Information Security
CS 526
Topic 15

Malware Defense & Intrusion Detection
Anti-Virus Software

• Goal: Find malware programs on a system, in transmission, etc.

• Main deployed approach: Signature-based detection
  – Uses pattern matching
  – Searches for known patterns of data belonging to malwares in executable programs or other types of files
  – Maintains and updates a blacklist of signatures

• Problems
  – Cannot detect new malwares, variants of malwares, etc.
  – Hard to keep up with new malware
    • More malwares are created each day than benign programs
Polymorphic Malwares

- Uses a polymorphic engine (a mutation engine or mutating engine) to generate multiple copies of the same malware that look different
  - E.g., serve a different version to each computer subject to a drive-by download attack
- Typically encrypts the majority of the code, each time with a different key is used
- Weakness: decryption code often remains the same, and may be detected and/or used as signatures
Metamorphic Malware

• A malware automatically changes itself each time it propagates
• Each new version has different code, though the same functionality
• Uses techniques that include
  – Adding varying lengths of NOP instructions, permuting use of registers, add useless instructions, use functional equivalent instructions, reorder functions, reorder data structures, etc.
Semantic, or Heuristics Based Malware Detection

- Static approach: Looks for specific code behavior instead of specific strings

- Dynamic approach: Execute the program to identify potentially malicious behavior

- Main limitations
  - Performance overhead
  - Potential of high false positives
Application Whitelisting

- Instead of finding malwares and stop them, list all known good/allowed programs and only run them.

- Typically deployed by enterprise, who can afford to maintain a list of allowed programs
CodeShield: Personalized Application Whitelisting

• Goal: Practical Application Whitelisting on Windows desktops
  – Give the user flexibility
    • Allow the user to add software to the whitelist
  – Maintain the security advantage of whitelisting
    • New software isn’t automatically allowed onto whitelist
    • Protect against certain types of Social Engineering attacks

• Not designed to stop all infection
  – Make persistence harder
  – Prevent most current attacks

• Focus on usability
  – A key challenge of many security mechanisms is the ability for a typical user to understand and use it

Christopher S. Gates, Ninghui Li, Jing Chen, Robert Proctor: CodeShield: towards personalized application whitelisting. ACSAC 2012
Analysis of Existing Security Interface

- Users are asked questions they do not know how to answer and presented with info that is difficult to understand
- Users are asked to make a decision too often
- Users are made to passively respond and provided an easy and insecure way out
Design Principles

• Reduce – decrease the number of times users are asked to make a decision.

• Simplify – ask questions that a user can understand.

• Safe (Fail Safe Defaults) – do not provide an easy and insecure way out.

• Active – avoid passively respond to security prompts.
Design of Personalized Whitelisting

<table>
<thead>
<tr>
<th>Normal Mode</th>
<th>Installation Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Only execute known software</td>
<td>• Execute all software</td>
</tr>
<tr>
<td>• Trusted Signatures = add to whitelist</td>
<td>• Executed = added to whitelist</td>
</tr>
<tr>
<td>• Trusted Installers = add to whitelist</td>
<td>• Written = added to whitelist</td>
</tr>
<tr>
<td>• All else blocked</td>
<td>• Try to exit installation mode quickly</td>
</tr>
</tbody>
</table>

- “Stopping” vs “Warning” approach

- The decision a user needs to make
  - “Do I want to install new software now”
Design Principles in Practice

- Reduce – there is a single security decision to make for installing any application

- Simplify – this paradigm more closely matches how typical users understand their actions. “I’m adding something new”

- Safe (Fail Safe Defaults) – Not allowing new code is the easiest action

- Active – In order to add new software, the user needs to actively participate and initiate the action.
This dual mode can more closely match the mental model of a typical user.

- Users may not understand “Do you want to allow this program to make changes”
- But most can be educated about “Do you want to add something new to your computer right now”

Furthermore, users can be educated about when not to enter installation mode.
The Burden Benefit of Installation Mode

• Simple switch to installation mode
  – Advantage – it’s easy
  – Disadvantage – user may enter installation mode often

• High overhead switch to installation mode (ex. reboot)
  – Advantage – it makes a user less likely to switch unless needed
  – Disadvantage – high overhead may lead to annoyance

• Advantage of reboot
  – Clear out memory, malware in memory can’t take advantage of installation mode
  – Minimal number of applications active just after reboot
User Study

- 35 person user study running CodeShield for 6 weeks
- Longest use of CodeShield is 203 days (8 switches, 25 days/switch), next is 168 days (13 switches, 13 days/switch).
- Participants sat through a 30 minute training session
- Then installed CodeShield (standalone installer)
- Take a survey, Run for 6 weeks, Take a survey
- Uninstall if they want to
- 7 of 38 participants continued to use CodeShield at least 3 months after study ended.
  - 5 were using reboot only client
  - 2 using switch or reboot
Switches to Installation Mode

- **Switch**
  - Median - 17
  - Useful - 13

- **Reboot**
  - Median - 3.5
  - Useful - 3.5
Network IDSs

- Deploying sensors at strategic locations
  - E.G., Packet sniffing via *tcpdump* at routers
- Inspecting network traffic
  - Watch for violations of protocols and unusual connection patterns
- Monitoring user activities
  - E.g., look into the data portions of the packets for malicious code, or known exploits
- Inspection ability limited by encryption
  - Data portions and some header information can be encrypted
  - The decryption engine may still be there, especially for exploit
Architecture of Network IDS

Packet capture libpcap

TCP reassembly

Protocol identification

Signature matching (& protocol parsing when needed)
Host-Based IDSs

• Running on a single host
• Monitoring
  – Shell commands
  – System call sequences
  – Etc.
Misuse Detection (aka Signature detection)

Intrusion Patterns

Example: \texttt{if (src\_ip == dst\_ip) then “land attack”}

*This causes some TCP implementation to keep sending ack packet to itself.

\textbf{Can't detect new attacks}
Anomaly Detection

Problem: Relatively high false positive rate
- Anomalies can just be new normal activities.
- Anomalies caused by other element faults
  - E.g., router failure or misconfiguration, P2P misconfiguration

- probable intrusion
Problems with Current IDSs

- Inaccuracy for exploit based signatures
- Cannot recognize unknown anomalies/intrusions
- Cannot provide quality info for forensics or situational-aware analysis
  - Hard to differentiate malicious events with unintentional anomalies
    - Anomalies can be caused by network element faults, e.g., router misconfiguration, link failures, etc., or application (such as P2P) misconfiguration
  - Cannot tell the situational-aware info: attack scope/target/strategy, attacker (botnet) size, etc.
Key Metrics of IDS/IPS

- Algorithm
  - Alarm: $A$
  - Intrusion: $I$
  - Detection (true alarm) rate: $P(A|I)$
    - False negative rate $P(\neg A|I)$
  - False alarm (aka, false positive) rate: $P(A|\neg I)$
    - True negative rate $P(\neg A|\neg I)$
• See Slides on "The Base Rate Fallacy and its Implications for the Difficulty of Intrusion Detection"
Coming Attractions …

• Discretionary Access Control