Hershel: Single-Packet OS Fingerprinting

Zain Shamsi, Ankur Nandwani, Derek Leonard, and Dmitri Loguinov

Internet Research Lab
Department of Computer Science and Engineering
Texas A&M University

June 18, 2014
Agenda

• Introduction
• Background
• Building Hershel
• Simulations
• Internet Scan
Introduction

• The goal of **OS fingerprinting** is to determine OS of a remote host based on its network behavior

• Stack differentiation is possible due to:
  – Unclear language and lack of response standardization in IETF RFCs
  – No mandated behavior for malformed requests
  – Broken (non-compliant) implementations

• Network administrators and industry analysts have used OS fingerprinting as a tool
  – Identify and secure devices in own network
  – Market analysis of OS usage
Introduction

• Internet measurement studies are important to researchers
  – Detect vulnerabilities
  – Show deployment of new software and protocols

• Scans have become progressively faster
  – 30 days, 1K pps [Heidemann 2008]
  – 24 hours, 24K pps [Leonard 2010]
  – 45 minutes, 1.4M pps [Durumeric 2013]

• Large-scale measurement tools need to be fast, low overhead, and accurate
  – OS fingerprinting at large scale has not been explored before, which is our topic here
Agenda

• Introduction
• Background
• Building Hershel
• Simulations
• Internet Scan
Background

• Active OS fingerprinting typically requires open port

• Rooted in banner grabbing, which has many drawbacks
  - Protocol must be known
  - High overhead
  - Defeated by generic software (e.g., Apache)
  - Admins can also remove/obfuscate OS-identifying strings

• Nmap is the current state of the art
  - Database of over 4K different OSes
  - Default 1032 probes per target, but no less than 38 in the least-verbose mode
Background

• Why not use Nmap?
  - Not a polite tool, generates complaints
  - Sends malformed probes, performs vertical port scans
  - Slow, infeasible for large scale
  - Packets easily blocked by IDS such as snort

• Therefore, a more subtle approach is needed
  - p0f, RING, Snacktime are single-packet tools
  - Use header fields and timing of SYN-ACKs
  - Have small OS fingerprint databases (~20 different stacks)
  - Inaccurate when features change (e.g., packet loss)

• As a result, the issue of low-overhead and accurate fingerprinting remains open
Agenda

• Introduction
• Background
• Building Hershel
• Simulations
• Internet Scan
Building Hershel

• Our aim is to build a single-packet tool that is robust to network and user modification
  – “Single-packet” means one outbound probe, but multiple responses from the remote OS are allowed

• Assume remote host responds to TCP SYN
  – Specific port/protocol does not matter
  – A SYN probe provides minimal intrusiveness, along with non-malicious operation

• Suppose each OS $j$ can be described by some fingerprint vector $y_j$
  – Consists of two types of features – network and user
Building Hershel

- Network features are SYN-ACK RTOs

- Examples:

<table>
<thead>
<tr>
<th>OS</th>
<th>SYN-ACK RTO</th>
<th>Reset RTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 7</td>
<td>3 6</td>
<td>12</td>
</tr>
<tr>
<td>Mac OSX 10.3</td>
<td>2.92 6 12 24</td>
<td>30</td>
</tr>
<tr>
<td>NetBSD 4.0</td>
<td>2.92 6 12 24</td>
<td>-</td>
</tr>
<tr>
<td>Juniper Netscreen</td>
<td>1.67 2 2 2 2 2 2 2 2 2</td>
<td>2</td>
</tr>
<tr>
<td>Huawei Embedded</td>
<td>0.7 1 1.2 3 4 5</td>
<td>-</td>
</tr>
</tbody>
</table>
## Building Hershel

