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

Portland, OR
June 15–16, 2011

Joint ATC, WebApps, and HotCloud Keynote Address

An Agenda for Empirical Cyber Crime Research

Stefan Savage, Director of the Collaborative Center for Internet Epidemiology and Defenses (CCIED) and Associate Professor, UCSD

See the 2011 USENIX Annual Technical Conference report for this session.

Server-side Security

Summarized by Ioannis Papagiannis (ip108@doc.ic.ac.uk)

GuardRails: A Data-Centric Web Application Security Framework

Jonathan Burket, Patrick Mutchler, Michael Weaver, Muzzammil Zaveri, and David Evans, University of Virginia

Web frameworks significantly facilitate the creation of Web applications. However, they do little to facilitate the development of applications that are secure by design. In reality, lots of applications suffer from known, persistent types of vulnerabilities. Popular examples are cross-site scripting, SQL injection, and data disclosure vulnerabilities. For Rails Web applications, the root cause of such vulnerabilities is that developers have to consistently attach checks to controllers every time controllers manipulate user data. When developers forget these checks, they introduce vulnerabilities. The key idea of GuardRails is to associate data dissemination policies with the data that they protect and have the framework enforce the policies automatically.

GuardRails does source-to-source code transformations of Rails applications after they have been annotated with policies. There are two types of policies that GuardRails can enforce: access control policies and taint tracking policies. Policies can contain arbitrary code, and GuardRails supports a special syntax to define them. There are various default policies that can be used by application developers with minimal effort. For taint tracking policies, GuardRails can associate and track multiple policies per string, supporting up to character-level tainting. In order to support fine-grained sanitization according to the precise HTML context where data are used, GuardRails invokes the sanitization operations after the HTTP response has been fully generated. The authors tested GuardRails with existing vulnerable Rails applications of different sizes. Jonathan Burket reported that GuardRails prevented all vulnerabilities they tested it with. However, in terms of performance, GuardRails reduced the sustainable rate of transactions per second down to one-fourth of the rate of the original applications. Most of the overhead can be attributed to character-level taint tracking. They plan to improve performance by implementing most taint tracking routines inside the Ruby interpreter.

The Q&A mainly involved questions about how existing applications are affected by GuardRails. Jonathan responded that if the policies are written correctly, then GuardRails can avoid repeating access control operations and redundant sanitization. He also mentioned that the current prototype of GuardRails is not optimized for performance and, therefore, he sees a lot of room for improvement there.

PHP Aspis: Using Partial Taint Tracking to Protect Against Injection Attacks

Ioannis Papagiannis, Matteo Migliavacca, and Peter Pietzuch, Imperial College London

Modern Web applications rely on sanitization functions to avoid injection attacks. A sanitization function transforms user-provided strings so that the user cannot change the semantics of the operations that the Web application invokes. Developers often forget to call these functions and, by doing so, they introduce injection vulnerabilities. Past research has shown that taint tracking is effective in preventing such vulnerabilities: it can invoke sanitization functions automatically and use the taint information to sanitize the exact string characters that originate from the user. However, it is not supported by PHP.

Ioannis Papagiannis introduced PHP Aspis, a taint tracking system for existing PHP applications. PHP Aspis does taint tracking at the source-code level, by transforming the application scripts to propagate taint explicitly. This does not require support from the official interpreter or the maintenance of a custom interpreter. However, the transformations replace efficient low-level PHP operations with more computationally expensive counterparts (e.g., the concatenation operation is replaced by a function call that also propagates taint), and this reduces performance. To improve performance, the authors suggest partial taint tracking, i.e., to track taint only in the most vulnerable parts of Web applications. Ioannis supported their approach using the Wordpress example, where most past injection vulnerabilities involved plugins and not the Wordpress core. PHP Aspis separates application code in tracking and non-tracking code: the former is protected from injection attacks, but the latter is not.

In the Q&A, people wondered how PHP Aspis handles the dynamic features of PHP. Ioannis responded that PHP Aspis is also invoked at runtime to process dynamically generated code and inspect dynamic function calls. Another question concerned the separation between tracking and non-tracking code. Ioannis clarified that this separation is decided by the application's administrator and that the criteria for the separation may vary according to where injection vulnerabilities are expected to occur.

