Micro-Virtualization Memory Tracing to Detect and Prevent Spraying Attacks

Stefano Cristalli, Mattia Pagnozzi, Mariano Graziano, Andrea Lanzi, Davide Balzarotti

Universita’ degli Studi di Milano
Cisco Systems, Inc.
Eurecom

USENIX Security 2016 - Austin, USA
SPRAYING

- Traditional code based spraying
- JIT spraying
- Data spraying and stack pivoting
RELATED WORK

HEAP SPRAYING

- Egele et al. - DIMVA 09
- Nozzle - USENIX Security 09
- BuBBLE - ESSoS 10
RELATED WORK

HEAP SPRAYING
- Egele et al. - DIMVA 09
- Nozzle - USENIX Security 09
- BuBBLE - ESSoS 10

JIT SPRAYING
- JITsec - VEE 06
- Bania - Whitepaper 10
- JITDefender - IFIP 11
- Lobotomy - Ares 14
RELATED WORK

HEAP SPRAYING
Egele et al. - DIMVA 09
Nozzle - USENIX Security 09
BuBBLE - ESSoS 10

JIT SPRAYING
JITsec - VEE 06
Bania - Whitepaper 10
JITDefender - IFIP 11
Lobotomy - Ares 14

DATA SPRAYING
EMET - Microsoft 09
Browser solutions
RELATED WORK

HEAP SPRAYING
- Egele et al. - DIMVA 09
- Nozzle - USENIX Security 09
- BuBBLE - ESSoS 10

- No comprehensive and scalable solutions vs spraying
- No OS agnostic solutions

DATA SPRAYING
- Bania - Whitepaper 10
- JITDefender - IFIP 11
- Lobotomy - Ares 14

EMET - Microsoft 09
Browser solutions
MEMORY FOOTPRINT

Internet Explorer 8 - Normal Behavior vs CVE-2011-1996

Time (s)

Memory (MB)

3 Tabs - Google/Slashdot/University Web Site
4 Tabs - Maps/News Site 1/News Site 2/Youtube
CVE-2011-1996 - Default Metasploit Exploit
GOALS

‣ The system should be completely independent from the memory allocator used by the protected application.

‣ The system should not require any OS dependent information.

‣ The overhead introduced by our system should be “reasonable”.

‣ Modular framework based on plugins.
ARCHITECTURE OVERVIEW

```
VulnBrowser
int interpreter()
{
    ...
    malloc(...);
    ...
}
```

GET / HTTP/1.1

explo.js

http://evil.com

User mode
Kernel mode
malloc

Kernel

\[ \text{Trap!} \]

\#1023
\[ \ldots \]
\#0

Non-root mode
Root mode

Memory Tracer  Attack Detector

Hypervisor

Hardware
Creation/Modification/Removal of the page: Detected by looking at the PTE
Creation/Modification/Removal of the page table: Detected by looking at the PDE
INTERFERENCE PROBLEM

- Overhead issues:
  - A modification of the memory page of the running process creates a side effect modification of a memory page in another process
  - Due to kernel memory operations optimizations
INTERFERENCE PROBLEM

- Overhead for Internet Explorer 10 (IE): 22%
- Overhead for Acrobat Reader and Firefox on top of IE: 63%
- IE’s overhead increases from 22% to 63%
MICRO-VIRTUALIZATION

- Each monitored process runs inside its own virtual memory sandbox
- Graffiti enables the memory protection only for the actual running and monitored process
Micro-virtualization is based on EPT:

- Select a set of physical pages to monitor the target process
- One EPT pointer (EPTP) per process
STATIC ANALYZER

Set of heuristics to detect the different spraying techniques:

- Malicious code detector [ACSAC10, SEC11]
Set of heuristics to detect the different spraying techniques:

- Malicious code detector [ACSAC10, SEC11]
- Self-unpacking shellcode detector [ACSAC07]
Set of heuristics to detect the different spraying techniques:

- Malicious code detector [ACSAC10, SEC11]
- Self-unpacking shellcode detector [ACSAC07]
- Data spraying detector [RAID13, RAID15]
EXPERIMENTS: OVERHEAD

- The memory tracer is always active
- Stress suite results (8MB every 2s):
  - Windows 7: 24%
  - Linux 3.2: 25%
EXPERIMENTS: OVERHEAD

The graph illustrates the relationship between Total Overhead and Activation Threshold. As the Activation Threshold increases, the Total Overhead decreases significantly. The curve shows a steep decline between 200 and 300, indicating a more pronounced reduction in overhead in this range.
## DETECTION ACCURACY

<table>
<thead>
<tr>
<th>CVE</th>
<th>Application</th>
<th>Exploit Technique</th>
<th>Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-0248</td>
<td>Adobe Flash player</td>
<td>ROP + packed sc</td>
<td>Yes</td>
</tr>
<tr>
<td>2011-0609</td>
<td>Adobe Reader</td>
<td>JIT + packed sc</td>
<td>Yes</td>
</tr>
<tr>
<td>2011-2462</td>
<td>Adobe Reader</td>
<td>ROP + packed sc</td>
<td>Yes</td>
</tr>
<tr>
<td>2010-2883</td>
<td>Adobe Reader</td>
<td>Ret2Lib + packed sc</td>
<td>Yes</td>
</tr>
<tr>
<td>2011-1996</td>
<td>IExplorer</td>
<td>ROP</td>
<td>Yes</td>
</tr>
<tr>
<td>2009-2477</td>
<td>Firefox</td>
<td>Plain Shellcode</td>
<td>Yes</td>
</tr>
</tbody>
</table>
DETECTION ACCURACY

- System tested also with 1000 malicious PDF, 1000 benign PDF documents and top 1000 Alexa websites
- Conservative threshold (150MB)
- Graffiti detected all the attacks with zero false alarms
GLOBAL EXPERIMENTS

- Final global test in which real users use Graffiti during their everyday activities for a total of 8-10 hours per day in a period of 7 days

- Conservative threshold of 150MB on IE8 for Windows 7 machines
GLOBAL EXPERIMENTS

- 492 distinct web pages visited

- Detectors activated 55 times (~8 times per day)

- 12 alerts on pages that seemed to be benign. A closer inspection of the false positive shows the data spraying detector to be the only responsible
CONCLUSIONS

- First efficient and comprehensive solution to defeat spraying
- Micro-virtualization
- Open source
Code available at

https://github.com/graffiti-hypervisor/graffiti-hypervisor
QUESTIONS?

Mariano Graziano
magrazia@cisco.com
@emd3l