ROPMEMU:
A Framework for the Analysis of Complex Code-Reuse Attacks

Mariano Graziano, Davide Balzarotti, Alain Zidouemba

Cisco Systems, Inc.
Eurecom

AsiaCCS 2016 - Xi’an, China
CODE INJECTIONS

OS KERNEL
CODE INJECTIONS

Attackers load or inject malicious code (or modify the existing one)
CODE REUSE - ROP

OS KERNEL
CODE REUSE - ROP

OS KERNEL
MOTIVATIONS

- HW and OS **countermeasures** force ROP adoption
MOTIVATIONS

- HW and OS **countermeasures** force ROP adoption
- Vogl et al. [NDSS 2014] — Persistent ROP **rootkit**
- ROP as an **obfuscation** technique adopted by malware
- All existing tools cope with **injected code**
- Lack of RE **tools** to analyze/dissect/decompile ROP
Verbosity
CHALLENGES

[C1] Verbosity
[C2] Lack of immediate values
CHALLENGES

[C1] Verbosity
[C2] Lack of immediate values
[C3] Stack based instruction chaining
CHALLENGES

[C1] Verbosity
[C2] Lack of immediate values
[C3] Stack based instruction chaining
[C4] Conditional branches
CHALLENGES

[C1] Verbosity
[C2] Lack of immediate values
[C3] Stack based instruction chaining
[C4] Conditional branches
[C5] Return to functions
CHALLENGES

[C1] Verbosity
[C2] Lack of immediate values
[C3] Stack based instruction chaining
[C4] Conditional branches
[C5] Return to functions
[C6] Dynamically generated chains
CHALLENGES

[C1] Verbosity
[C2] Lack of immediate values
[C3] Stack based instruction chaining
[C4] Conditional branches
[C5] Return to functions
[C6] Dynamically generated chains
[C7] Stop condition
CHALLENGES

[C1] Verbosity ✔
[C2] Lack of immediate values ✔
[C3] Stack based instruction chaining ✔
[C4] Conditional branches ☐
[C5] Return to functions ☐
[C6] Dynamically generated chains ☐
[C7] Stop condition ☐
CHALLENGES

[C1] Conditional branches
- Lu et al. - ACSAC 2012
- Yadegari et al. - S&P 2015

[C2] Return to functions
- Stancill et al. - RAID 2013

[C3] Dynamically generated chains
- Lu et al. - ACSAC 2012

[C4] Conditional branches
[C5] Return to functions
[C6] Dynamically generated chains
[C7] Stop condition
ROPMEMU framework adopts many techniques:

- Memory forensics
- Code emulation
- Multi-path execution
- CFG recovery
- Compiler transformations
ROPMEMU
<table>
<thead>
<tr>
<th>CHAIN</th>
<th>INSTRUCTIONS</th>
<th>GADGETS</th>
<th>BLOCKS</th>
<th>BRANCHES</th>
<th>FUNCTIONS</th>
<th>CALLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPY</td>
<td>414,275</td>
<td>184,126</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DISPATCHER</td>
<td>63,515</td>
<td>28,874</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>PAYLOAD</td>
<td>6320</td>
<td>2913</td>
<td>34</td>
<td>26</td>
<td>9</td>
<td>17</td>
</tr>
</tbody>
</table>
RESULTS - CFG
LIMITATIONS

- Prone to anti-acquisition
- Prone to anti-emulation
- Lack of completeness on arbitrary inputs
CONCLUSIONS

- Source code: https://github.com/vrtadmin/ROPMEMU
- First public tool to analyze ROP payloads
- Tested on the most complex public threat
QUESTIONS?

Mariano Graziano
magrazia@cisco.com
@emd3l