosystem.

Android runs every application in a limited privilege sandbox, while any requests for elevated privileges have to be approved by the user. Unfortunately, the permission model requests can often be confusing to users, who are usually in a hurry to just accept and get the application running. Furthermore, even in cases where the user does try to limit permissions, it is often hard because of the generality of the request. Requests often also grant the application more power than is implied; for instance, allowing an application to receive

SMSes allows it to receive them before the standard messaging application and modify the contents accordingly.

Votipka then discussed how allowing carriers to provide updates created artificially large exploit windows, with some phones being patched a full year after the security update had originally been released by Google. Other attack vectors include the developer and debugging interface, which could be used to exploit any phone one has physical access to.

At this juncture, Votipka switched to discussing potential countermeasures, which range from shortening the exploit window, to better privilege handling by having hierarchical permissions or explicit rule checks to flag dangerous combinations of seemingly innocuous permissions. Another proposed idea was to have multiple tiers of applications in the Marketplace, with applications desiring higher permissions required to undergo verification.

Emery Burger asked if Google had been contacted and, if so, what their response was. Don said that a meeting was forthcoming. Adam Drew asked about the usability of some of these countermeasures. Dan said that while there had been several studies, it was not something they had explicitly looked into.

Exploiting the Hard-Working DWARF: Trojan and Exploit Techniques with No Native Executable Code

James Oakley and Sergey Bratus, Dartmouth College

Awarded Best Student Paper!

Sergey Bratus accepted the award, saying that James Oakley had done all the heavy lifting in this work on injecting trojan logic into binary executables using the DWARF bytecode interpreter. While DWARF is traditionally associated with debugging information, it is also used for exception handling, and every process created from a gcc-compiled binary with exception handling enabled will load the DWARF bytecode interpreter at runtime. DWARF bytecode is Turing complete and can be used as a backdoor into any such binary. This is particularly dangerous, because antivirus typically overlooks this type of trojan.

Bratus then described how DWARF bytecode, once the attacker has managed to sneak it in, can read arbitrary process memory, can defeat ASLR, can perform arbitrary computations, and is built to influence the control flow of the program. Using this, he demonstrated how a simple program could be exploited such that all code and data sections remained identical but resulted in a root shell when it threw an exception.

Bratus went on to explain the structure of several undocumented or scantily documented header frames, and how exception handling occurred for these binaries. The C++ exception handler has a "personality routine" that decides whether there is a handler at a particular level of a stack frame or whether the stack needs to be further unwound to catch the exception. Modifying the table the routine uses allows the backdoor to remain hidden until an exception is thrown and to return control to any point in the program or its loaded libraries after that.

Bratus concluded by discussing how this class of attack is currently difficult to detect but extremely powerful, because of the inherent power of DWARF bytecode; however, work is underway to mitigate it. He listed several hacker research projects, such as ElfSh/ERESI, LOCREATE, and several grsecurity/PaX-related papers in Phrack, as inspirations.

Rik Farrow asked about the root prompt displayed during the demo, and Sergey replied that they had the DWARF exploit execute a SUID shell, as the root prompt appears more interesting. He then said that there are real exploits out there, not just for C and C++ but also for some Java implementations.

Advances in Low-Level Exploitation

Summarized by Karl Koscher (supersat@cs.washington.edu)

DieHarder: Securing the Heap

Gene Novark and Emery D. Berger, University of Massachusetts Amherst

In this invited talk, Emery D. Berger revisited the DieHarder memory allocator, originally presented at CCS 2010. He began by analyzing the various ways modern heap allocators can be exploited and then described what's new in DieHarder, namely, that objects are immediately destroyed when freed, and that each object is allocated at a highly random location.

The performance of DieHarder was tested and compared to other allocators under two scenarios: the SPECint2006 benchmark suite, and Firefox loading the Alexa Top 20 Web sites from a local network cache. For SPECint2006, DieHarder was approximately 20% slower than other allocators due to it breaking TLB locality. However, for the Firefox tests, the difference between DieHarder and other allocators was not statistically significant, leading to the conclusion that DieHarder is a practical solution for Internet-facing applications such as browsers.

Rik Farrow asked about how DieHarder works, as the memory allocators he was familiar with used linked lists. Emery replied that DieHarder uses hashing and bitmaps instead of the more familiar heap allocation techniques. Mike Ryan asked about the TLB problem and Emery said that this was specifically an Intel issue. Other architectures allow software-based TLB control, and if Intel didn't fill in the entire TLB, this more flexible approach would help with DieHarder's performance. Mike then asked about how DieHarder leaves traps ("bombs" in the slide) over the entire address space in OpenBSD. Emery said that Linux does this as well, by using lots of unmapped pages that act as bombs.

Vulnerability Extrapolation: Assisted Discovery of Vulnerabilities Using Machine Learning

Fabian Yamaguchi and Felix "FX" Lindner, Recurity Labs GmbH; Konrad Rieck, Technische Universität Berlin

Fabian Yamaguchi presented this solution to a compelling problem: given a known vulnerable function, find all other functions with similar vulnerabilities. Many code bases repeat the same vulnerability mistakes, making this technique useful for finding additional vulnerabilities. The basic intuition is that functions are composed of different usage patterns, and by comparing the dominant usage patterns, you can find functions with similar vulnerabilities.

In particular, each function is represented by a sparse vector whose dimensions map to a particular type or function name. The value of each of these dimensions is simply a 1 or a 0, depending on whether that type or name is used in the function, weighted by the identifier's TFIDF, a standard weighting term used in information retrieval. Then, principal component analysis is used to find the dominant usage patterns. Functions can then be represented as a combination of these dominant usage patterns, and functions with combinations similar to a vulnerable function are likely candidates for vulnerability exploration.

As a case study, the researchers evaluated their technique on FFmpeg. They provided their system with a previously identified vulnerable decoder function and found that the most similar function (at 96% similarity) was also vulnerable. Although this second function had been patched as well, they found that the fifth-most similar function (at 72% similarity) contained the same vulnerability and was not yet patched.

Someone asked if they had only tried this one example, and Fabian said they had tried another evaluation on this data to prove that it works. Nicholas Carlini asked about the level of dimensionality, and Fabian said that they used 200 for dimensionality, which seems to work. In his thesis work, he found that using more produces more similarity.

Exposing iClass Key Diversification

Flavio D. Garcia, Gerhard de Koning Gans, and Roel Verdult, Radboud University Nijmegen

Awarded Best Paper!

Gerhard de Koning Gans presented a paper that looks at the key diversification scheme built in to the iClass contactless smart card system. The key diversification scheme was known to involve a single DES operation followed by a key fortification function. Through some amount of reverse engineering, they determined that the key fortification function is highly invertible. For a given output of the fortification function, there are an average of four possible inputs that can be easily determined. Thus, the diversification scheme offers little protection over standard DES.

The reverse engineering involved several steps, including extracting the secret Omnikey reader secure mode key and emulating an ISO 15693 card with the ISO 14443 protocol. The main technique used was to emulate cards with slightly different serial numbers and observe changes in the rekeying command sent. While they did not have a DES cracker to verify their results, they were able to use other recently published techniques to extract the master key from a legitimate reader and found that their attack did indeed find the master key.

2nd USENIX Workshop on Health Security and Privacy (HealthSec '11)

August 9, 2011
San Francisco, CA

Keynote Address

Joy Pritts, Chief Privacy Officer, Office of the National Coordinator for Health Information Technology

No report is available for this session.

