OXYMORON

Stefan Nürnberg, Michael Backes
STEFAN

PhD

SAARLAND UNIVERSITY

Germany
EARLY 2011
EARLY 2011

You are an IT Security Competence Centre!
EARLY 2011

Center for IT Security Privacy and Accountability

CISPA

Cyber Intelligence Sharing and Protection Act
MAKING FINE-GRAINED MEMORY RANDOMISATION PRACTICAL BY ALLOWING CODE SHARING
Oxymoron /ˌɒks.ɪˈmɔrən/ (noun)
Greek. A figure of speech that combines contradictory terms.
OXYMORON

- Code Reuse Attacks
  - Fine-Grained Randomization
- Modern OS
  - Shared Libraries

ILR: Where'd My Gadgets Go?
Software Diversity
Smashing the Gadgets
Binary Stirring

Oxymoron
CODE-REUSE ATTACKS

Program

I’m the most lovely and benign program you can imagine. I embody functionality like document analysis of different types. I’m not harmful in any way.

Adversary

The same program

I am a lovely and benign program that just runs on your computer without the intent of doing anything harmful. Not even my words are malicious.
ASLR

Program

Adversary

I’m the most lovely and benign program you can imagine. I embody functionality like document analysis of different types. I’m not harmful in any way.
FINE-GRAINED RANDOMIZATION

Program

1

lovely

program

am

the

most

Adversary

?
CROSS-PROCESS RANDOMIZATION

Process 1
- I
- program
- am
- the
- most

Process 2
- program
- am
- I
- lovely
- most
SO WE’VE SOLVED
THE PROBLEM.
RIGHT?
NOT QUITE
## OBSERVATION

Load averages for the last 1, 5, 15 minutes: 0,36, 0,39, 0,21

<table>
<thead>
<tr>
<th>Process Name</th>
<th>Virtual Memory</th>
<th>Resident Memory</th>
<th>% CPU</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>compiz</td>
<td>396,3 MiB</td>
<td>171,6 MiB</td>
<td>100%</td>
<td>1831</td>
</tr>
<tr>
<td>Xorg</td>
<td>150,8 MiB</td>
<td>46,6 MiB</td>
<td>0</td>
<td>816</td>
</tr>
<tr>
<td>evolution-calendar-factory</td>
<td>116,9 MiB</td>
<td>29,6 MiB</td>
<td>0</td>
<td>1896</td>
</tr>
</tbody>
</table>

![Activity Monitor](image)

<table>
<thead>
<tr>
<th>Process</th>
<th>Memory</th>
<th>Threads</th>
<th>Ports</th>
<th>PID</th>
<th>User</th>
<th>Private Mem</th>
<th>Shared Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>302.7 MB</td>
<td>6</td>
<td>443</td>
<td>479</td>
<td>stefan</td>
<td>255.3 MB</td>
<td>72.2 MB</td>
</tr>
<tr>
<td></td>
<td>126.0 MB</td>
<td>31</td>
<td>244</td>
<td>304</td>
<td>stefan</td>
<td>119.5 MB</td>
<td>35.0 MB</td>
</tr>
<tr>
<td></td>
<td>79.8 MB</td>
<td>36</td>
<td>577</td>
<td>303</td>
<td>stefan</td>
<td>74.7 MB</td>
<td>107.5 MB</td>
</tr>
<tr>
<td></td>
<td>35.4 MB</td>
<td>6</td>
<td>65</td>
<td>219</td>
<td>root</td>
<td>70.9 MB</td>
<td>12.7 MB</td>
</tr>
<tr>
<td></td>
<td>55.2 MB</td>
<td>10</td>
<td>81</td>
<td>999</td>
<td>stefan</td>
<td>54.5 MB</td>
<td>81.2 MB</td>
</tr>
<tr>
<td></td>
<td>57.9 MB</td>
<td>5</td>
<td>510</td>
<td>115</td>
<td>_window</td>
<td>52.2 MB</td>
<td>111.4 MB</td>
</tr>
<tr>
<td></td>
<td>56.4 MB</td>
<td>10</td>
<td>81</td>
<td>3421</td>
<td>stefan</td>
<td>51.8 MB</td>
<td>91.0 MB</td>
</tr>
<tr>
<td></td>
<td>44.3 MB</td>
<td>10</td>
<td>84</td>
<td>1167</td>
<td>stefan</td>
<td>44.1 MB</td>
<td>88.5 MB</td>
</tr>
</tbody>
</table>
SHARING CODE AND DATA