Secure Data Preservers for Web Services

Jayanthkumar Kannan, Google Inc.; Petros Maniatis, Intel Labs; Byung-Gon Chun, Yahoo! Research

Users trust services with large quantities of their sensitive data, a situation that can result in large-scale data leaks if the service gets attacked. Byung-Gon Chun suggested that the root cause of the problem is that users provide to centralized services complete access over their data. As a result, users cannot control how these services use the data. At a high level, this violates the least-privilege principle. In their paper the authors suggest the concept of Preservers, proxy objects that encapsulate user data and expose a secure API for data access. The approach targets applications that only access user data via a well-defined interface, rather than via direct raw access, but they claim that this is the norm for many real-world Web services. The client, instead of releasing his data, sets up a custom Preserver that connects to the service and implements arbitrary access control policies. Preservers may process the data they enclose and may also filter them to remove information that can identify the user. There are both stateless and stateful versions of Preservers.

Preservers may execute either in a third-party server that is trusted by both parties or be co-located with the user or the service. This flexibility avoids limitations placed by dominant service providers and enables the user to select an appropriate placement strategy according to his performance and security requirements: a trusted third party may offer better security, but co-location with the service offers reduced latency. To achieve secure co-location of Preservers with Web services, the authors suggest a Preserver implementation that relies on separate VMs for isolation. For the evaluation, the authors used three representative applications (day-trading, targeted advertising, and secure payments) and did micro-benchmarks to measure the latency overhead that the different Preserver placement options introduce. Their results show that a trusted third-party placement is the most secure but results in an order of magnitude higher latency compared to either client-side or server-side co-location.

In the Q&A, people worried about the latency of the third-party placement strategy. Byung-Gon suggested that latency can be improved by reducing the physical distance to the Preserver's host. He also said that their approach enables smaller Web sites to access user data that users would not trust otherwise. Another open question is how to find a proper interface for the Preserver, as this is not always straightforward.

Researchers' Workbench

Summarized by Veena Udayabhanu (veena@cs.umass.edu)

BenchLab: An Open Testbed for Realistic Benchmarking of Web Applications

Emmanuel Cecchet, Veena Udayabhanu, Timothy Wood, and Prashant Shenoy, University of Massachusetts Amherst

Web applications have evolved from serving just static content to dynamically generating Web pages. Modern Web 2.0 applications include JavaScript and AJAX technologies

that manage complex interactions between the client and the server. Currently, widely used benchmarks such as TPC-W and RUBiS rely on browser emulators that only mimic basic network functionality but cannot emulate other complex interactions that today's browsers possess, such as JavaScript processing. They use static load distribution. Also, the fact that most benchmarking experiments are only carried out in a LAN environment poses several questions about the accuracy of the results obtained, because the real latencies seen by geographically distributed users are not taken into account.

All these facts scream that Web applications have evolved but benchmarks have not! There are various factors that impact server performance, such as typing speed and the state size on the server. Emmanuel Cecchet presented BenchLab, an open testbed designed to address these issues.

BenchLab captures real-application workload, which is then replayed using real Web browsers, and detailed results are stored in the benchmark repository. The benchmark repository also can store virtual machines of applications under test, test traces, configurations, and results. This is useful in repeating experiments and comparing results. Capturing real traces can be done at the browser, proxy, or httpd level, depending on one's needs. Separating the generation and injection of workload is a key concept of BenchLab, and the BenchLab Webapp itself is a JEE Web application with an embedded database and repository.

In terms of the results obtained, we have seen the difference between using emulators and real browsers on server utilization. We have also seen the effects of JavaScript processing on the server workload. Finally, we have compared the effects of LAN versus WAN load injection.

One audience member asked how we deal with non-deterministic behavior of personalized Web pages such as someone's profile page on Facebook. We handle it by using HTML comparison. The repository can also contain the complete HTML responses generated by the server and we can use this to compare results between successive experiments. Another asked how they distinguish between real URLs and ones automatically generated by JavaScripts and style sheets. It is done by using the referrer field in the httpd logs and some intelligent processing.

Resource Provisioning of Web Applications in Heterogeneous Clouds

Jiang Dejun, VU University Amsterdam and Tsinghua University Beijing; Guillaume Pierre, VU University Amsterdam; Chi-Hung Chi, Tsinghua University Beijing

Emmanuel Cecchet presented this paper on behalf of Jiang Dejun, because Jiang had problems getting his US visa. In this work, the authors said that provisioning Web applications in a cloud that consists of heterogeneous machines is tough. The performance of these applications is the main concern. When the same program was run on 30 small instances of Amazon EC2 servers, they saw different performances on each of the systems.