Short Papers

Summarized by Ben Ransford (ransford@cs.umass.edu)

Implantable Medical Device Communication Security: Pattern vs. Signal Encryption

Fei Hu and Qi Hao, University of Alabama; Marcin Lukowiak, Rochester Institute of Technology

Fei Hu discussed his group's "cyber-physical approach" to the security of implantable medical devices (IMDs). His group has built body area networks (BANs) based on sensor motes and RFID readers. The BANs are structured as peer-to-peer networks whose trust relationships exhibit a ring structure. Hu described an assortment of attacks on IMDs and proposed preliminary defenses. He concluded with a description of "intentional signal entanglement," a mechanism by which an external device could destructively interfere with an IMD's traffic to hide private data from eavesdroppers.

Raj Rajagopalan asked to what degree intentional signal entanglement would depend on the signaling protocol the IMD uses. Hu responded that it was a physical-layer technique that would work independent of higher-level protocols.

Exposing Privacy Concerns in mHealth Data Sharing

Aarathi Prasad, Jacob Sorber, Timothy Stablein, Denise Anthony, and David Kotz, Dartmouth College—Institute for Security, Technology, and Society

Aarathi Prasad presented the preliminary results of a study on patients' privacy concerns with respect to data collected using mobile health (mHealth) devices. They conducted focus groups with patients of all ages in order to learn what people saw as the benefits and drawbacks of electronic health records (EHRs) and mHealth. In the context of a "typical" mHealth architecture, in which patients upload data to their EHRs for sharing with caregivers and family, the authors found that patients wanted the ability to turn mHealth devices on and off and to control the release of the collected data. They found that people perceived a variety of privacy risks, with diet and exercise information considered least sensitive and social interactions the most sensitive. Some patients did not understand why data such as heart rate would be considered sensitive. Patients felt more comfortable sharing data with caregivers than with their friends, families, or insurance companies.

Prasad concluded with several suggestions for mHealth architects. First, privacy controls should have access logs, and changes to privacy settings should be logged. Second, mHealth data should be annotated to aid patient understanding. Third, mHealth devices should have sensible, conservative default privacy settings, because users are unlikely to change them. Fourth, mHealth data can be presented and privacy-controlled in a hierarchical manner that matches patients' mental models.

An audience member asked whether the authors studied the effect of monetary incentives on patients' willingness to divulge data; Prasad said they had not. Raj Rajagopalan noted that privacy decisions can be context-sensitive; Prasad agreed and noted for an example that patients perceive the privacy risks of continuous versus periodic monitoring differently. Another audience member asked whether the authors told the patients how their mHealth devices worked; Prasad said they allowed patients to form their own opinions but were allowed to ask questions. Jim O'Leary asked about the effect of patient age on perceived privacy risks, considering that younger people tend to be better connected on online social networks; Prasad acknowledged that although there were clear differences between age groups, everyone in the study seemed to know about social networks.

Persistent Security, Privacy, and Governance for Healthcare Information

W. Knox Carey, Jarl Nilsson, and Steve Mitchell, Intertrust Technologies

Knox Carey pointed out that healthcare data is not flowing as easily as it should. Despite technological advances and huge investments, healthcare systems lack interoperability standards. Different organizations exhibit mismatching policies that hinder data sharing; enforcing policies across organizations is a nightmare. The current situation, Carey said, incentivizes wrong behavior such as data hoarding.

The authors propose a DRM-like approach to healthcare data, with data being encrypted at the source and persistent policies attached (and governmentally enforced). They propose associating healthcare data with sets of well-defined computations that result in different views of patient data for different interested parties, such as patients, doctors, and insurance companies.

An audience member likened the authors' proposal to a fine-grained informed-consent system, then pointed out that change in circumstances requires patient consent to be revisited; would the proposed system offer backward compatibility? Carey said it would have to. Another audience member asked how to do key management in the DRM context. Carey agreed that that was a problem and cited the additional problem of making a uniform, trustworthy DRM enforcement environment. He suggested that patients should hold their own DRM keys somehow. Carey concluded by summarizing some computations that would produce different views of healthcare data for different observers.

Who Does the Autopsy? Criminal Implications of Implantable Medical Devices

Marc Goodman, Future Crimes Institute

Marc Goodman, who has worked with the LAPD, Interpol, and FBI, offered a law-enforcement view of medical-device-related threats on the horizon. He gave examples of technology being integrated in human bodies and suggested that people might soon receive elective implantable medical devices (IMDs). This integration raises questions for forensics, such as: can medical examiners conduct forensic analysis of an IMD? The answer, given the current state of medical exams, is no, meaning that a forensic analysis might fail to discover an IMD's role in an event. He further noted the increasing use of consumer-grade computing devices in health care. He invited the audience to consider what kinds of recording and recovery mechanisms IMDs should use to alleviate the problems he mentioned.

Raj Rajagopalan noted the dearth of standards for forensic analysis of mainstream computers and asked what hope there was that the niche of IMDs would be standardized. Goodman pointed out that computer-forensic standards were beginning to appear in Europe and suggested that now was a good time to innovate. Another audience member asked whether there were standards related to default passwords on medical equipment. Goodman agreed that there ought to be standards now in order to set a precedent, since "past is prologue."

Long Papers

Summarized by Shane S. Clark (ssclark@cs.umass.edu)

A Research Roadmap for Healthcare IT Security Inspired by the PCAST Health Information Technology Report

Matthew D. Green and Aviel D. Rubin, Johns Hopkins University

Matt Green presented this work on the recent report by the President's Council of Advisors on Science and Technology (PCAST) titled "Realizing the Full Potential of Health IT," which outlines a vision for the future of electronic medical records (EMRs). Green noted that deployment of EMRs in the US is increasing but that the systems are generally not interoperable and that both sharing and security are at an early stage. He also noted that there is significant existing legislation such as the HIPAA, CCR, and HITECH acts, but that much of the legislation suffers from excessive complexity or underspecification.

Green next listed several open research areas that he feels must be addressed before a vision like that articulated in the PCAST report can be realized. The list included managing user identity, audits and logging, patient interaction with EMRs, cryptographic access controls, de-identification, and security metrics. Green contended that all of these areas require significant new work and that researchers should seek new results in each area before those outside the academic community implement poor technical solutions.

During the Q&A, an audience member asked about the problem of legacy data. Green answered that he does not believe the PCAST report addresses legacy data at all, but that it is definitely an important problem. Another audience member asked about defining security metrics and how one can claim success in solving the problem. Green said that he does not have the answer, but that an important first step is separating apparent security from actual security.

Take Two Software Updates and See Me in the Morning: The Case for Software Security Evaluations of Medical Devices

Steve Hanna, University of California Berkeley; Rolf Rolles, Unaffiliated; Andrés Molina-Markham, University of Massachusetts Amherst; Pongsin Poosankam, University of California Berkeley and Carnegie Mellon University; Kevin Fu, University of Massachusetts Amherst; Dawn Song, University of California Berkeley

Steve Hanna presented this work on software security for software-based medical devices. The researchers chose to examine the security of automated external defibrillators (AEDs) because they are widely deployed (an estimated 1.9 million worldwide in 2009) and make heavy use of software. The researchers reverse-engineered an AED's firmware, as well as the associated update and reporting programs, uncovering a variety of vulnerabilities and successfully deploying a benign worm capable of infecting the tested AED.

The first vulnerability that the researchers identified is a weak firmware verification system that allowed them to create malicious firmware for the device. The second vulnerability is a buffer overflow in the update program that leads to arbitrary code execution on the PC running the software. They also found that the AED's PC software used hard-coded plaintext passwords for data upload and stored other user credentials unprotected on the Windows host. Hanna outlined a scenario for a malicious worm using these vulnerabilities. If an attacker is able to compromise a single AED, he could use the buffer overflow in the update program to gain arbitrary code execution on the host during the next update. The compromised host could then infect other AEDs during the update process.

Finally, Hanna outlined a series of recommendations to improve the state of medical device software security. He suggested that machines used for updates be physically isolated from the network or that updates be run only within fresh virtual machines. He also suggested that device owners carefully monitor physical access to the devices. Hanna closed by saying that the researchers had notified both the FDA and the OEM of the vulnerabilities and advocated continued use of AEDs based on their life-saving potential and the low risk of compromise.

