Intrusion Detection and Malware Analysis

Malware collection

Pavel Laskov
Wilhelm Schickard Institute for Computer Science
Motivation for malware collection

- Understanding vulnerabilities and attack techniques
- Development of protection and neutralization tools
- Understanding the attacker communities and their “business models”.
Malware collection tools

- **Honeypot**: an isolated, unprotected and monitored system, containing seemingly valuable for attacker resources, aimed at collecting examples of malicious activity.
- **Honeyclient**: an automated client-side vulnerable system executed in a controlled environment.
- **Honeynet**: a distributed collection of honeypots and email filters intended for a large-scale collection and observation of malware.
Honeypot taxonomy

- **Low-interaction**: simple daemons simulating network services; no exploitation.
- **Medium-interaction**: emulated vulnerabilities for attracting and executing malware in a controlled environment.
- **High-interaction**: real systems communicating with malware in a controlled environment.
Low-interaction honeypots

(+) Low security risks due to emulation
(+) Simple installation and recovery
(+) Suitable for analysis of automatic attacks
(+) High scalability
(−) Not suitable for detection of interactive attacks due to limited emulated functionality
(−) Hardly suitable for acquisition of malware binaries
High-interaction honeypots

(+ ) Suitable for detection and acquisition of any malware kinds
(− ) Time and resource consuming installation and maintenance
(− ) High security risks: additional security mechanisms are required
(− ) Virtualization can be detected by malware
Medium-interaction honeypots

(+) A relatively wide exploit coverage
(+) Extensive monitoring and collection functionality
(+) Full virtualization not necessary
(+) Relative ease of deployment and maintenance
(+) Low to moderate security risks (egress outbreak)
(−) Manual emulation of vulnerabilities still necessary
(−) Detection of novel exploits not always reliable
A honeypot example: Nepenthes

- **Vulnerability modules**: emulate vulnerable parts of network services.
- **Shellcode parsers**: analyse shellcode to locate its source.
- **Fetch modules**: download binaries from remote locations.
- **Submission modules**: store binaries in a specified location.
Nepenthes vulnerability modules

Poor man’s implementation of the original vulnerability

- Send $N$ fixed strings, random junks, exploit “stages”
- Dismiss intermediate received stages
- Record final stage and use in payload

Example:

```c
ConsumeLevel LSASSDialogue::incomingData(Message *msg) {
    m_buffer->add(msg->getMsg(), msg->getSize());

    char reply[512];
    for (int32_t i = 0; i < 512; i++) {
        reply[i] = rand() % 256;
    }
}
```
Nepenthes shellcode analysis

- Analyze the incoming payload and extract malware location
- shellcode-signature module
  - Signature-controlled shellcode analyzer
  - Perl-compatible RE patterns for commonly seen shellcode
  - Identify parameters of shellcode (ports, URIs, …)
- Shell emulator with arbitrary commands
Download the actual malware from previously generated URL

Several modules for various protocol:
- HTTP(S), (T)FTP, RCP, ...
- “Proprietary” malware protocols:
  - CSend and CReceive from AgoBot
  - LinkBind and LinkConnectback from linkbot

RFC-incompliant implementations of HTTP and FTP
Objective: detection of attacks directed at client-side software, mostly web browsers:
- browser exploits
- “drive-by downloads”
- typo-squatting

Applications:
- security analysis of web sites
- finding malicious content distribution sites
- detection of new browser exploits
- malware collection
Browser exploitation via redirection

1. Obfuscated Java Scripts

2. Third-Party Redirection

3. Malicious Scripts
   Attempting to exploit
   Multiple vulnerabilities

4. Malware Installation

Source: Yi-Min Wang, Microsoft Research
Honeyclient example: HoneyMonkey

- A VM-based high interaction honeyclient, running a vulnerable browser.
- Automatic detection of redirection relationships between content distribution sites
- Detection of zero-day attacks
HoneyMonkey architecture

Stage 1: \( N \) URLs per VM, unpatched WinXP, no redirection analysis

“Interesting” URLs

Stage 2: 1 URL per VM, unpatched WinXP SP2, redirection analysis

Exploit URL topology graph

Exploit URLs

Analysis of exploit URL density

Stage 3: 1 URL per VM, patched WinXP SP2, redirection analysis

Zero-day exploits

Access blocking

Vulnerability patching

Interesting URLs
HoneyMonkey deployment results

Results were obtained in May-June 2005 on a list of 16,190 URLs with known bad content (pornography, adware distribution, some shopping and freeware screensaver sites).

