Effective Entropy for Memory Randomization Defenses

William Herlands, Thomas Hobson, Paula Donovan

7th Workshop on Cyber Security Experimentation and Test
18 August 2014

This work is sponsored by Assistant Secretary of Defense for Research & Engineering under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.
User Space Memory Randomization

- User-space memory randomization defenses protect against memory-corruption attacks
  - Attackers require knowledge of the layout of memory
  - Defenses randomize layout
- E.g. Address Space Layout Randomization (ASLR)

Image Reference: Didier Stevens, yaisc.com
Metric Requirements

- Current metrics use **exploits** or **entropy** to evaluate randomization technologies

<table>
<thead>
<tr>
<th>Exploits</th>
<th>Entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pro</strong></td>
<td><strong>Pro</strong></td>
</tr>
<tr>
<td>• “Real life,” holistic test against adversary technology</td>
<td>• Quantitative, information theoretic</td>
</tr>
<tr>
<td><strong>Con</strong></td>
<td><strong>Con</strong></td>
</tr>
<tr>
<td>• Anecdotal</td>
<td>• Does not consider threat models</td>
</tr>
<tr>
<td>• Not comparable</td>
<td>• Not holistic</td>
</tr>
<tr>
<td>• Biased towards existing exploits</td>
<td></td>
</tr>
</tbody>
</table>

We developed Effective Entropy, a metric which is quantitative, comparable, and indicative of adversary workload
# ASLR entropy improvements

<table>
<thead>
<tr>
<th>Entropy (in bits) by region</th>
<th>Windows 7</th>
<th>Windows 8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>32-bit</td>
<td>64-bit</td>
</tr>
<tr>
<td>Bottom-up allocations (opt-in)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Stacks</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>Heaps</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Top-down allocations (opt-in)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PEBs/TEBs</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>EXE images</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>DLL images</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Non-ASLR DLL images (opt-in)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

* 64-bit DLLs based below 4GB receive 14 bits, EXEs below 4GB receive 8 bits

ASLR entropy is the same for both 32-bit and 64-bit processes on Windows 7

64-bit processes receive much more entropy on Windows 8, especially with high entropy (HE) enabled

We developed Effective Entropy, a metric which is quantitative, comparable, and indicative of adversary workload.
Outline

• Background on Memory Randomization
• Effective Entropy
• Evaluation
User Memory Layout

- Multiple sections required to run a program:
  - Code to run ("Program Image")
  - Variables used in execution ("Heap" and "Stack")
  - Kernel functions ("VDSO")
  - Libraries ("MMAP")

MMAP: Memory Map
VDSO: Virtual Dynamically-linked Shared Objects
User Memory Layout

- In a static layout a variety of attacks are possible since an adversary can trivially know the location of objects in memory.
Entropy in User Memory Layout

- Memory randomization techniques randomize sections’ location in memory
  - Base address randomization
  - E.g. Ubuntu 32-bit provides 256 \((2^8)\) possible MMAP locations

![Diagram of Linux 32-bit memory layout]

- Program Image
- Heap
- Stack
- VDSO
- MMAP

![Sketch of a thief thinking about library location]
Entropy is a means of measuring randomness

- E.g. MMAP base can take $2^8$ values with equal probability so it has 8 bits of entropy
- Standard calculation of entropy measures total uncertainty of a variable in bits

$$H(X) = - \sum_{i=1}^{n} p(x_i) \log p(x_i)$$
Static Address Space Layout Randomization (ASLR)

- Static (non-PIE) ASLR randomizes base addresses of memory sections
  - Heap, stack, VDSO, and MMAP randomized independently
  - Program image not randomized
- Implemented in most modern operating systems
  - Windows, OS X, Linux, OpenBSD

0 Bits
Program Image

13 Bits
Heap

19 Bits
Stack

8 Bits
VDSO

8 Bits
MMAP
• Position Independent Executable (PIE) ASLR randomizes all base addresses of memory
  – Heap, stack, VDSO, MMAP, and program image randomized independently

• Increasingly prevalent
  – Compiler option in GCC
  – Default in OpenBSD 5.3
• Fine grain randomization
  – Randomize smaller blocks, not only section base addresses
  – E.g. Independent library randomization
Outline

• Background on Memory Randomization
• Effective Entropy
• Evaluation
Connections in User Memory Layout