Panel

Do Medical Devices Have Significant Forensic Value?

Moderator: Ben Adida

Panelists: Kevin Fu, University of Massachusetts Amherst; Marc Goodman, FutureCrimes Institute; Nate Paul, University of Tennessee/Oak Ridge National Laboratory; Mark Day, iRhythm Technologies, Inc.

Summarized by Shrirang Mare (shrirang@cs.dartmouth.edu)

Ben Adida started by asking the panelists about their position on the topic. Kevin Fu said that software-controlled medical devices ought to be trustworthy, more particularly for forensics; otherwise, how can one tell whether a failure is accidental or malicious? Mark Day (the industry representative in the panel) made two points: first, that the industry is overwhelmed with regulations, budgets, and various other issues, and among all these issues, it is hard to have state-of-the-art security in medical devices; second, that the raw data from medical devices is very sensitive, and people don't realize that. From raw data from medical devices one can infer many things about the user. Marc Goodman said that today's medical devices are used for health alignments, but increasingly they will be used for enhancements and conveniences. As the number of medical devices increases, he thinks it will be even more important that these devices should have secure logs that will help forensics identify the root cause of a failure. Nate Paul shared his concerns and experiences with medical devices that control therapies (e.g., insulin pumps). He also thinks that it is important to add security to these devices, and ways to do forensics analysis later on, if required.

Ben Adida asked Mark Day to elaborate on what kinds of inferences one can draw from raw data. Mark Day said that from 14 days of heart rhythm data (gathered using a single channel ECG at 200 Hz sampling rate), they could identify different user activities (e.g., brushing teeth, sleeping), respiratory rate, quality of sleep, whether the user was right-handed or left-handed, and much more. An audience member asked what security measures manufacturers have in their devices. Mark Day said that people in industry try to implement what security they can (e.g., encryption, checksums), but they do not have good imaginations for future attack surfaces. The development cycle for medical devices is four to five years, and so by the time their devices are out, things have changed in the real world (i.e., new attacks emerge). He said that remotely programming a device is possible and would help a lot, but it has its own risks. He stressed the point that the people in industry are under enormous pressure—from regulatory bodies, budgets, market—and they cannot change things easily in their devices. Marc Goodman commented that he understands the pressure in the industry and thinks having dialogues between different sectors (e.g., between manufacturers and the FBI) will help manufacturers think about future attack surfaces.

David Kotz (Dartmouth College) asked how difficult it will be to implement forensic techniques in low-power sensors. Kevin Fu said the sensors have tight-resource constraints and that they don't have enough memory for additional code, but he is hopeful that in the future sensors will have more resources to work with. Mark Day pointed out that to bring ideas into reality we must bring economics into the picture. Device manufacturers already have many issues to deal with, but if we can put the security risks and benefits in terms of economics, then manufacturers will start taking security issues seriously. To a follow-up question about whether it might be too much to ask of tiny sensors, Marc Goodman said that everything does not have to be done on the sensor; a few things can be offloaded. But he thinks that because of Moore's law, sensors will have enough resources for security/forensics requirements in the near future.

An audience member raised a concern about the four- to five-year development cycle, and asked if there was any way to add security easily and quickly. Mark Day said there are many reasons why it takes so long: proprietary platforms, need to support legacy systems, long approval process, to name a few. The same person commented that we have done it for automobiles—we have added on-board diagnostic systems and, going forward, we support newer auto models. Mark Day thinks that it is not an option for medical devices. Marc Goodman said that the idea of an industry alliance coming together and forming something like an on-board diagnostic tool is great, but such a tool will also be quickly available to an attacker, increasing the attack surface.

An audience member commented that adding IT to hospitals is hard. Nurses need 50% more time to add data to devices, and it takes away from their time doing actual health care. Kevin Fu said that auditing or data logging can be automated to some extent, and he thinks that an effective and safe system does not mean that the system is going to be unusable. Ben Adida asked, if less usable might be better for patient care, does it mean that less secure might be better for patient care as well? Nate Paul pointed out that for any solution, you have to balance different factors—security, privacy, usability, and cost. Finding the right balance among these factors is the key. Mark Day thinks that people are trained for patient care, not for security, and so they do not take security seriously and they try to circumvent it whenever a system is secure but is a little hard to use.

Carl Gunter (UIUC) asked the panel for their take on regulations. For example, flights are required to have a black box as a recording device. In medical space we have the FDA deciding what the scope of regulations should be, but the regulation spectrum makes it really unclear where things stand. Marc Goodman pointed out the trend in Europe, where authorities are looking into black-box technologies in automobiles, and he thinks there is no reason not to have them in sensors in future.

Concluding the discussion, all the panelists agreed that people from different sectors need to talk to each other and get a better understanding of perspectives and problems of other groups. Nate Paul mentioned that they had some success in their talks with the FDA. He thinks physicians share the security concerns of medical devices and are interested in helping security researchers. Mark Day emphasized the need to understand real-life problems and the importance (and difficulties) of regulations.

Short Papers

Summarized by Aarathi Prasad (aarathi.prasad@dartmouth.edu)

Providing an Additional Factor for Patient Identification Based on Digital Fingerprint

Guy C. Hembroff and Xinli Wang, Michigan Technological University; Sead Muftic, KTH—Royal Institute of Technology

Guy Hembroff conducted a study which involved 13 hospitals, including critical care households at a rural setting and trauma care facilities associated with a federation. All these hospitals follow the HL7 versions for Health Information Exchange (HIE). They have seen some success with PKI. The hospital security architecture involves patients, medical staff, physicians, roaming physicians, databases, and ID management servers and certificate authorities. Sometimes test results end up with incorrect patient information. Medical staff get additional rights such as search capabilities, which they should not get. Patient-matching algorithms occasionally return duplicate results.

Given the existence of sophisticated fingerprint identification algorithms and improved biometric recognition technology, Hembroff suggests that patient identification can be based on their fingerprints, which can be indexed as a master patient identifier. This identifier becomes the biometric part of the HL7 stream, along with the patient's photo ID. A record locator service can then identify the patient based on their fingerprint and retrieve their health information, based on the security policy associated with the information. If more than one record is retrieved, the photo ID will be used to identify the correct record. Hembroff is concerned about cultural issues regarding the acceptance of fingerprints as a source of identification, and how to convince people that their digital fingerprint is secure and won't be used for other purposes.

An audience member asked about issues regarding legacy systems—what happens when biometric readers change. Hembroff answered that he knows of seven such fingerprint readers; some of them have changed since their origin, but not all of them. Another audience member asked whether it was necessary for the patient to be there every time, to which Hembroff answered that the patient needs to be there the first time her fingerprint is collected. Another question was how to deal with a patient who lost his finger. Hembroff answered that the patient would have to re-enroll in the system, and hence it is better to use multifactor biometrics.

Context-Aware Anomaly Detection for Electronic Medical Record Systems

Xiaowei Li, Yuan Xue, You Chen, and Bradley Malin, Vanderbilt University

Xiaowei Li presented an intrusion detection system for electronic medical records (EMR) using existing knowledge and traces from the clinical environment. Context information—organizational information, user role, etc.—is extracted from traces and applied to the model at runtime. In one clinical workflow, for example, you have a physician who needs to check a patient's lab test results before prescribing medications.

The workflow model he presented works in three tiers. In the first tier, a profile of the user behavior is constructed for each user or role. Next, the session is decomposed into a set of record-oriented clinical workflows. The third tier indicates the treatment guideline for the patient, which involves multiple users and roles.