<table>
<thead>
<tr>
<th>HoneyMonkey configuration</th>
<th>Exploit num./freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage 1, fully unpatched</td>
<td>207 (1.3%)</td>
</tr>
<tr>
<td>Stage 2, fully unpatched (SP1)</td>
<td>688 (4.2%)</td>
</tr>
<tr>
<td>Stage 2, fully unpatched (SP2)</td>
<td>204 (1.3%)</td>
</tr>
<tr>
<td>Stage 3, SP2 partially patched</td>
<td>17 (0.1%)</td>
</tr>
<tr>
<td>Stage 3, SP2 fully patched</td>
<td>0 (0%)</td>
</tr>
</tbody>
</table>

In July 2005, 27 URLs were discovered that distributed a zero-day exploit.
Goals:
- Wide coverage of up-to-date “malware landscape”
- Fast discovery new malware strains

Challenges:
- Maintenance: deployment by less qualified administrators
- Security: avoid potential infection of host systems
- Automation: adjust to potentially unknown vulnerabilities
- Scalability: infrastructure for storing massive amounts of malware
- Utility: interface for analysis tools
- Stealth: should not be detectable by malware
Message extraction from TCP flows
- Generation and refinement of a finite state machine model for a communication protocol used by malware
- Generation of a honeyd-compatible script for implementation of a finite-state machine.
- Communication interface for interaction with the repository and analysis components.
What do we want to know about malware?

- Is it recognized by existing antivirus products?
- What is its functionality?
  - How is malware distributed?
  - What other harmful functions does malware carry out?
- What are relationships between various classes of malware?
  Do they share common techniques? How do they evolve?
**How much of malware is unknown?**

- **Experiment:**
  - Current instances of malware were collected from a Nepenthes honeypot.
  - Files were scanned with Avira AntiVir.

- **Results:**
  
  **First scan:**

  - **Detected:** 76%
  - **Undetected:** 24%

  After four weeks, 15% of malware instances were still not recognized!
How much of malware is unknown?

- **Experiment:**
  - Current instances of malware were collected from a Nepenthes honeypot.
  - Files were scanned with Avira AntiVir.

- **Results:**

  **First scan:**
  - Undetected: 24%
  - Detected: 76%

  **Second scan:**
  - Undetected: 15%
  - Detected: 85%

After four weeks 15% of malware instances were still not recognized!
How much of malware is unknown?

Experiment:
- Current instances of malware were collected from a Nepenthes honeypot.
- Files were scanned with Avira AntiVir.

Results:

First scan: detected 76% undetected 24%

Second scan: detected 85% undetected 15%

After four weeks 15% of malware instances were still not recognized!
CWSandbox: system architecture

- During the initialization of a malware binary cwmonitor.dll is injected into its memory to carry out API hooking.
- DLL intercepts all API calls and reports them to CWSandbox.
- The same procedure is repeated for any child or infected process.
Goals:
- Detect variations of known malware families.
- Detect previously unknown families.
- Determine essential features of specific families.

