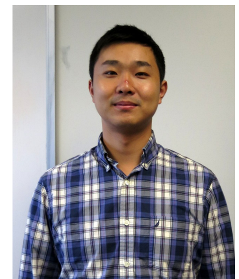
Safe and Automated Live Malware Experimentation on Public Testbeds

Abdulla Alwabel, Hao Shi,

USC/ISI


Genevieve Bartlett, Jelena Mirkovic

Aug 18, 2014: CSET14

Live Malware Analysis

- Obfuscation, encryption, downloading binaries
- Exhibits more behavior when not being watched
- Analyze malware while it runs
  - Allow access to systems
  - Allow access to the Internet
- Risk/reward trade-off

Existing Malware Analysis

- Custom, expert designed, expert used
- Mesocosms, Malwarelab (completely contained)
- GQ, BotLab (small amounts of Internet traffic allowed)

Publicly Accessible Malware Testbeds

- Requiring expertise limits the researcher pool, & stifles advancement of live malware experimentation and tools.
- Pool together resources
- Low-barrier to entry to experts in other domains

Needs

- Safe (any lab needs to be)
- Accessible (to differing levels of expertise)
- Flexible (support variety of experiments)
- Automated mechanisms (for scaling)
- Support existing tools and integration of new tools

What we bring to the table:

- Automated and Flexible Containment = scalable/safe
- Flexible tools and environments that support High-fidelity Emulation = flexible/integration

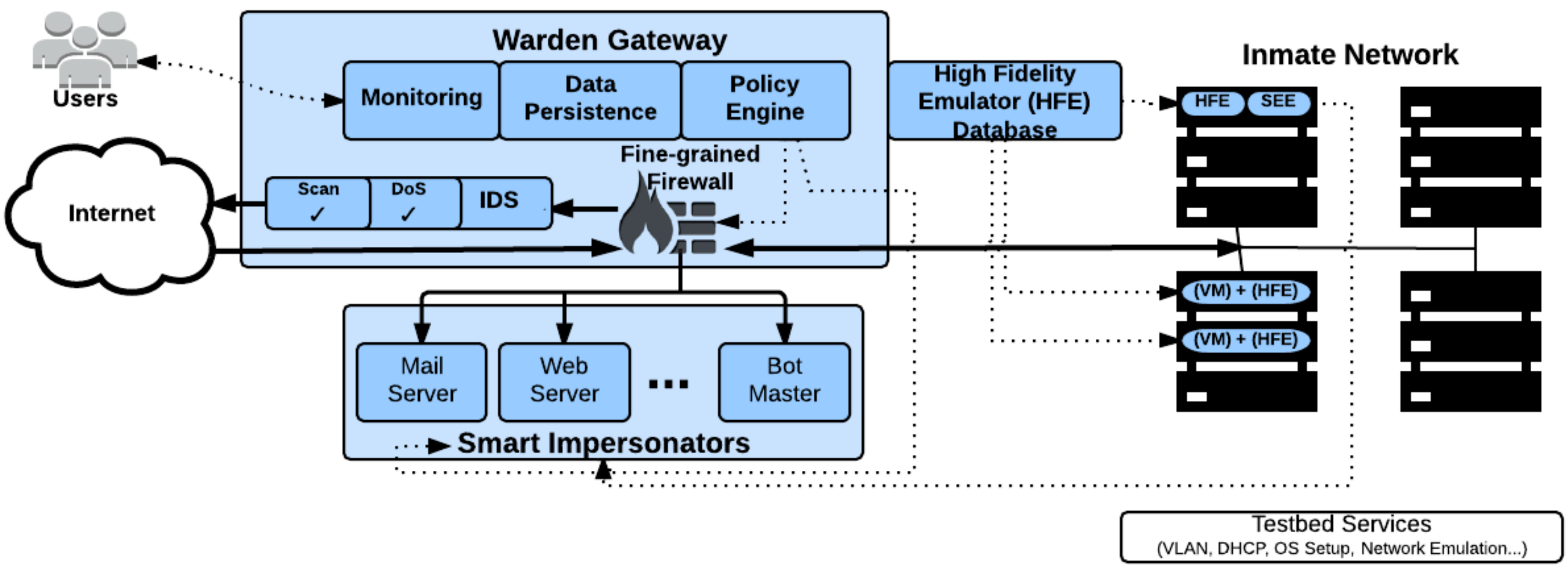
What we bring to the table:

- Automated and Flexible Containment = scalable/safe
- Flexible tools and environments that support High-fidelity Emulation = flexible/integration
- DETER and/or stand-alone setup for distribution = accessible