An audience member asked how an anomaly is usually detected and what features are used for this detection. Li replied that normally action sets, action sequences, or other modeling techniques are used. Another member asked what would happen if the decisions in the workflow do not happen in the correct order, as in the example Li presented. Li replied that such challenges will be handled in the future with some preprocessing of the data.

Role Prediction Using Electronic Medical Record System Audits

Wen Zhang, Vanderbilt University; Carl A. Gunter, University of Illinois at Urbana-Champaign; David Liebovitz, Northwestern University; Jian Tian, Vanderbilt University; Bradley Malin, Vanderbilt University

Wen Zhang talked about role prediction, which uses audit logs to predict automatically whether a user is associated with a role. The group's work attempts to find a synergy between the two dominant strategies: role-based access control (RBAC), and experience-based access management (EBAM). They used role prediction on the EMR system deployed at Northwestern Memorial Hospital and found 8095 users, 140 roles, 143 reasons to access records, and 43 services provided at 58 locations. The role predictor accurately predicted the job titles of 51.3% (4152) of the users in the system.

For better role prediction, Zhang introduced the concept of role hierarchy. It was observed that prediction accuracy increases as you go higher up in the hierarchy. But at higher levels, the number of roles is small and thus the "separation of duty" will be violated. He also talks about the "role-up" algorithm which tries to find a balance between prediction accuracy and role number. It was found that when the algorithm was biased to accuracy and there were a small number of resulting roles, the accuracy of role prediction was 63%; when it was biased towards specificity and number of roles was high, accuracy was 52%.

One audience member asked how many beds were in the hospital. Zhang said that the study involved 8000 users, though there were not necessarily that many beds. Was "physician" considered a role? The system deployed at Northwestern is Cerner, where physician is not a role. It was also pointed out that roles and privileges are mapped from a physician's nature; when a new physician comes in, it is unclear whether a new role should be assigned.

Audit Mechanisms for Privacy Protection in Healthcare Environments

Jeremiah Blocki, Nicolas Christin, Anupam Datta, and Arunesh Sinha, Carnegie Mellon University

Anupam Datta talked about how audit mechanisms are essential for privacy protection in healthcare environments. However, the cost of inspections by a human auditor would be very high if the auditor were to inspect each access to a patient record to determine whether it was appropriate or not. Their approach, "regret minimizing audits," learns from experience to recommend budget allocations for audits in every cycle to different types of accesses. For example, if in a given audit cycle celebrity record violations caused greater loss to the organization, then the algorithm ensures that there is a higher probability that the next time the auditor performs an audit, accesses to celebrity records will be checked more. The algorithm provably converges to the best fixed strategy (e.g., budget allocation) in hindsight.

He explained that the algorithm doesn't make any assumptions about the adversary's incentives; the learning is based on the loss that is incurred during each cycle. As future work, he wants to consider alternative adversary models and audit mechanisms (which incorporate incentives), test whether experts can be identified from the logs using machine-learning techniques, and conduct experimental evaluation of the approach.

An audience member asked how it is possible to figure out who the celebrities are, to which Datta answered that their records are typically marked as celebrity records and audited separately. Another audience member asked whether logs are perfect and what would happen if all actions are not captured in the logs. Datta replied that the auditing is only as good as the information recorded in the log; he gave an example of how someone might look up information on a record and make a phone call, and not alter the data; this action would not be captured in the logs. Cory Cornelius asked whether attackers would be able to run this algorithm. Datta answered that the guarantees of the algorithm hold even when the attacker runs the algorithm.

Panel

Can We Do Meaningful De-identification of Medical Data?

Panelists: Arvind Narayanan, Stanford University; Lee Tien, Electronic Frontier Foundation; Kelly Edwards, University of Washington; Sean Nolan, Microsoft

Summarized by Aarathi Prasad (aarathi.prasad@dartmouth.edu)

Sean Nolan presented an organizational perspective on the topic. He said that it is fiction that data is anonymized and cannot be re-identified. He stated, however, that there is an increased willingness to disclose identified information to allow research to happen. The question, he pointed out, is how we can maximize people's understanding of doing it and how to maximize the value of doing it. Kelly Edwards presented an ethics perspective. She agreed that her goal was also to protect people while promoting clinical care. She said that we are caught up in the negative sense of privacy. The positive sense is that people have the right to choose and can opt in. She said that people are willing to participate at a high level, if they perceive benefits in doing so. A trustworthy system, in an ethical sense, is based on relationships and accountability. The question, she pointed out, is how to launch a public campaign to educate people about what is happening to their data.

Arvind Narayanan agreed that anonymization is pure fiction. He pointed out that understanding the data flows and who gets access to the data is very complex, so narrowing the process to a set of identifiers is not the right approach. Lee Tien's focus is on privacy, with an emphasis on health privacy. The big takeaway, according to him, was that no one knows anything about laws in health privacy, health information exchange architectures, etc. So he said it is not right to put the burden on the doctor to inform the patients where their data goes.

An audience member asked whether de-identification is the right way to go. Nolan said the question is what you are doing the research for—treatment for one person vs. analysis of 10,000. If the data is identified, you can reach back to the participants and learn more about them. Edwards said that providers are more nervous than patients, and no regulations say that identification has to be stripped from clinical studies. Narayanan replied that there are a variety of context and threat models. De-identification is useful in case of celebrity records and with an adversary who does not have technical expertise. He suggested differential privacy as a possible option instead of having fully identifiable data and de-identified data.

Another audience member pointed out that de-identification doesn't work as well as people think, especially if there is a threat from an adversary. He asked what is more important—privacy protection with de-identification or having the ability to cure AIDS if we have identifiable data? Nolan said that in the future we might have sufficient opt-in raw material to make public health claims. Edwards replied that in the US people want individual benefits and are willing to be part of something that might benefit them in the future. They are willing to contribute if we ask them. Nolan gave the example of how people donate blood because they know it is safe to do so. Narayanan argued that it was not clear to him if this could be scaled to a large population. He wondered if it was possible to provide incentives, using game theory, so that individuals could see some benefits of providing their data for research. Tien said that it is important for participants to know who is conducting the research. Sean said that a patient might not want his data to be used for research, when he is going to the doctor for treatment; the patient has to trust the system before contributing her data.

Ben Adida pointed out that hospitals were able to find correlations between patients with negative heart rates and a drug. In such cases it might be good to have identifiable data, but where do we draw the line? Tien said that when providing data for research, the patient might not know what utility there is in his data. According to Edwards, no one can decide what counts as a benefit for a diverse population; maybe a educational campaign is the solution.

Another audience member asked whether researchers could write programs, able to be run by the data holders, in such a way that the data collected could not be identified. Nolan replied that this solution, though exciting, may not work, since it is not possible to get aggregated data in all cases. In order to build this program, synthetic data is required, which is difficult to generate. There is not enough incentive for companies to adopt this solution—you will have to charge the patients to run this program—so this solution will need fundamental infrastructural changes.

Carl Gunter asked whether the panel could comment about consent bias—how to measure who opted out or opted in. Nolan said that we are still trying to comprehend consent. Edwards talked about exempt research, where it is possible to do research without requiring the participant's consent. Narayanan wondered whether we could work with self-reported data, but this data might not be useful in all cases, since the fidelity is questionable.

An audience member pointed out that biologists are required to publish their data, so that their results can be verified. Can re-identified data be used for other purposes? Edwards pointed out that biological data is usually de-identified and comes with lots of restrictions. Tien was concerned about how if some (remotely identified) data is released, people might want access to it, and access cannot be denied. Narayanan said that companies also should have a system, similar to IRBs, when conducting studies to collect data, that could audit the research.

The final question was about why data gets disclosed and about differential privacy, which, according to the audience member, has not been verified with studies other than those by the authors. Nolan pointed out that data is usually disclosed so that it can be verified. Maybe there are other ways to verify data. Narayanan said that in cases of differential privacy, anonymization comes, not from privacy protection, but from the noise that is included in the data. This has been verified in academic settings but has not been adopted anywhere.