- **User features** are values taken from packet header fields

<table>
<thead>
<tr>
<th>OS</th>
<th>Win</th>
<th>TTL</th>
<th>DF</th>
<th>OPT</th>
<th>MSS</th>
<th>RST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 7</td>
<td>8192</td>
<td>128</td>
<td>1</td>
<td>MNWST</td>
<td>1460</td>
<td>1,0,1,0</td>
</tr>
<tr>
<td>Mac OSX 10.3</td>
<td>33304</td>
<td>64</td>
<td>1</td>
<td>MNWNNT</td>
<td>1460</td>
<td>1,1,1,32768</td>
</tr>
<tr>
<td>NetBSD 4.0</td>
<td>32768</td>
<td>64</td>
<td>1</td>
<td>MNWNNTSNN</td>
<td>1460</td>
<td>0,-,-,-</td>
</tr>
<tr>
<td>Juniper Netscreen</td>
<td>8192</td>
<td>64</td>
<td>0</td>
<td>M</td>
<td>1380</td>
<td>1,0,0,8192</td>
</tr>
<tr>
<td>Huawei Embedded</td>
<td>1536</td>
<td>255</td>
<td>0</td>
<td>M</td>
<td>768</td>
<td>0,-,-,-</td>
</tr>
</tbody>
</table>

- **M** = MSS, **N** = NOP, **W** = Window Scale, **S** = Selective ACK, **T** = Timestamp
Building Hershel

- Challenges
  - One-way delay (OWD) jitter (usually zero-mean)
  - Packet loss

<table>
<thead>
<tr>
<th>With OWD</th>
<th>1 packet lost</th>
<th>2 packets lost</th>
<th>3 packets lost</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2.8, 6.4, 12.1)</td>
<td>(9.2, 12.1)</td>
<td>(21.3)</td>
<td>empty</td>
</tr>
<tr>
<td>(2.8, 18.5)</td>
<td>(6.4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2.8, 6.4)</td>
<td>(18.5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(6.4, 12.1)</td>
<td>(9.2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(6.4, 12.1)</td>
<td>(12.1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2.8)</td>
<td>(2.8)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Not just many possibilities, but also drastically different values!
## Building Hershel

- **Challenges (cont’d)**
  - User modification of default TCP/IP parameters (e.g., OS tuning software, fingerprint scrubbers, NAT, IDS)
  - Unlike OWD, these result in arbitrary value fluctuations
  - **Example**: Window size is more likely to jump from 8,192 to 65,535 than to 8,193

- Treating all features as volatile, an observed sample can match pretty much any OS

<table>
<thead>
<tr>
<th>Fingerprint</th>
<th>Win</th>
<th>TTL</th>
<th>DF</th>
<th>OPT</th>
<th>MSS</th>
<th>RST</th>
<th>RTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observed</td>
<td>65535</td>
<td>64</td>
<td>1</td>
<td>MNW</td>
<td>1460</td>
<td>1,1,0,0</td>
<td>2.8</td>
</tr>
<tr>
<td>Windows 7</td>
<td>8192</td>
<td>128</td>
<td>1</td>
<td>MNWST</td>
<td>1460</td>
<td>1,0,1,0</td>
<td>3</td>
</tr>
<tr>
<td>Mac OSX</td>
<td>33304</td>
<td>64</td>
<td>1</td>
<td>MNWNNT</td>
<td>1460</td>
<td>1,1,1,32768</td>
<td>2.9</td>
</tr>
</tbody>
</table>
Building Hershel

• Thus, any observation $x$ can be viewed as a distortion of each original fingerprint $y_j$ from underlying OS $j$.

• Given a sample $x$, our goal is to determine the most probable $y_j$ that could have produced it:

$$ s(x) = \arg \max p(y_j | x) $$

• Which is equivalent to:

$$ s(x) = \arg \max p(x | y_j) p(y_j) $$

- probability that $y_j$ became distorted into $x$
- probability that observation $x$ comes from OS $j$
- fraction of hosts running OS $j$
Building Hershel

- To obtain these probabilities, we need a new model
  - Machine learning techniques don’t work due to lossy features

- We develop a stochastic theory of single-packet fingerprinting to account for these random effects
  - See paper for details

- We then build a classifier called Hershel, which can additionally handle OSes with random feature vectors, and construct a database of 116 OSes

- Can distinguish not only between OS families (Windows, Linux, FreeBSD, embedded devices), but also patch levels (SP1 vs SP2)
Agenda

• Introduction
• Background
• Building Hershel
• Simulations
• Internet Scan
Simulations

- Emulate a FIFO queue between server and client
  - Run simulations to classify $2^{18}$ IP samples with random network/user modifications
  - Vary packet loss and user feature modification from 0 to 50%

- First, we perform comparison with Snacktime, which is the most accurate previous single-packet tool
  - Uses only RTO and Win/TTL (Pareto OWD, mean 0.5 sec)

