CSE 227
Computer Security

Spring 2010

Vulnerability Automation

Stefan Savage
• Brumley et al, Automated Patched-based Exploit Generation, Oakland 2008
  – Automatically generate attacks

• Vigilante: End-to-end Containment of Internet worms, SOSP 2005
  – Automatically generate and distributed defenses

• Both are about speed
Exploit generation

• Premise: patches provide blueprint to vulnerability
  – Anecdotally true
  – How does this get done now?

• Challenge: automating process

• Assumptions:
  – Patch fixes vulnerability by restricting input
One idea

• Diff old version with new version
• Define safety policy on dynamic state
• Use constraint solver to identify inputs caught by new check and see if any violate safety policy
• Similar idea:
Tricky to do with binaries...

- Incomplete CFG
- Data aliasing
- But doable up to a point...
Basic procedure

• Generate binary diff (off-the-shelf tool)
• Model binary x86 instruction control/data flow
  – Identify new predicates added
• Try to find weakest precondition for failing the new predicate
  – Dynamic version (take trace), static version (explore set of acyclic paths), and combination
• Use constraint solver to find inputs satisfying WP
• Check if those inputs would violate safety policy (e.g., by doing dynamic taint checking)
## Bottom line

<table>
<thead>
<tr>
<th>Application</th>
<th>Vulnerability</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASPNet_Filter</td>
<td>Information Disclosure</td>
<td>29</td>
</tr>
<tr>
<td>GDI</td>
<td>Hijack Control</td>
<td>135</td>
</tr>
<tr>
<td>PNG</td>
<td>Hijack Control</td>
<td>131</td>
</tr>
<tr>
<td>IE COMCTL32 (B)</td>
<td>Hijack Control</td>
<td>456</td>
</tr>
<tr>
<td>IGMP</td>
<td>Denial of Service</td>
<td>186</td>
</tr>
</tbody>
</table>
Countermeasures considered

- Obfuscate patches
- Encrypt patches, distribute the key only when everybody’s ready
- Speed up patch distribution
Vigilante

• Focused on network worms exploiting previously unknown vulnerability
Worms: context

• Autonomous, active code that can replicate to remote hosts via vulnerability
• Because they propagate autonomously, they can spread very quickly
• This was a really big deal between 2001-2004
How to think about worms

• Reasonably well described as infectious epidemics
  – Simplest model: Homogeneous random contacts

• Classic SI model
  • \( N \): population size
  • \( S(t) \): susceptible hosts at time \( t \)
  • \( I(t) \): infected hosts at time \( t \)
  • \( \beta \): contact rate
  • \( i(t) \): \( I(t)/N \), \( s(t) \): \( S(t)/N \)

\[
\frac{dI}{dt} = \beta \frac{IS}{N} \\
\frac{dS}{dt} = -\beta \frac{IS}{N} \\
\frac{di}{dt} = \beta i(1 - i) \\
i(t) = \frac{e^{\beta(t-T)}}{1 + e^{\beta(t-T)}}
\]

courtesy Paxson, Staniford, Weaver

![Graph showing number of scans vs. hour of the day](image)

- First ~1min behaves like classic random scanning worm
  - Doubling time of ~8.5 seconds
  - CodeRed doubled every 40mins

- >1min worm starts to saturate access bandwidth
  - Some hosts issue >20,000 scans per second
  - Self-interfering (no congestion control)

- Peaks at ~3min
  - >55million IP scans/sec

- 90% of Internet scanned in <10mins
  - Infected ~100k hosts (conservative)

Vigilante

• Focused on network worms exploiting previously unknown vulnerability

• Goals:
  – Identify new attack/exploit as it happens
  – Distribute information quickly and securely
  – Build defenses against vulnerability (try to generalize)
Vigilante’s components

- Detection
- SCA generation
- SCA distribution
- SCA verification
- Protection
Detectors

• Some hosts run exploit detectors
  – Taint tracking, no execute bits, etc...

• Assumptions?
Dynamic dataflow analysis

- high coverage and low false positive rate
- allows direct extraction of verification information
Self-certifying alerts

• Detect generates and SCA that describes the vulnerability
  – Then distributes SCA to other vulnerable hosts

• Receiving hosts validate SCA
  – In sandboxed environment

• What problem does this solve?
Distribute SCA via P2P network

- Why via P2P network?
- What is goal here?
Protection

• Hosts generate “filter” from SCA
• **Dynamic data and control flow analysis**
  – run vulnerable application in a sandbox
  – track control and data flow from input messages
  – compute conditions that determine execution path
  – filter blocks messages that satisfy conditions
• Specific filters have **zero** false positives
• General filters (relaxation by heuristics) can have false positives
  – Why bother with these?
Vulnerability filter generation

```
mov al,[netbuf]
mov cl,0x31
cmp al,cl
jne out
xor eax,eax
loop:
    mov [esp+eax+4],cl
    mov cl,[eax+netbuf+1]
    inc eax
test cl,cl
    jne loop
out:
```

**netbuf**

| 0x31 | 0x24 | 0x67 | 0x42 | 0x0 |

**Conditions:**

- netbuf[0] == 0x31
- netbuf[1] != 0
- netbuf[2] != 0
Time to generate SCAs

<table>
<thead>
<tr>
<th>Dynamic dataflow</th>
<th>Slammer</th>
<th>Blaster</th>
<th>CodeRed</th>
<th>Slammer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>18</td>
<td>206</td>
<td>2667</td>
<td>2</td>
</tr>
</tbody>
</table>
Time to verify SCAs

<table>
<thead>
<tr>
<th></th>
<th>SCA verification time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slammer</td>
<td>10</td>
</tr>
<tr>
<td>Blaster</td>
<td>18</td>
</tr>
<tr>
<td>CodeRed</td>
<td>75</td>
</tr>
</tbody>
</table>
Time to generate filters

- Slammer: 24 ms
- Blaster: 273 ms
- CodeRed: 3402 ms
For next time...

• Web security I