Long Papers

Summarized by Shrirang Mare (shrirang@cs.dartmouth.edu)

Quickshear Defacing for Neuroimages

Nakeisha Schimke and John Hale, University of Tulsa

Nakeisha Schimke presented her work on de-identification method for neuroimages. The goal of this work is to sufficiently de-identify neuroimages that they cannot be linked back to an individual, and to do this task efficiently compared to existing techniques.

In neuroimages, facial features can be used to identify an individual. There are two existing de-identification methods used to remove these facial features: skull stripping, a process of segmenting brain and non-brain elements (which include facial features), and MRI defacing, a process of removing only the identifying facial features leaving the brain and surrounding tissues intact. The MRI Defacer algorithm relies on a manually labeled atlas to identify facial features. The skull stripping process is not always accurate, and it is hard to automate. The MRI Defacer process is accurate, but it requires a manually constructed atlas and is also computationally expensive. Quickshear is an effort to make the de-identification process better by making it automatic and computationally inexpensive.

The Quickshear algorithm finds a plane that divides the volume (i.e., the head in the image) into two parts: one containing the facial features and the other containing the entire brain volume. All the voxels of the "face" side are set to zero, which (apparently) is effective to de-identify the face. The key is to find the right plane such that the brain volume is kept intact. The researchers use convex hull algorithms (Andrew's monotone chain) to identify the brain mask, and once the points on the convex hull are identified, the dividing plane is drawn using the points closest to the forehead. The researchers compared their method against MRI Defacer, using 42 images from 21 subjects. They used OpenCV Face Detector to evaluate how well a method has de-identified an image. Out of the 42 de-identified images, OpenCV identified nine MRI Defacer images as faces and about 10 Quickshear images as faces. However, according to the researcher, Quickshear removes fewer brain voxels, and its running time was less than MRI Defacer; thus, Quickshear achieves nearly the same output in terms of preserving the user's privacy but is more efficient.

Arvind Narayanan (Stanford) wondered about the possibility of identifying an image based on geometry of the face; for example, distance between eyes (eye sockets are present in Quickshear images). Schimke agreed that it's a possibility but pointed out that it is hard to measure the precise distance between eyes using the eye sockets in the Quickshear images.

Adaptive Security and Privacy for mHealth Sensing

Shrirang Mare and Jacob Sorber, Institute for Security, Technology, and Society, Dartmouth College; Minho Shin, Myongji University, South Korea; Cory Cornelius and David Kotz, Institute for Security, Technology, and Society, Dartmouth College

Shrirang Mare presented his work on an adaptive protocol for mobile health sensing. People are increasingly using mobile medical sensors to measure their activity and health information, and these sensors transmit data to an aggregator device like a smartphone. Together, the sensors and the aggregator device form a body area network (BAN). BAN has similar security and privacy problems as a WiFi network. The privacy-preserving wireless protocols (proposed for WiFi networks) cannot be used for BAN because of their large overhead. The proposed adaptive method is designed to make these privacy-preserving wireless protocols efficient so that they can be used in the low-power BAN, while preserving the security and privacy properties of those protocols.

The protocol overhead is typically the header and message authentication code (MAC). The larger the overhead, the stronger the security, increasing, for example, the resistance to forgery attacks. Non-adaptive protocols use a fixed long MAC for strong security. Mare argued that a user (a user's BAN, really) is not always in a hostile environment, so always using strong security is inefficient. Instead, he suggests using a small overhead, but increasing the overhead when an adversary attacks, when the adversary is trying to forge a message. The condition on "when" to increase the overhead is critical, and he presented a probabilistic condition to identify an ongoing attack based on the number of corrupted packets (i.e., packets that fail MAC verification).

An audience member asked what happens in the case of a passive attack. Mare said the adaptive protocol does not change any parameter that would make it easier for a passive adversary to learn anything about the payload. For example, changing the MAC size does not help the adversary learn the contents of an encrypted payload. That is, the proposed method does not make the adaptive protocol any more vulnerable to passive attack than the original non-adaptive protocol.

Controlled Dissemination of Electronic Medical Records

Guido van 't Noordende, University of Amsterdam, The Netherlands

Building upon a security analysis of the Dutch electronic patient record system, Guido van 't Noordende presented his ideas on how to share electronic medical records. His approach is decentralized and keeps access to patients' information to a minimum. In this talk, he identified several paths that can be used to share information between different parties, such as patient, physician, family.

Noordende first described the traditional healthcare model: a patient visits a doctor (Doc #1). The doctor keeps all the records of the patient. When the patient visits another doctor (Doc #2), Doc #2 asks for the patient's records from Doc #1 (a pull-based model). Alternatively, Doc #1 can also send the records to Doc #2, if the patient's visit to Doc #2 is planned (a push-based model). Noordende thinks that using a controlled push-based approach with the convenience of a pull-based model is the right approach to sharing patients' records. He then presented his architecture model, outlining different paths to disclosure.

He described five different paths to disclosure for the patient's information. The idea is that the data stays in one place, but the pointer to the data is moved across different parties in a controlled fashion (i.e., controlled dissemination). The five different paths to disclosure basically describe the medium through which the pointer is shared with the doctors. The five different paths are: professional (secure) push model (e.g., emails), patient's mailbox, USB drive, smartcards, and paper (pointer writing on paper). The idea is that the patient carries the pointer to the data with him, and whoever gets the pointer from the patient gets access to the data. Thus, the patient controls the dissemination of the information.

An audience member wondered if this model can be extended to include insurance companies. Noordende said, for simplicity, he did not include insurance companies in his slides, but it is certainly possible to include them in his model. Another audience member asked how the patient can get all the active references (i.e., pointers) that are floating around. Noordende said the pointers are floating but the data is in one place. One can have access logs at that place,and can tell who has accessed your data, and also keep a log of all the pointers.

Rump Session

Summarized by Aarathi Prasad (aarathi.prasad@dartmouth.edu)

Atif Khan, University of Waterloo

Atif Khan's interests lie in patient consent and consent management. His goal is to understand what a patient wants out of the system. Can the patient choose what her data is used for, whether it is shared with her family physician or with hospitals in another state? Khan uses semantic Web technologies to define information using ontologies. The patient consent rules will be based on these ontologies.

Ben Ransford, University of Massachusetts Amherst

Ben Ransford previewed a SIGCOMM paper he coauthored. It is well known that certain medical devices are vulnerable to passive eavesdropping or the issuance of unauthorized commands. The authors' methods can protect devices that are already implanted and cannot easily be replaced. They developed an auxiliary, wearable device, called an IMD Shield, that acts as a proxy. The Shield has two antennas: one that sends a random jamming signal and another that transmits and receives data. The IMD Shield's jamming reduces the risk of private data loss and active commands by jamming transmissions to and from the IMD. The IMD Shield cancels its own jamming signal so that it is the only device that receives the bidirectional communications in the clear.

Andrés Molina-Markham, University of Massachusetts Amherst

Andrés developed a platform for medical applications, called Moo. It includes an RFID reader that provides energy to power this device, which has no battery. The microcontroller can be programmed in C. Moo has an accelerometer and temperature sensor; external sensors, storage, and harvesters can be added to the device as well.

Kevin Fu, University of Massachusetts Amherst

Kevin Fu talked about the Open Medical Device Research Laboratory, which helps researchers conduct trustworthy computing research on IMDs. MIT and Berkeley have already used IMDs from this library. A student at the University of Pennsylvania opened up the devices to understand the digital logic that goes on inside them. The devices are sterilized so that they are safe for research.