Browser (Process 1)

Main Executable

C Library

Threading Lib

PDF Viewer (Process 2)

Main Executable

C Library

Threading Lib
SHARING CODE AND DATA

Browser (Process 1)

PDF Viewer (Process 2)
IMPACT
234 Processes

Firefox started, $\Sigma = 1435$ MiB
Idle Desktop, $\Sigma = 1388$ MiB
ON THE QUEST TO A SOLUTION
ONE POSSIBLE SOLUTION

• Split Code in pieces
ONE POSSIBLE SOLUTION

Problem:
They still need to be connected!

• Share pieces individually
• Enough entropy
VALUES ENCODED IN CODE

1

Call @foo

Indirection

@foo: 0x82ab7
OXYMORON

• Enables individual pieces to be shared
• Each process can be randomised separately
• Hides all addresses from the attacker
• It’s a modified x86 calling convention

PALACE

(Position AND Layout AGNOSTIC CODE)
PALACE

This is how x86 is structured
16 Bit Feature: Segmentation
SEGMENTATION

- To move programs in memory before MMUs existed
- Dedicated registers to add offsets to code/data/stack

```
call 0xD722B
```

```
GDT
Code: + 0x1000
Data: + 0x2000
```

```
call 0xD822B
```
X86 INDIRECT CALLS

call *0x1234

push %ebp
mov %esp, %ebp

call 0x9876

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1234</td>
<td>0x9876</td>
</tr>
<tr>
<td>0x1238</td>
<td></td>
</tr>
<tr>
<td>0x123C</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
RATTLE
Randomisation-agnostic Translation Table

GDT
Code: + 0x1230

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1234</td>
<td>0x9876</td>
</tr>
<tr>
<td>0x1238</td>
<td>...</td>
</tr>
<tr>
<td>0x123C</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

call *0x4

0x9876

push %ebp

mov %esp, %ebp

call 0x9876

call *0x1234
**RATTLE**

Randomisation-Agnostic Translation Table

---

```
call %fs:*0x4
push %ebp
mov %esp, %ebp
```

---

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1234</td>
<td>0x9876</td>
</tr>
<tr>
<td>0x1238</td>
<td>…</td>
</tr>
<tr>
<td>0x123C</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
ADVANTAGES

• %fs does not reveal an address

• Encodings of \texttt{call} and \texttt{mov} are all identical across randomisations

• Code stored in a memory page can be shared among different processes

• Different processes have different permutation of memory pages

• Pure user-mode implementation (works on every Linux kernel)
OXYMORON

Executable

Piece 1

mov $4, %eax
add $6, %eax

Piece 2
Disk

Executable
- Piece 1
- Piece 2

Library
- Piece L1
- Piece L2

Process 1

Process 2

OXYMORON
mov 0x807f364,%edx

mov %fs:0x8a,%edx

nop

jmp %fs:0x7f

cmp %ebx,%edx

jb 80480fa

mov %fs:0x991,%eax

test %eax,%eax

je 804812b

sub $0xc,%esp

push %fs:0x44

call 8066db7

cmp %ebx,%edx

jb 80480fa

mov $0x8066db7,%eax

test %eax,%eax

je 804812b

sub $0xc,%esp

push $0x8073fa8

call 8066db7

cmp %ebx,%edx

jb 80480fa

mov %fs:0x991,%eax

test %eax,%eax

je 804812b

sub $0xc,%esp

push %fs:0x44

call *%fs:0x21
PERFORMANCE RESULTS
Runtime Performance Overhead

![Bar chart showing runtime performance overhead for different benchmarks. The chart compares Page Jump (avg. 0.5%) and RaTTle (avg. 2.7%).]
PERFORMANCE RESULTS

File Size Overhead

- ELF File (avg. 1.76%)
- Instructions (avg. 12%)
OXYMORON?
MEMORY CORRUPTION EXPLOITS (DEVELOPMENT)
ATTACK TARGETS

Distribution of exploits by type of attacked application

Source: Kaspersky Security Bulletin 58265