<table>
<thead>
<tr>
<th>Loss</th>
<th>Feature mod</th>
<th>Snacktime</th>
<th>Hershel</th>
<th>Snacktime</th>
<th>Hershel</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>0%</td>
<td>12%</td>
<td>22%</td>
<td>58%</td>
<td>86%</td>
</tr>
<tr>
<td>3.8%</td>
<td>10%</td>
<td>10%</td>
<td>21%</td>
<td>44%</td>
<td>78%</td>
</tr>
<tr>
<td>10%</td>
<td>10%</td>
<td>7%</td>
<td>20%</td>
<td>33%</td>
<td>76%</td>
</tr>
<tr>
<td>50%</td>
<td>50%</td>
<td>0.8%</td>
<td>10%</td>
<td>2%</td>
<td>28%</td>
</tr>
</tbody>
</table>
Simulations

- Hershel’s RTO classifier doubles Snacktime accuracy at low loss, triples at 10%, and improves an order of magnitude at 50% loss
  - However, Hershel works even better with new features

<table>
<thead>
<tr>
<th>Loss</th>
<th>Feature mod</th>
<th>RTO Only</th>
<th>+Win/TTL</th>
<th>+DF</th>
<th>+TCP OPT</th>
<th>+MSS</th>
<th>+RST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>0%</td>
<td>22%</td>
<td>86%</td>
<td>89%</td>
<td>96%</td>
<td>99%</td>
<td>99.9%</td>
</tr>
<tr>
<td>3.8%</td>
<td>10%</td>
<td>21%</td>
<td>77%</td>
<td>79%</td>
<td>91%</td>
<td>94%</td>
<td>95%</td>
</tr>
<tr>
<td>10%</td>
<td>10%</td>
<td>20%</td>
<td>76%</td>
<td>77%</td>
<td>91%</td>
<td>94%</td>
<td>95%</td>
</tr>
<tr>
<td>50%</td>
<td>50%</td>
<td>10%</td>
<td>28%</td>
<td>35%</td>
<td>54%</td>
<td>57%</td>
<td>60%</td>
</tr>
</tbody>
</table>

- Numerous other scenarios and delay distributions omitted here, but shown in the paper
Agenda

• Introduction
• Background
• Building Hershel
• Simulations
• Internet Scan
Internet Scan

- Port-80 SYN scan of the Internet
  - 2.1B IPs in 24 hours, 37.8M responses, 94% with at least one RTO

- Extensive sanity verification of the dataset
  - Not enough room to show here, see the paper

- We see a lot more values for each header field than we have in our dataset
  - Emphasizes the importance of probabilistic matching

- Run Hershel on all hosts and obtain a non-zero matching probability on 37.4M devices

<table>
<thead>
<tr>
<th>RTOs</th>
<th>Hosts</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9.6M</td>
<td>27</td>
</tr>
<tr>
<td>2</td>
<td>9.0M</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>7.8M</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>5.0M</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>2.6M</td>
<td>1</td>
</tr>
</tbody>
</table>
**Internet Scan**

- Classification results – top 5 OSes and families

<table>
<thead>
<tr>
<th>OS</th>
<th>Hosts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux 2.6 / 2.4</td>
<td>9.6 M</td>
</tr>
<tr>
<td>VxWorks Embedded</td>
<td>4.1 M</td>
</tr>
<tr>
<td>Windows Server 2003 SP1 SP2</td>
<td>2.3 M</td>
</tr>
<tr>
<td>VxWorks 5.4 / Xerox Embedded</td>
<td>1.8 M</td>
</tr>
<tr>
<td>Linux 2.6 / Debian / CentOS</td>
<td>1.1 M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Family</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>13.8 M</td>
</tr>
<tr>
<td>Embedded</td>
<td>13.5 M</td>
</tr>
<tr>
<td>Windows</td>
<td>7.5 M</td>
</tr>
<tr>
<td>Other (Mac, BSD, Novell, etc)</td>
<td>2.3 M</td>
</tr>
</tbody>
</table>

- Compared to previous application of Snacktime to this dataset [Leonard10], 9M more embedded devices

- Manual verification vs. Snacktime
  - We pick 1000 random hosts to compare classifications
  - When Hershel and Snacktime disagree, 97% of the time Hershel is correct, 1.8% Snacktime, and 1.2% neither
Thank you!

Questions?