Joseph Ayo Akinyele, Johns Hopkins University

Joseph Ayo Akinyele developed a framework, called Charm, to help cryptographers who want to apply ideas to medical applications and to secure health data in the cloud, in mobile devices, etc. Implementing, measuring, and comparing crypto methods is difficult, especially since it takes a long time to write crypto code. The functional library has math libs at the lowest level and crypto schemes that focus on the algorithms at the higher level. Charm is implemented in Python. The alpha version has already been released and has been used. This version has implementations of attribute-based encryption, key policies, and ID-based encryption.

Matthew Pagano, Johns Hopkins University

Matthew Pagano's work is focused on using attribute-based encryption (ABE) to secure electronic medical records (EMRs) on mobile devices. It is difficult to get access to EMRs and other medical data during catastrophes or network outages. Access policies in healthcare can be complex, and medical systems might not have adequate security policies. With ABE, EMRs can be encrypted with expressive policies that allow the records to be exported outside the trust boundary of a medical institute. This provides self-protecting, offline access control, which is especially vital when network access is unavailable.

This solution allows patients to access their medical records and potentially store them on untrusted storage servers. In this system, the medical institute encrypts a patient's records using ABE with a suitable access-control policy.

The encrypted records are then stored on a Web server, from which patients can download their records onto their mobile devices. After receiving an ABE private key from the medical institute in an out-of-band channel, patients will be able to access their records at any time. Patients can also store their medical data with their PHR providers, either unencrypted, partially encrypted, or fully encrypted.

Mike Rushanan, Johns Hopkins University

Mike Rushanan is working on creating a trusted computing base (TCB) for mobile electronic health records (EHR). Mobile devices could have malware, and it might not be safe to build mobile health applications that can store EHR. His approach involves a Java card with attribute-based encryption (ABE), so that this card will become a trusted ABE service on the phone. The card can be installed in the phone, and it can store the patient's health data on it. They will also develop a communication protocol for the phone to interact with the card. Some processing will have to be done in the cloud, due to the resource limitations on the mobile phone. ABE can be broken up so that processing can be done away from the trusted base.

Michael LeMay, University of Illinois

Michael LeMay's research focuses on providing strong isolation for medical applications on a mobile platform. He presented the idea of a dual persona smartphone, which could be used either by the patient or the physician. However, this phone could have enterprise data or the user's personal data. It is necessary to provide clear isolation of the user's medical information on the phone. Existing software solutions have drawbacks. Protection policies are distributed and access controls are discretionary. He pointed out that errors can compromise protection if they are related to memory management and that VMMs are not enough for isolation. He also said that resource sharing could lead to vulnerabilities such as covert channels.

Raj Rajagopalan, HP

Raj Rajagopalan presented a new general notion of privacy. If you release information, you leak more information than you want. He said it is better to measure the relative release of information. He pointed out that a tradeoff should be drawn between utility (explicit disclosure) and privacy (implicit disclosure);that way you can reveal data with different levels of precision. Data exchanges involve a lot of people and sometimes time is important, so it is better if the data is not deleted. Rajagopalan wants to know whether it is possible to provide positive incentives for data holders to obey the privacy needs of individuals and whether it is possible to establish joint ownership of medical data. He also wants to know if it is possible to have a common interface between security and privacy. An audience member asked whether it is legal to sell data, to which Rajagopalan replied that there is a 4-billion-dollar industry based on selling medical data.

6th USENIX Workshop on Hot Topics in Security (HotSec '11)

August 9, 2011
San Francisco, CA

Welcome

Program Chair: Patrick McDaniel, Pennsylvania State University

Summarized by Rik Farrow (rik@usenix.org)

Patrick McDaniel, the chair of HotSec, explained how he and the PC had decided to revitalize the workshop. Their acceptance rate was 17%, and they included papers that might not otherwise be accepted—for example, for new ideas that are not yet well developed. He said that the format would be three 15-minute presentations followed by 45 minutes of discussion. Session chairs had prepared questions to help get the conversation moving, and he expected the attendees to ask their own questions as well.

New Age System Security

Summarized by Julie Ard (julieard@gmail.com)

Building Secure Robot Applications

Murph Finnicum and Samuel T. King, University of Illinois

Murph Finnicum described how increasing use of robots (Roomba, PR2) requires us to consider their unique security issues. There are many differences between robots and computers, including the fact that robots move around and have inherently probabilistic interactions. The immediate consequences of bad behavior are also much worse, although this line is becoming blurred by cyber-physical systems. For example, improper disclosure of proprietary data or loss of data can result from bad behavior directed at conventional information systems, but a robot's bad behavior could result in your house being burned down or harm to a human being.

Much of the presentation and discussion revolved around fundamental differences between robots. They include probabilistic identification, privacy, and permissions for applications. Because robots will go out into the world and interact, you cannot simply write a program identifying what they can and cannot do. The number of objects, for example, that a robot could pick up is infinite. Orders will be given by one human, and interactions would be with other humans—for example, consider a robot going to get coffee. Facial, voice, and location-based recognition do not guarantee the parameters of an operating environment but provide only probabilistic parameters.

Logging provides a necessary infrastructure for accountability, but this may violate humans' privacy. Robots will have a flawless and complete memory, and one fundamental difference to bear in mind is that unlike computers, humans don't choose where they will interact with robots. A robot could be required to notify humans when it is recording. However, can the infrastructure identify whether surrounding humans are aware that a robot is present and in operation if the robot's presence is not obvious? Perhaps humans could identify their preferences for information sharing, such as "only friends can know my location."

Finally, to carry out a task, a robot would have to take actions on behalf of the user. Permissions would have to involve high-level constructs, such as moving short distances or within a specified area. A discussion of robot behavior and morality followed, based on popular literature and movies, including Asimov's "Three Laws" and the concept of surrogates.

Security Fusion: A New Security Architecture for Resource-Constrained Environments

Suku Nair, Subil Abraham, and Omar Al Ibrahim, Southern Methodist University

Omar Al Ibrahim conveyed how the concept of "security fusion" aims to move complexity from the components to the system level in resource-constrained devices such as sensor and SCADA systems. These devices are characterized by constrained attributes such as gate count, memory, power consumption, bandwidth, physical size, and processing power. The authors propose exploring how these simple structures can lead to emergent security.

Traditional security is not possible on these devices, because the resource constraints may preclude cryptography, energy is limited, there are numerous devices deployed, nodes can be easily compromised, and oftentimes they use a wireless medium.

We were introduced to a "state machine model," which is promising and feasible for this application because resource-constrained devices are less complex than computers. Finite automata concepts will be explored in future work. Inherent in this will be a comparison of the growth of software versus hardware security complexity. The discussion resulted in a suggestion of considering what an adversary could do given a certain number of compromised nodes, in addition to the author's direction of determining how many nodes need to be compromised for an adversary to achieve a particular malicious goal.

DISTROY: Detecting Integrated Circuit Trojans with Compressive Measurements

Youngjune L. Gwon, H.T. Kung, and Dario Vlah, Harvard University

Youngjune L. Gwon began with background information on modern manufacturing methods and third-party involvement making it difficult to determine whether the received silicon is strictly what was ordered. The authors focused on power or current side-channel measurement analysis to detect trojans in integrated circuits (ICs). In particular, they explored driving the IC to a low-power state so that the trojan's power signature would be more pronounced. Their goal is to identify test vectors that will reveal anomalies indicative of trojans.

Compressive sensing is a signal processing technique for recovering data with the number of measurements proportional to the sparsity of data. However, the reduced measurements tradeoff results in an increase in false positives. One method of reducing false positives is by testing multiple chips from the same fabrication process. Additional explorations will include tradeoffs in the number of test measurements needed to reduce false positives to an acceptable level. Scalability of test vectors is a necessary factor for application of this approach. The discussion segued into supply-chain security, which is a human problem.

