智慧安全 连接赋能
whoami

- Security researcher at Cisco Talos
- Ph.D. from Telecom ParisTech/Eurecom
- Hackademic
- Malware analysis / Memory forensics / Mitigations
SPRAYING

- Traditional code based spraying
- JIT spraying
- Data spraying and stack pivoting
SPRAYING - TIMELINE

- 2001: telnetd - Team TESO, Microsoft IIS - eEye
  - 2004: IE6 - Skylined
- 2007: Heap feng shui - Alex Sotirov
- 2010: JIT Spraying - D. Blazakis
  - 2013: Data spraying
  - 2014: physmap - V. Kemerlis, Kernel JIT BPF - K. McAllister, Pool Spraying - j00ru
SPRAYING – 64 bit

- Spraying still a valuable technique
  - UAF vulnerabilities
  - Flaws in ASLR implementation
  - Particular types of vulnerabilities
  - 32bit processes in 64bit OS

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 10
- 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 10
- 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

JIT SPRAYING
- JITsec – VEE 06
- No comprehensive and scalable solutions vs spraying
- No OS agnostic solutions
- JITDefender – IFIP 10
- Lobotomy – Ares 14

DATA SPRAYING
- EMET – Microsoft 09
- Browser solutions
MEMORY FOOTPRINT

IE8 Allocations - Comparison

Time (s)

Memory (MB)

0  50  100  150  200  250

0  20  40  60  80  100  120

6 tabs
CVE-2011-1196 80% success rate delayed
RELIABILITY – HEAP SPRAYING

- 80%: 131 MB – CVE-2011-1996
- 0%: < 131 MB - CVE-2011-1996
- Possible Watering hole scenarios
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 the system should be “reasonable”.
- Modular framework based on plugins.
VIRTUALIZATION – VT-x

- vmexit/vmentry transitions
- vmcall instruction
- Virtual Machine Control Structure (VMCS)
- Extended Page Tables (EPT):
  - EPT misconfigurations
  - EPT violations
ARCHITECTURE OVERVIEW
ARCHITECTURE OVERVIEW
MEMORY TRACER

- Detection by looking at the PTEs:
  - page creation/modification/removal

- Detection by looking at the PDEs:
  - page table creation/modification/removal
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 optimizations
INTERFERENCE PROBLEM

- Overhead for Internet Explorer 10 (IE): 22%
- Overhead for Acrobat Reader and Firefox on top of IE10: 63%
- IE’s overhead increased from 22% to 63%
MICRO-VIRTUALIZATION

- Each monitored process runs inside its own virtual memory sandbox
- Graffiti enables the memory protection only for actual running and monitored process
MICRO-VIRTUALIZATION - DETAILS

- Micro-virtualization is based on EPT:
  - Select a set of physical pages to monitor the target process
  - One EPT Pointer (EPTP) per process
ARCHITECTURE OVERVIEW
STATIC ANALYZER

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

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

- 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

![Graph showing the relationship between activation threshold and total overhead. The graph indicates a decreasing trend in total overhead as the activation threshold increases.]
## 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 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
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
- Detector 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 to be the only responsible
GRAFFITI - CONFIGURATION

- godfather/hyperdbg/hyperdbg_host.c:
  - `#define TARGET1 "iexplorer.exe"`
- godfather/hyperdbg/godfather.c:
  - `#define EXPLOIT_THRESHOLD THRESHOLD_150MB`
- Compilation:
  - `make -f Makefile.windows`
DEMO

DEMO 0x01
LIMITATIONS

- Possible evasion of the three heuristics described:
  - Code pointer frequency
  - Shellcode frequency analysis
- Big chunks and then ad-hoc allocation functions
- Current supported OS: Windows and Linux
- Architecture dependent (Intel)
FUTURE

- Propose stronger heuristics
- More comprehensive testing:
  - New browser versions
  - Kernel Heap spraying
CONCLUSIONS

- First efficient and comprehensive solution to defeat spraying
- Micro-virtualization
- Open source
SOURCE CODE & PAPER
- https://github.com/graffiti-hypervisor/graffiti-hypervisor

- “Micro-Virtualization Memory Tracing to Detect and Prevent Spraying Attacks”

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

25th USENIX Security Symposium
THANK YOU

智慧安全 连接赋能

magrazia@cisco.com
@emd3l