Outline

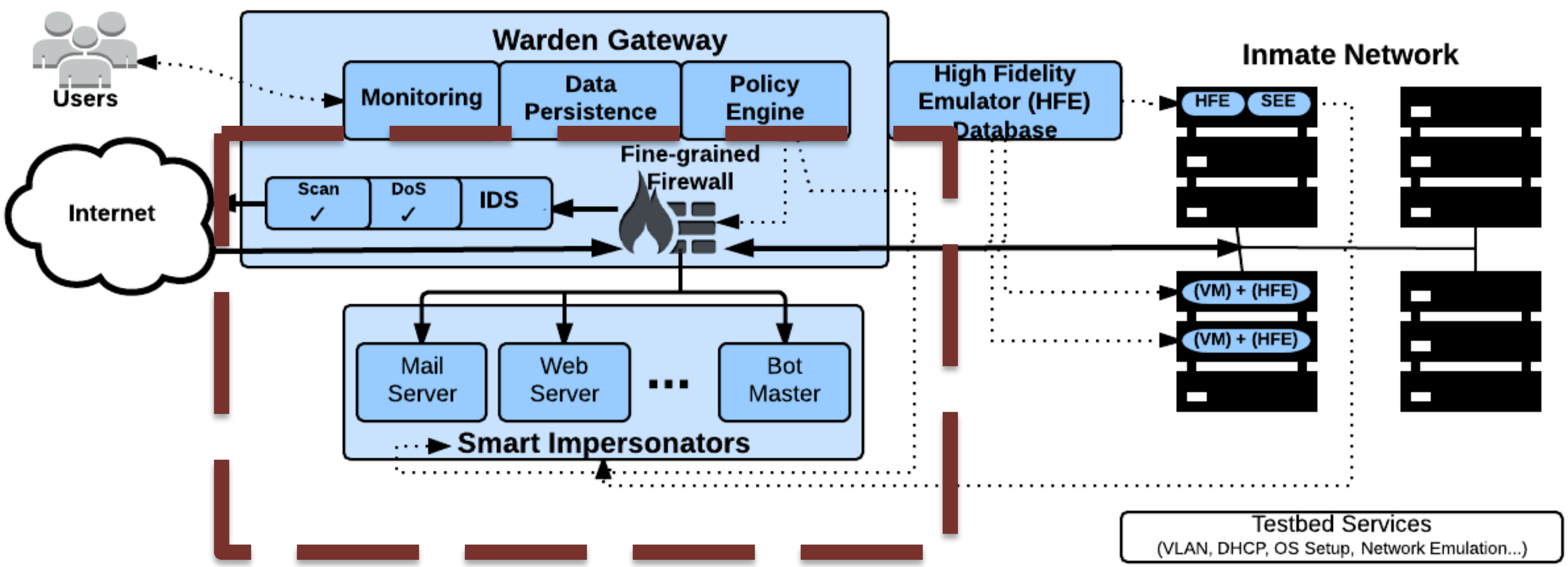
- General Architecture
- Containment
- High-fidelity Emulation