Privacy and Anonymity

Summarized by Ryan MacArthur (ryan.macarthur@gmail.com)

Privacy-Preserving Applications on Smartphones

Yan Huang, Peter Chapman, and David Evans, University of Virginia

Peter Chapman covered the important topic of smartphone applications that actually consider users' privacy. The phone that you carry around with you contains very personal information, be it contacts, location history, pictures, email, or banking payment records. Chapman covered an application that was built to securely "make friends" with neighboring devices, so-called "mutual contact discovery." It is known that trust is an issue, and, given evil devices, we cannot trust a device with such private data. So the common theme here is to interact with others and secure our data.

Currently this is achieved through a trusted third party such as a social media site, bank, or video game producer. The trusted third party has become the "untrusted" third party, with cases of major corporations losing massive amounts of data (e.g., Sony, Citi, Sega). To remove the third party, Chapman discusses the usefulness of the "garbled circuit protocol" proposed by Yao in the '80s. You can think of it as collective voting, implemented securely in Java. Implementation problems arise using certain immutable Java classes, and novel optimizations were developed to achieve impressive speedups. The beta version of their application is able to anonymously find common contacts with a peer, with a performance of 128 contacts in 150 seconds. Future directions are leveraging the carrier for peer discovery, software-based attestation, and lower-level (OS) support to handle secure communications.

Public vs. Publicized: Content Use Trends and Privacy Expectations

Jessica Staddon and Andrew Swerdlow, Google

For this talk, Jessica described the studies conducted on users concerning privacy expectations during their normal interactions with Internet-based services they use on a daily basis. They apparently took pains to use a diverse pool of global candidates, creating a diverse human study on current interest in privacy. There seems to be a common misconception as to where users' data actually goes and how it can be used. Staddon proposed three major categories to improve privacy expectations: transparency—in-context awareness of where data is going; control—data-use settings that users understand and can find; utility—users being given the data they need to make informed choices.

Herbert West—Deanonymizer

Mihir Nanavati, Nathan Taylor, William Aiello, and Andrew Warfield, University of British Columbia

Mihir delivered a comical talk describing efforts toward identifying authors of critical paper reviews. They collected reviews from program committees and utilized machine learning through a naive Bayes classifier utilizing NLTK in Python. They trained on unigrams, bigrams, and trigrams scored on an authorial basis using TF-IDF. The results were very interesting, as they were able to mimic the voice of PC members. It seems that simple machine classifiers are capable of identifying supposed anonymous reviews. Someone suggested that humans are good classifiers; you know whose paper it is 90% of the time. Ted Faber (USC/ISI) followed up by asking whether humans really are such good classifiers. Mihir replied that to reduce the set of possible candidates, you should use both computer and human techniques. Sandy Clarke disagreed with fingerprinting, citing Rachel Greenstadt's work at Drexel, where they found that if people disguised their own styles detection becomes impossible.

Discussions

Summarized by Rik Farrow (rik@usenix.org)

Peter Chapman was questioned about their privacy-preserving Android app. Bill Aiello (UBC) wanted to know if they had examined other work, such as fair play, as opposed to the garbled circuit implementation that they had used. Bill also wanted to know whether the authors could imagine a library of best implementations to help other developers solve these issues. Peter responded that fair play is the most famous of the garbled circuits, and he suggested checking out the Telex paper (Wustrow) that was presented on Friday. Patrick Traynor wondered whether there was a semantic difference in the results of searches performed this way. Peter said that there was not and added that their approach to performing the calculations were orders of magnitude faster. Franzi Roesner (University of Washington) considered a denial-of-service attack where the attacker would make lots of trivial changes to her address book. Peter responded that they could limit the number of times the protocol could be run with a particular partner.

Patrick Traynor asked Jessica Staddon whether we aren't already warning users of the potential for publicizing their posts or responses. Jessica replied that privacy policies are, for the most part, impenetrable. Mike Ryan (USC/ISI) pointed out that Google+ has summaries of parts of the EULA in the plainest language, such as "Google will not resell your pictures." Perry Metzger (University of Pa) said that using simply and clearly worded privacy policies is totally legal and it is just custom to word them in impenetrable legalese. John Springer of USC mentioned that Laurie Kramer at CMU has done a lot of work on copyright, a related area. Patrick McDaniel wondered about some of the results in the paper: that over half of the participants didn't realize that their posts would become public. Jessica said that a surprising number of people are not as aware as they should be. Vern Paxson commented that people don't value their private discussions and also pointed out that there is no visibility for the cost of giving away your privacy. Jessica summarized by saying we could be doing far more than we are doing now, particularly in social networks.

The paper on de-anonymizing referees' reviews generated discussion among PC members about the culture of reviewing. Ted Faber asked if the researchers had compared results from humans to results from their classifier. Mihir Nanavati said that they hadn't, as they only read those reviews where their classifier had failed in an attempt to discover why, and that algorithmic classifiers work differently from people: people tend to pick out features that algorithms ignore. Patrick Traynor thought that perhaps he should get his graduate students to write his reviews, causing Patrick McDaniel to quip, "They don't already?" Traynor responded that he isn't tenured yet.

Information Protection

Summarized by Ryan MacArthur (ryan.macarthur@gmail.com)

Towards Practical Avoidance of Information Leakage in Enterprise Networks

Jason Croft and Matthew Caesar, University of Illinois at Urbana-Champaign

Jason Croft points out that we need to differentiate between sensitive and nonsensitive data, network-wide. Problematically, protecting and configuring data against theft is challenging, as has been indicated by recent attacks on large amounts of sensitive data. Better protection is needed. Previous work (Tightlip) tends to focus on data protection at the machine level. This limits the functionality of applications that demand that data be shared, and also incurs high overhead, as each machine needs to be configured properly.

Croft presented a technique using shadow processes to compare between the original process and one where sensitive data has been scrubbed. The two processes are synchronized at system calls, where both receive the same results. When compared, if the two streams of data match, then it is safe to share. One problem they encountered is false positives relating to data that is nonsensitive. The current implementation achieves a 2x slowdown, where they hook read/write APIs to compare data. An audience member was concerned with encrypted data, but since this implementation marks data as sensitive before encryption, it is not a concern. A majority of questions hinged on the fact that managing such a system is an administrative nightmare.

Towards Client-side HTML Security Policies

Joel Weinberger, University of California, Berkeley; Adam Barth, Google; Dawn Song, University of California, Berkeley

The landscape of local HTML security offerings was detailed in this talk by Joel Weinberger. A history of attacks was given, most notably the Samy worm that wreaked havoc on MySpace. The argument was made that we need to segregate elements of content on Web pages into trusted and untrusted as a first step. The next step would be to implement a policy to deal with both types of data. Web application frameworks like RoR and Django were mentioned as proving weak amounts of policy relating to trust levels of data. It was also made clear that sanitization is hard, and we have been failing to do it properly for a while. It is for these reasons that explicit policies on how to manage both kinds of data need to be implemented. Weinberger introduced three off-the-shelf solutions—BEEP, BLUEPRINT, and Content Security Policy (CSP)— and listed the pros and cons of each. BEEP focuses on preventing XSS by whitelisting scripts. BLUEPRINT uses its own trusted JavaScript parser and the blueprint, a security policy. CSP is actually included in Firefox 4. The authors ported two applications, Bugzilla and HotCRP, to determine the impact on developers and on performance. The porting effort was substantial, because CSP does not support dynamic script generation. The performance hit was between 35% and 55%. In conclusion, Weinberger suggested that a combination of whitelisting, as found in BEEP for inline scripts, and CSP might work.

An audience member asked, “Does performance really matter?” Weinberger’s response was that performance is a big deal. Rewriting the application is part of the performance issue.