- Not so simple
- Interconnectedness
  - Control flow instructions
  - Pointers

- Program Image
- Heap
- Stack
- VDSO
- MMAP
Connections in User Memory Layout

0. Attacker uses buffer overflow to write the address of ‘`pop %ecx, jmp *%ecx`’ gadget into ret addr, followed by the address of `exec_ptr`

1. Function attempts to return, control redirected to gadget in Program Image

2. Pops `&exec_ptr` from stack and jumps to value at that address (exec function in MMAP)

X - Attacker supplied values
Connections in User Memory Layout

--- Absolute connections
Read-only pointers
Direct jumps

--- Dynamic connections
Writable pointers
Indirect branches

Linux 32-bit Base Entropy
Program Image
Heap
Stack
VDSO
MMAP

0 Bits
13 Bits
49 Bits
8 Bits
8 Bits
Connections in User Memory Layout

Absolute connections
- Read-only pointers
- Direct jumps
  - 0 Bits

Dynamic connections
- Writable pointers
- Indirect branches
  - 13 Bits
- 8 Bits

Difficult to determine, requires runtime analysis
- 8 Bits

Linux 32-bit Base Entropy
- Program Image
- Heap
- Stack
- VDSO
- MMAP
Identifying Dynamic Pointers

Run deterministic execution path twice:

Examine writeable memory at every control flow statement

Eliminate inconsistencies

Identify dynamic pointers
Effective Entropy (EffH)

$$\text{EffH}_s = \min \left\{ h_s, \min(H^x_{\text{conn}}), \min(H^p_{\text{conn}}) + \min(H^x) \right\}$$

$$H^p_{\text{conn}} = \{ h^p_j : \exists \text{connection}(j, s) \}$$

$$H^x_{\text{conn}} = \{ h^x_j : \exists \text{connection}(j, s) \}$$
Measuring Randomization Technologies

- EffH is a property of a randomization technology and threat model
- On any particular platform, sufficiently large programs exhibit similar memory interconnections
  - E.g. Global Offset Table → Library functions
- Any non-degenerate execution of a program is representative of all non-degenerate executions with respect to memory usage
  - Connections are drawn from same distribution
Outline

• Background on Memory Randomization
• Effective Entropy
• Evaluation
Experiment Overview

• Goals:
  – Evaluate current and emerging security technologies against realistic threat models
  – Assess utility of the EffH metric

Security Technologies Considered

• Static ASLR
• PIE ASLR
• Independent Library Randomization
  – Simulation of fine grain randomization technique

Threat Models

• Consider viable adversaries at multiple tiers of sophistication and resources

Threat Model

• Moderate Adversary
  – Control flow hijacking vulnerability
  – Modern exploitation methods including Return Oriented Programming (ROP)

• Memory Disclosure Adversary
  – Control flow hijacking vulnerability
  – Modern exploitation methods including ROP
  – *Memory disclosure vulnerability that reveals location of one memory section*

Return Oriented Programming

• Use snippets of executable code called “ROP gadgets”
• Combine gadgets to create a custom exploit
Static ASLR provides zero bits of EffH to Moderate Adversary
Memory Disclosure Adversary – PIE ASLR

PIE ASLR provides zero bits of EffH to Adversary disclosing Program Image
Memory Disclosure Adversary - Fine Grain

Independent Library Randomization

Fine Grain provides 8 bits of EffH for some but not all libraries
Memory Disclosure Adversary - Fine Grain

Independent Library Randomization

- Executable libraries
- Bits
- ROP Gadgets

- Easily available with 0 bits EffH
- Difficult with 8 bits EffH

Protecting only some libraries does not mitigate attacks
Memory Disclosure Adversary - Fine Grain

Independent Library Randomization

- Easily available with 0 bits EffH
- Difficult with 8 bits EffH

Protecting only some libraries does not mitigate attacks
Conclusions on Memory Randomization

• Static ASLR does not provide effective defense against adversaries

• PIE ASLR and independent library randomization improve EffH

• Sophisticated adversaries can overcome more advanced randomization techniques
  – Memory disclosure adversary can overcome PIE ASLR and independent library randomization

• Minimum entropy often more important than mean or max
Summary

• Effective Entropy metric for memory randomization security
  – Quantitative
  – Comparable between techniques
  – Provides insight into adversary difficulty

• Fundamental weaknesses in randomization techniques

• Raise minimum entropy and limit connectivity