The authors say there are two simple solutions to this problem: ignore the heterogeneous resource factors and apply current resource provisioning to make a decision, or profile each of the VM instances at each tier to make a decision (extremely time-consuming). Instead, they propose a novel resource provisioning technique consisting of five steps: (1) use a reference application for calibration; (2) correlate resource demands of reference applications and tier services on the calibration instance; (3) profile new instances with the referenced application; (4) check the performance on the new instance; and (5) apply "what-if" technique to predict the performance when a new instance is added. They showed the evaluation of their technique using TPC-W on EC2 and compared the results they got against standard techniques such as homogeneous provisioning. They concluded that profiling new instances with reference applications can be used to provision a Web application on a heterogeneous cloud platform.

In the ensuing discussion (rather than Q&A, since the authors weren't around to answer questions) the one thing most people agreed on was that checking which tier is the bottleneck is a better technique than just seeing the application performance or profiling VM instances.

C3: An Experimental, Extensible, Reconfigurable Platform for HTML-based Applications

Benjamin S. Lerner and Brian Burg, University of Washington; Herman Venter and Wolfram Schulte, Microsoft Research

Ben Lerner said that the focus of their research was on the client side of Web applications and then described what the client side of a Webapp behaves like. He mentioned the factors that make Web applications so "Webby" as follows: the code is always up-to-date; the code mainly comprises HTML, CSS, and a few JavaScripts; there is an option to view the source; and it has remixability, the ability to be combined, extended, and customized. He then explained that browser extensions are basically pieces of code that are written in

order to customize browsers dynamically at runtime; these are required in order to experience new features as closed systems but are not sufficient for researchers who want to try new things.

The thinning barrier between Web applications and browsers and the slowly evaporating role of browsers prompted the development of the C3 framework. The goal of C3, which is a reconfigurable platform for HTML-based applications, is to make experiments with extensions that facilitate research. They followed a bottom-up approach to building C3, comprising design choice, layout tree structure, language bindings, extensible parser, and overlays similar in spirit to Firefox overlays. Ben demonstrated a PowerPoint application designed as a Webapp. He also explained that, as part of the future work, they plan to add conflict detection of extensions, security monitoring, pushing the limits of HTML5, and new user interface ideas to their framework.

Emmanuel Cecchet asked whether they expect to see C3 portability on iPhone or Android. Since more and more applications are becoming like a Web application platform, they will be treated like any Web application. Another questioner pointed out that most people are terrible at making UIs that are easy to use. Given this, will having such frameworks help? Ben answered that extensions don't become popular if they break the UI.

Lessons and Experience

Summarized by Ioannis Papagiannis (ip108@doc.ic.ac.uk)

The Effectiveness of Application Permissions

Adrienne Porter Felt, Kate Greenwood, and David Wagner, University of California, Berkeley

Traditional operating systems associate permissions with users, and this leads to overprivileged applications. Instead, modern mobile operating systems such as iOS and Android use fine-grained permission systems. Each application requests a set of privileges that the user has to approve. This can happen either at runtime or at installation time. In theory, a permission system can make users aware of what the applications they install can do. Moreover, fine-grained permissions can limit the impact of vulnerabilities of benign but buggy applications. However, this assumes that applications do not request more permissions than those they really need and that the permission system's design enables useful applications using only a few permissions. So, are modern permission systems effective? This paper, presented by David Wagner, attempted to answer this question by studying existing platforms that use fine-grained permissions.

For their evaluation, the authors tested a lot of Chrome extensions and Android Market applications. They categorized the extensions and the applications according to the dangerousness of the permissions that they requested. For Chrome, only 3% of the extensions requested the ability to run native code and another 50% asked for a significant number of dangerous permissions. For Android, most applications use fewer than four dangerous permissions. However, only 9% of Chrome extensions and 10% of Android applications do not ask for any permissions, and this can result in warning fatigue to the end users. Overall, David argued that the permissions system is better than the traditional overprivileged approach, as most extensions and applications are now significantly more constrained than before.