TouchLogger: Inferring Keystrokes on Touch Screen from Smartphone Motion

Liang Cai and Hao Chen, University of California, Davis

A novel proof-of-concept keylogger was presented by Hao Chen. The technique utilizes hardware devices normally thought of as safe, such as accelerometers and gyroscopes. Using a custom keyboard, Chen described how they are able to track which finger is tapping which section of the screen and were able to recreate entered text. The concept is in its early stages and was not implemented on a stock touchscreen keyboard. The hardware utilized by Chen et al. is readily accessible through JavaScript, which is a non-privileged interpreter. An audience member suggested discovering the handedness of the target, then optimizing for the detected hand. It was also clarified that the test trials had the targets sitting still with phones in hand, so any movement of the person holding the phone, such as walking or riding, was avoided.

Emerging Areas in Security

Summarized by Ryan MacArthur (ryan.macarthur@gmail.com)

On Dynamic Malware Payloads Aimed at Programmable Logic Controllers

Stephen McLaughlin, Pennsylvania State University

Stephen McLaughlin tackled the tough problem of generating a process dependency graph for logic variables, with the goal of exploiting interlocking variables in PLCs. The interlocking variables may represent safety controls, never exceeding, for example, a particular speed in a controlled device. He reviewed common systems that utilize these controllers. The Stuxnet sample was explained, as it contained a precompiled PLC payload. This indicated that the Stuxnet authors had a priori knowledge of the system they were attacking.

McLaughlin postulates that writing malware to overcome the obscurity of process control systems is an engineering problem.

He has created a system that takes binary code and translates it into an intermediate language code, and then translates that even further into Boolean expressions, all with the intent of inferring device types and interlocking variables. The goal would be to create an intelligent exploit that would determine how to manipulate key controls to wreak havoc.

Effective Digital Forensics Research Is Investigator-Centric

Robert J. Walls, Brian Neil Levine, and Marc Liberatore, University of Massachusetts Amherst; Clay Shields, Georgetown University

Walls argued that digital forensics lacks a solid scientific foundation. Without such a foundation, it becomes difficult to successfully prosecute alleged offenders. Digital forensics is inherently investigator-centric, and as such the research should be driven by the investigator, not the prosecutor. The problem we all seem to face is that forensics and the law are inseparable, yet the law is always struggling to keep up.

Investigations are about people and their actions, and intent is left out of the security domain. Walls provided simple rules, which hold close to Occam's razor, to follow for creating new policies around digital forensics. Someone made the comment that forensics show that a suspect did something with a computer, but computers do things without the owner taking action, so it is hard to prove ownership over many low-level computing functions. One open question was around the underlying issues in forensics, such as the burden of proof: how do we support the law?

Linux Magazine
ACADEMY

20% off for USENIX members

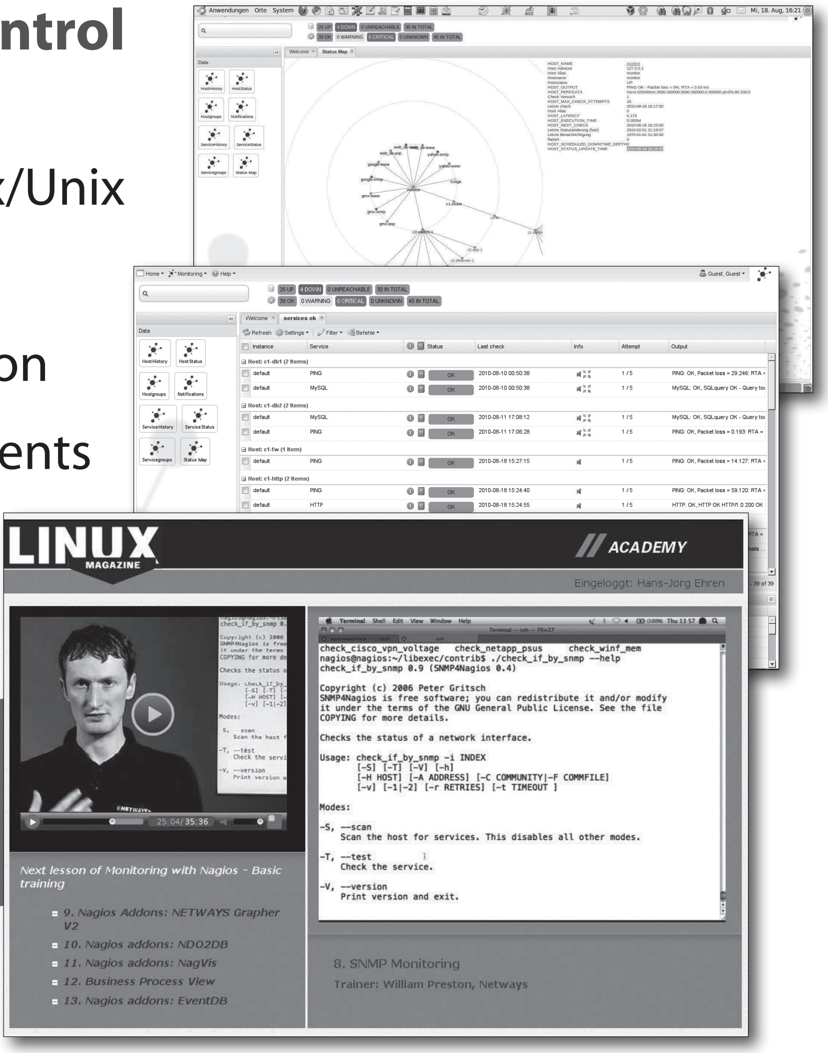
Online Training at Linux Magazine Academy

Monitoring with Nagios

Getting your IT under control the easy way:

- Monitoring Windows/Linux/Unix
- Web front-end
- Structuring the configuration
- Monitoring SNMP-components
- Nagvis, Grapher V2, and ND02DB add-ons

Many practical examples!



LINUX MAGAZINE
ACADEMY
Eingeloggt: Hans-Jörg Ehren
Terminal Shell Edit View Window Help

```
check_cisco_vpn_voltage   check_netapp_psus     check_winf_mem
nagios@nagios:~/libexec/contrib$ ./check_if_by_snmp --help
check_if_by_snmp 0.9 (SNMP4Nagios 0.4)

Copyright (c) 2006 Peter Gritsch
SNMP4Nagios is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License. See the file
COPYING for more details.

Checks the status of a network interface.

Usage: check_if_by_snmp -i INDEX
       [-S] [-T] [-V] [-h]
       [-H HOST] [-A ADDRESS] [-C COMMUNITY|-F COMMFILE]
       [-v] [-1|-2] [-r RETRIES] [-t TIMEOUT ]

Modes:

-S, --scan
    Scan the host for services. This disables all other modes.

-T, --test
    Check the service.

-V, --version
    Print version and exit.
```

Next lesson of Monitoring with Nagios - Basic training

- *9. Nagios Addons: NETWAYS Grapher V2*
- *10. Nagios addons: NDO2DB*
- *11. Nagios addons: NagVis*
- *12. Business Process View*
- *13. Nagios addons: EventDB*

8. SNMP Monitoring
Trainer: William Preston, Netways

More information:
www.academy.linux-magazine.com/usenix

usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710

POSTMASTER
Send Address Changes to *;login:*
2560 Ninth Street, Suite 215
Berkeley, CA 94710

PERIODICALS POSTAGE
PAID
AT BERKELEY, CALIFORNIA
AND ADDITIONAL OFFICES

SAVE THE DATE!

FEB. 14–17, 2012 • SAN JOSE, CA

FAST '12

10TH USENIX CONFERENCE ON FILE AND STORAGE TECHNOLOGIES

Join us in San Jose in February 2012 for the 10th USENIX Conference on File and Storage Technologies (FAST '12), as storage system researchers and practitioners come together to explore new directions in the design, implementation, evaluation, and deployment of storage systems.

Full program info and registration will be available in December 2011: www.usenix.org/fast12/login