Testbed Architecture



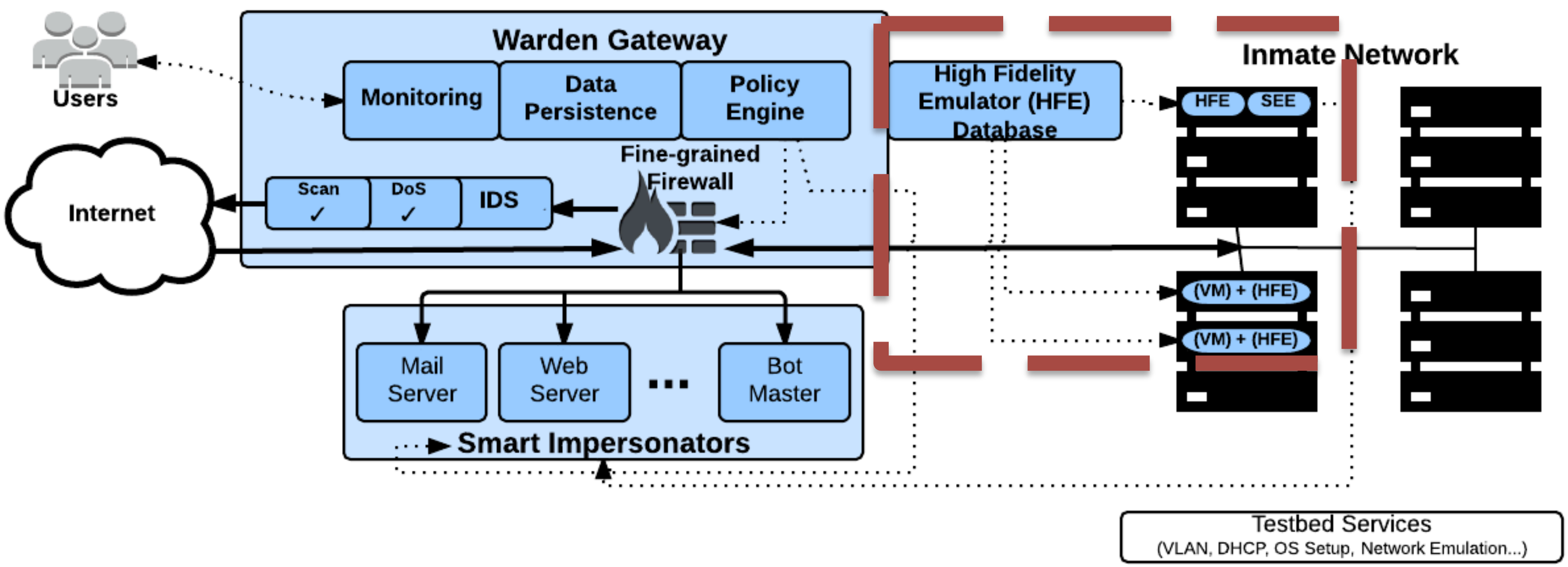
Testbed Architecture

Containment: Fine-grained firewall + Smart Impersonators



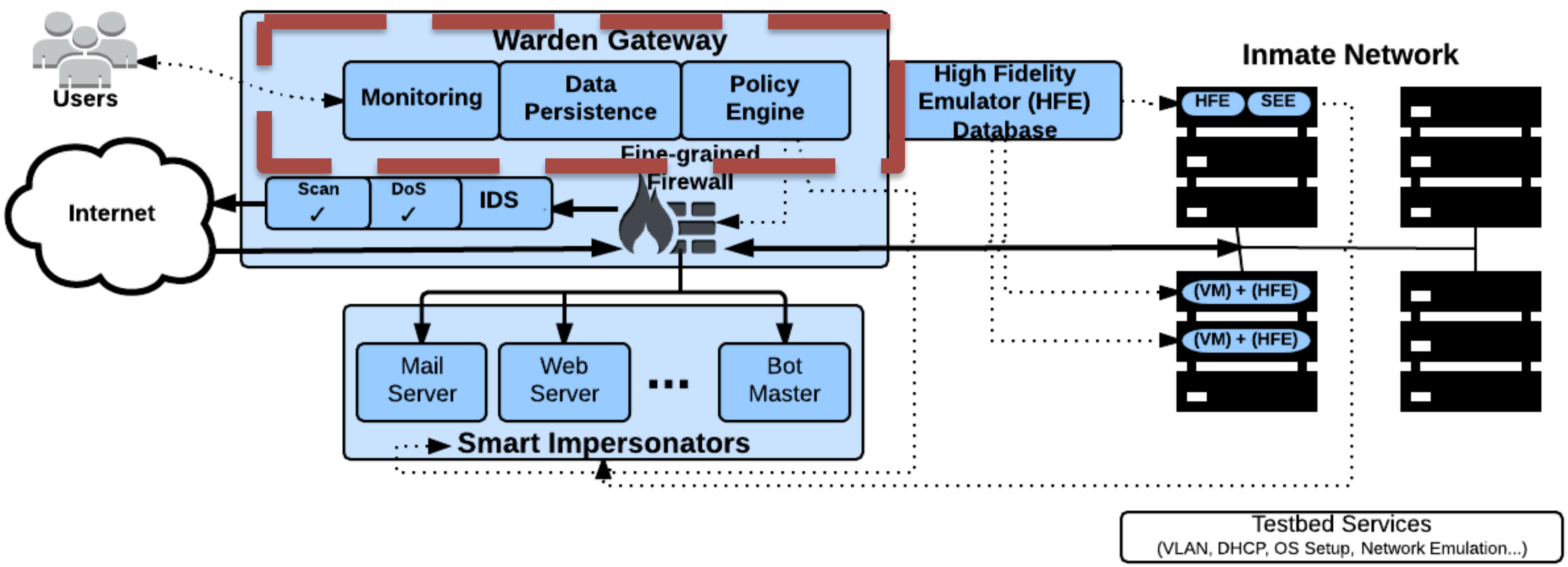
Testbed Architecture

High-Fidelity Emulation: HFE Database + VMs



Testbed Architecture

User and experiment monitoring: repeatability, customization,



Containment

- Balance risk and utility
- Support differing experimental needs
- Knowledge shared between experiments.
- Evolution of policies: from more restrictive to permissive.
- Plan: work in a cycle

Malware Management Cycle

- No guarantee that traffic is harmless.
- Four-step containment approach for communication attempt.
  - Contain and evaluate.
  - Redirect to a smart impersonator
  - Build a custom impersonator
  - Let the communication out.