Main ideas:
- Use AV scanners to assign labels to malware binaries.
- Use machine learning to classify binaries among known malware families.
Malware classification: data acquisition

- Binaries are collected from Nepenthes and from spam-traps, and are analyzed by CWSandbox.
- Labels are assigned by running *Avira AntiVir*. Unrecognized binaries are discarded.
- 14 classes are considered for training: 1 backdoor, 2 trojans and 11 worms.
Malware classification: feature extraction

- **Operational features:** a set of all strings contained between delimiters “<” and “>”.
- **“Wildcarding”:** removal of potentially random attributes.

```
<copy_file filetype="File" srcfile="c:\1ae8b19ecea1b65705595b245f2971ee.exe" dstfile="C:\WINDOWS\system32\urdvxc.exe"
 creationdistribution="CREATE_ALWAYS" desiredaccess="FILE_ANY_ACCESS"
 flags="SECURITY_ANONYMOUS" fileinformationclass="FileBasicInformation"/>
<set_value key="HKEY_CLASSES_ROOT\CLSID\{3534943...2312F5C0&}" data="lsslwhxtetntbkr"/>
<create_process commandline="C:\WINDOWS\system32\urdvxc.exe /start"
 targetpid="1396" showwindow="SW_HIDE"
 apifunction="CreateProcessA" successful="1"/>
<create_mutex name="GhostBOT0.58b" owned="i"/>
<connection transportprotocol="TCP" remoteaddr="XXX.XXX.XXX.XXX"
 remoteport="27555" protocol="IRC" connectionestablished="1" socket="1780"/>
<irc_data username="XP-2398" hostname="XP-2398" servername="0"
 realname="ADMINISTRATOR" password="r0flc0mz" nick="[P33-DEU-51371]"/>```
Malware classification: training

- For each known malware family, train a Support Vector Machine for separating this family for the others:

\[
\min_{w,b} \frac{1}{2}||w||^2 + C \sum_{i=0}^{M} \xi_i \\
\text{s.t. } y_i((w \cdot x_i) + b) \geq 1 - \xi_i, \ i = 1, \ldots, M. \\
\xi_i \geq 0
\]

- Determine the optimal parameter \( C \) by cross-validation
Classification of unknown malware binaries

- **Maximum distance**: a label is assigned to a new report based on the highest score among the 14 classifiers:

  \[ d(x) = (w \cdot x) + b \]

- **Maximum “likelihood”**: estimate conditional probability of class “+1” as:

  \[ P(y = +1 \mid d(x)) = \frac{1}{1 + \exp(Ad(x) + B)} \]

  where parameters \( A \) and \( B \) are estimated by a logistic regression fit on an independent training data set.
Results: known malware instances

- Test binaries are drawn from the same 14 families recognized by AntiVir.

(a) Accuracy per malware family

(b) Confusion of malware families

- Average accuracy: 88%.
Results: unknown malware instances

- Test binaries are drawn from the same 14 families recognized by AntiVir four weeks later.

![Graph showing accuracy per malware family](c)

- Average accuracy: 69%.

![Confusion matrix for prediction](d)
Lessons learned

- Malware collection is a crucial prerequisite for understanding new malware threats and development of appropriate protection tools.
- The main difficulty of malware collection lies in having to deal with highly dynamic and heterogeneous exploitation techniques.
- Malware analysis significantly facilitates malware understanding and the development of protection mechanisms.
- The main technique of malware analysis is execution of malware in a specially instrumented environment.
Recommended reading

Paul Baecher, Markus Koetter, Thorsten Holz, Maximillian Dornseif, and Felix C. Freiling.
The Nepenthes platform: An efficient approach to collect malware.
In Recent Advances in Intrusion Detection (RAID), pages 165–184, 2006.

Corrado Leita, Marc Dacier, and Frédéric Massicotte.
Automatic handling of protocol dependencies and reaction to 0-day attacks with ScriptGen based honeypots.
In Recent Advances in Intrusion Detection (RAID), pages 185–205, 2006.

Niels Provos and Thorsten Holz.
Virtual Honeypots: From Botnet Tracking to Intrusion Detection.

Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov.
Learning and classification of malware behavior.

Yi-Ming Wang, Doug Beck, Xuxuan Jiang, and Roussi Roussev.
Automated web patrol with Strider HoneyMonkeys: Finding web sites that exploit browser vulnerabilities.

Carsten Willems, Thorsten Holz, and Felix Freiling.
CWSandbox: Towards automated dynamic binary analysis.