WormTerminator: An Effective Containment of Unknown and Polymorphic Fast Spreading Worms

Songqing Chen, Xinyuan Wang, Lei Liu
George Mason University, VA
Xinwen Zhang
Samsung Computer Science Lab, CA
Zhao Zhang
Iowa State University, IA
Background

- Slammer infected 75,000 vulnerable hosts in about 10 minutes – Probe 4000 hosts per second on average
- Code Red I infected 360,000 servers – Double probing rate every 37 minutes

• Courtesy of CAIDA
Existing Solutions

• Signature based
  – EarlyBird [OSDI’04]
  – Autograph [Usenix Security’04]
  – Polygraph [Oakland’05]
  – Hamsa [Oakland’06]
• Traffic anomaly based
  – connection rate to unique IPs [Snort]
  – failed connection numbers [Bro]
  – failed connection rate [Usenix Security’04]
  – Honeypots based: Honeystat [RAID’04]
  – … …
Existing Solutions (con’t)

• Signature based
  – Pro: effective in detecting/containing known worms
  – Con: not efficient on unknown (zero-day) worms or polymorphic worms

• Traffic anomaly based
  – Pro: potentially detect previously unknown/polymorphic worms
  – Con: must wait till the worm has started its propagation and infected other hosts
Failure of Traffic Limitation

- Traffic Limitation
  - One new connection allowed per second/minute
  - Can delay the propagation
  - Is such delay large enough?
  - Delay normal traffic as well
    - Some normal operations become infeasible

Graph:
- Time (seconds) vs. Compromised Portion
- Curves for different values of K:
  - K=6.7 (Slammer)
  - K=1.00
  - K=0.67

Graph showing the compromised portion over time for different values of K.
What Is Desired?

– whether or not they are previously unknown or polymorphic

– without allowing any worm propagation on the Internet to infect any other host

– allowing all normal traffic

Are these possible?
Our Contributions

• Propose WormTerminator to detect the propagation of any (unknown/polymorphic) fast worm before it propagates to any other Internet host

• Implement a prototype system

• Experiment based on a real Internet worm
Outline

• Design Principles and Overview
• Design Issues & Implementation
• Evaluations
• Conclusions
WormTerminator Architecture

- **Hardware**
- **Host OS**
- **Virtual Machine Monitor**
- **Host OS Image**
- **app1 image**
- **app2 image**
- **service app1**
- **service app2**
Design Principles

• A worm always exploits the same set of vulnerabilities as coded.
  - If the host is infected by a worm, the VM must be vulnerable to the same worm.

• A fast worm always tries to propagate itself and infect others as soon as it has infected the current host.
  - If the host is infected by a worm, its propagation traffic, if diverted to the VM, thus must infect the VM.

  If the very first outgoing traffic is diverted……
Flow of Control
Outline

• Design Principles and Overview

• Design Issues & Implementation

• Evaluations

• Conclusions
Detection Criteria

- A natural criterion is to see, in a certain period of time, *if the VM traffic exhibits worm propagation pattern* after accepting the diverted traffic from the host.

- **How long** should the observation window be?
  - worm dependent
  - shorter is better
Timing Correlation

\[ I_2 = I_1 \times V M_{sd} \]
Benign Traffic

• There is **benign traffic** that may **look like worm propagation**
  – Email Relay
    • An email server receives a mail and forwards out
  
  – P2P Search
    • A peer receives a query and forwards to its neighbors
  
  – P2P Downloading
    • A BitTorrent client uploads a same file piece to multiple peers
Uniques of Benign Traffic

- **Email Relay**
  - The relay mail server is **not** the traffic destination
  - **No** processing involved except for tracing information

- **P2P Search**
  - The neighbor information is available **in advance**
  - Queries are **small**

- **P2P Downloading**
  - It is **not** unsolicited
Impact on Applications

• Application Transparency
  – Dynamically set IP address of the VM
  – Benign UDP is directly forwarded
  – The VM becomes a proxy for benign TCP

• Performance Overhead
  – Cache the examined connections
Implementation

• Diverter
  – Kernel module with ipchains/iptables

• Splitter
  – Squid 2.4STABLE1

• Detector/Controller
  – Pcap, ipchains/iptables, VMM

• Connection tracker
  – /proc
Outline

- Design Principles and Overview
- Design Issues & Implementation
- Evaluations
- Conclusions
Test Setup

• Worm: Linux/Slapper
  – OpenSSL buffer overflow in libssl
  – Apache 1.3 on RedHat, SuSe, Mandrake, Slackware, and Debian
  – More than 3500 computers were infected

• Host runs RedHat 7.3, 2.4 GHz CPU and 1GB memory
  – User-mode Linux as the VM
  – Slowdown is set to 18

• Another machine runs as the original source
Slapper Test  
-- Can WT catch Slapper?

<table>
<thead>
<tr>
<th></th>
<th>I₁</th>
<th></th>
<th>I₂</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I₁</td>
<td>I₂</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Infection</td>
<td>Code Xfer</td>
<td>Infection</td>
<td>Code Xfer</td>
</tr>
<tr>
<td>Average</td>
<td>9.3456</td>
<td>3.0654</td>
<td>91.8893</td>
<td>6.9773</td>
</tr>
<tr>
<td>Std_dev</td>
<td>0.4666</td>
<td>0.0120</td>
<td>1.2806</td>
<td>0.1103</td>
</tr>
</tbody>
</table>

*Averaged on 10 runs.*
Overhead Test Setup

- Latency: download 1 byte file
- Throughput: download a file of 100 MB
### WormTerminator Overhead

<table>
<thead>
<tr>
<th></th>
<th>Latency (ms)</th>
<th>Throughput (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>average</td>
<td>Std_dev</td>
</tr>
<tr>
<td>Direct Access</td>
<td>0.681</td>
<td>0.0697</td>
</tr>
<tr>
<td>Via WT</td>
<td>396.992</td>
<td>19.6012</td>
</tr>
<tr>
<td>Web in VM</td>
<td>4.7423</td>
<td>0.0220</td>
</tr>
<tr>
<td>Splitter Process</td>
<td>26.898</td>
<td>0.1200</td>
</tr>
</tbody>
</table>
Cache Impacts on Applications

- Lab environment, 6 clients, browser log for 4 months
- LRU replacement in cache

<table>
<thead>
<tr>
<th>Client</th>
<th>#reqs</th>
<th>#reqs (unique)</th>
<th>#cons (unique)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client1</td>
<td>8318</td>
<td>2130</td>
<td>362</td>
</tr>
<tr>
<td>Client2</td>
<td>12852</td>
<td>2724</td>
<td>455</td>
</tr>
<tr>
<td>Client3</td>
<td>8921</td>
<td>1843</td>
<td>289</td>
</tr>
<tr>
<td>Client4</td>
<td>7809</td>
<td>2074</td>
<td>337</td>
</tr>
<tr>
<td>Client5</td>
<td>24793</td>
<td>5789</td>
<td>1119</td>
</tr>
<tr>
<td>Client6</td>
<td>8457</td>
<td>2179</td>
<td>381</td>
</tr>
</tbody>
</table>
Connection Cache

![Graph showing connection cache size impact on examined client request percentage for different clients (client1 to client6).]
Outline

• Design Principles and Overview
• Design Issues & Implementation
• Evaluations
• Conclusions
Conclusions and Future Work

• We propose WormTerminator to detect and contain all unknown/polymorphic worms without infecting any other hosts

• We implemented and experimented on a real worm to demonstrate its feasibility

• We need to further improve its performance
  – Better virtual machine
  – Multi-core processor
Thanks
&
Questions?