Risk Management

- Let out communications:
  - Not able to build a Customer Impersonator and communication is necessary.
- But we keep an eye:
  - Monitor half-open TCP connections.
  - Ratio of of successful (reply received from server) vs unsuccessful communication attempts.
  - Detet DoS attacks by observing persistent communication attempts a destination failing to generate sufficient replies
  - Thwart spam campaigns by setting a low threshold to the number of allowed e-mail messages.
  - Block known exploits by forcing all communication to pass through one or more IDS engine

Determining if Communication is Necessary

- Measure in isolation and without isolation
- If activity is lower in isolation, communication is necessary
- Measures of malware activity:
  - Number of system calls
  - Number of unique system calls.
  - Entropy of system calls.

Smart Impersonators

- Impersonate common services: public Web, DNS and mail servers.
- Random Impersonator
  - HTTP: 200 OK, random content.
  - DNS: reply to any request with sink IP.
  - SMTP: accepts messages from any user & password
  - IRC server is a standard
- Custom Impersonator
  - Samples run in DECAF, based in QEMU:
    - Collect CPU instructions executed by malware.
    - VINE back-end to apply symbolic execution on the collected traces on the network
    - Find potential input required to execute other branches

Status of Containment

- Run 600 samples, 20 minutes each.
- 65% attempt to reach remote hosts.
- Focus on automating HTTP first

Purpose	#requests
Download binary	236
Registration	7
Contacting master	2513
Non-standard HTTP, encrypted	30
Connectivity Test	5

High-fidelity Emulation

- Need: Malware performs anti-virtulization
  - Detect VMs and change behavior from malicious to benign
- These differences called "Pills"
  - Matrix: Blue keeps you fooled, red show the truth
- Goal:
  - Enumerate all the pills
    - CPU semantic differences between a virtual machine (VM) and a physical machine (Oracle)
  - Lie to malware with the correct values (similar to kernel rootkit)

Based On

- Related Work
  - **Red Pill Testing** (EmuFuzzer, Martignoni09)
    - Generate random values for instruction parameters (user-space)
  - **KEmuFuzzer** [Martignoni10] (extend to kernel-space)
    - manually crafted test case templates for kernel instructions
    - Random values for instruction input
  - **Hi-Fi tests for Lo-Fi emulators** [Martignoni12]
    - Use symbolic execution to translate the code of a high-fidelity emulator
    - Generate test cases that can investigate all discovered code paths
    - These test cases are then used to test low-fidelity emulators

Problems with previous work

- Randomized tests
  - Cannot guarantee completeness
- Previous uses custom kernel, hard to generalize
- Comparisons are between two VMs, not a bare metal machine and a VM

Our approach: Cardinal Pill Testing

- Difference = contents of memory, values of registers & program counter
- Classify instructions into five broad categories
  - Arithmetic, data movement, logic, flow control, and misc
- Build tests off using logical coverage
- Automatically generate tests to enumerate pills

Logical Coverage & Grouping

Category	Instruction Count	Example Instructions	Parameter Coverage
arithmetic	48	`aaa`, `add`, `imul`, `shl`, `sub`	min, max, boundary values, randoms in different ranges
	336	`addpd`, `vminss`, `fmul`, `fsqrt`, `roundpd`	±infi, ±normal, ±denormal, ±0, SNaN, QNaN, QNaN floating-point indefinite, randoms
data mov	232	`cmova`, `fild`, `in`, `pushad`, `vmaskmovps`	valid/invalid address, condition flags, different input ranges
logic	64	`and`, `bound`, `cmp`, `test`, `xor`	min, max, boundary values, >, =, <, flag bits
	128	`andpd`, `vcomiss`, `pmaxsb`, `por`, `xorps`	±infi, ±normal, ±denormal, ±0, SNaN, QNaN, QNaN FP indefinite, >, =, <, flag bits
flow ctrl	64	`call`, `enter`, `jbe`, `loopne`, `rep stos`	valid/invalid destination, condition flags, privileges
misc	34	`clflush`, `cpuid`, `mwait`, `pause`, `ud2`	analyze manually and devise dedicated input

Status of High-fidelity Emulation

- QEMU (4 versions) vs. 2 oracles
  - Intel Xeon E3 3.40GHz, Windows7 Pro x86
  - Intel Xeon W3520 2.6GHz, Windows XP x86 SP3
- 19,412 generated test cases
- Achieved higher yield rate (pills/tests) than previous work (46.7% vs. 7-10%)

- Talk by Hao Shi: Cardinal Pill Testing of Virtual Machines **Thursday morning, August 21, 2014**

Summary

- Advocate for *publically* accessible malware testbed
- Containment + hi-fi flexible Emulation

- Talk by Hao Shi: Cardinal Pill Testing of Virtual Machines **Thursday morning, August 21, 2014**