The Q&A triggered a lively discussion. What is the correct permission granularity? They do not know, but the granularity of permissions is important as it can reduce the frequency of warning prompts. Since a lot of users may consent to any warning, can we be optimistic for the future of permission systems? Yes, because a vocal minority of users who care about security will push the developers to only ask for the permissions they really need or to justify their requirements.

Experiences on a Design Approach for Interactive Web Applications

Janne Kuuskeri, Tampere University of Technology

Current Web applications are developed with technologies more suited for traditional Web pages. Lots of Web requests waste bandwidth, as they propagate client data such as views multiple times. On the server side, application state, client state, and views are all mixed to generate the correct reply. Different, fragmented technologies such as JSPs, CSS, JavaScript, and HTML are all used for a single page. This makes good software patterns very hard to apply. Model-View-Controller (MVC) is helpful, but multiple vendors use implementations that, although similar, are sufficiently different to complicate Web development.

To facilitate Web application design, Janne Kuuskeri suggested two ideas: (1) implement the whole MVC stack in the client's Web browser with JavaScript, and (2) have the Web server expose a RESTful API that all clients will use. Web applications will be single pages that load once the necessary scripts from the server arrive and then issue AJAX queries according to the client's actions. This decouples the Web client from the Web service, allows native applications to use the same API as the Web front end, and facilitates security and error handling. With a suitable JavaScript framework, developers do not even have to worry about HTML and CSS.

The limitations of the approach are the lack of support from existing Web frameworks and the reduced ability to crawl the resulting application for search purposes. This architecture is used in Valvomo, a production system in Finland's transportation sector that Janne demoed.

Exploring the Relationship Between Web Application Development Tools and Security

Matthew Finifter and David Wagner, University of California, Berkeley

Modern Web development is characterized by an immense number of choices: programming languages, frameworks, template libraries, etc. But how should one choose? Are they all equal? The goal of this paper, presented by Matthew Finifter, was to evaluate the security of different platforms. For this, the authors used a data set from a programming contest. Nine teams of similarly experienced developers were given the same specification to create a Web application in 30 hours. Developers were free to select their own tools and languages.

Given these nine implementations (three in Java, three in PHP, and three in Perl), the authors did black-box penetration testing and manual security audits to discover vulnerabilities. Matthew reported that there is no statistically significant association between the language and the number of vulnerabilities of each implementation. The authors also studied the association between the existence of framework-provided security features and the number of vulnerabilities found for each implementation. Again, there was no significant association for XSS or SQL Injection but there was for Cross-Site Request Forgery and Session Management vulnerabilities. Overall, they report that framework-provided security features do have a measurable effect on the number of vulnerabilities, but only when the feature is fully automatic and does not require the developers to understand it and use it correctly. The authors also report that for all three languages, there is framework support to automatically prevent all types of the vulnerabilities that they identified, but this support was not always used by the developers.

Matthew was asked to estimate the total sample implementations that would have been required to generate statistically significant results. He replied that he preferred to audit applications using more samples of smaller and simpler implementations over fewer samples of more complicated ones. He also mentioned that developers should not be blamed for the increased numbers of vulnerabilities; instead, the community should focus on providing fully automatic security features for most popular Web frameworks.

Joint ATC and WebApps Invited Talk

Helping Humanity with Phones and Clouds

Matthew Faulkner, graduate student in Computer Science at Caltech, and Michael Olson, graduate student in Computer Science at Caltech

See the 2011 USENIX Annual Technical Conference report for this session.

Panel: The Future of Client-Side Web Apps

Moderator: Michael Maximilien, IBM Research Panelists: Patrick Chanezon, Google, Inc.; Charles Ying, Flipboard, Inc.; Erik Meijer, Microsoft Corp.; Raffi Krikorian, Twitter, Inc.

No report is available for this session.

Extending and Protecting the Client

Integrating Long Polling with an MVC Web Framework

Eric Stratmann, John Ousterhout, and Sameer Madan, Stanford University

Detecting Malicious Web Links and Identifying Their Attack Types

Hyunsang Choi, Korea University; Bin B. Zhu, Microsoft Research Asia; Heejo Lee, Korea University

Maverick: Providing Web Applications with Safe and Flexible Access to Local Devices

David W. Richardson and Steven D. Gribble, University of Washington

No reports are available for this session.

Joint ATC and WebApps Invited Talk

Software G Forces: The Effects of Acceleration

Kent Beck, Facebook, Inc.

See the 2011 USENIX Annual Technical Conference report for this session