Effective Entropy for Memory Randomization Defenses

William Herlands, Thomas Hobson, Paula Donovan

7th Workshop on Cyber Security Experimentation and Test

18 August 2014

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

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)

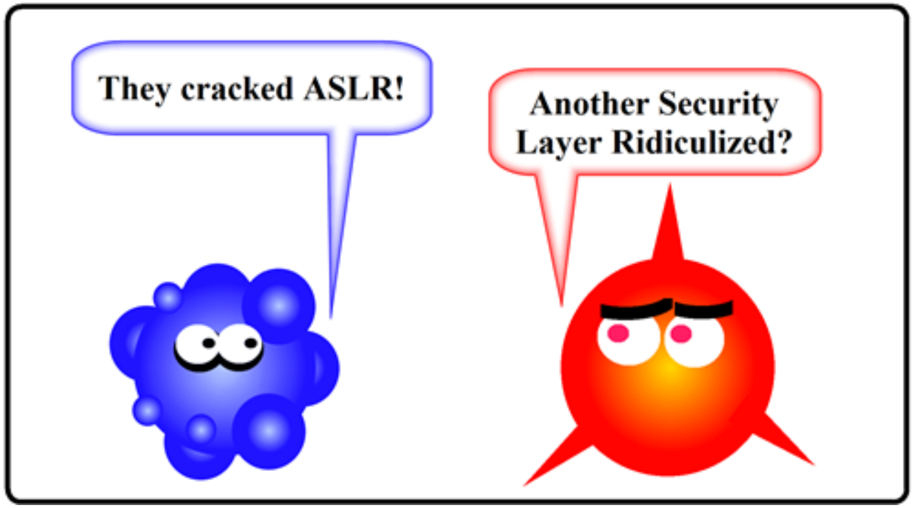


Image Reference: Didier Stevens, yaisc.com

Metric Requirements

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

Exploits	Entropy
Pro	**Pro**
• "Real life," holistic test against adversary technology	• Quantitative, information theoretic • Easy to compare
Con	**Con**
• Anecdotal • Not comparable • Biased towards existing exploits	• Does not consider threat models • Not holistic

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

ASLR entropy improvements

Entropy (in bits) by region	Windows 7		Windows 8		
	32-bit	64-bit	32-bit	64-bit	64-bit (HE)
Bottom-up allocations (opt-in)	0	0	8	8	24
Stacks	14	14	17	17	33
Heaps	5	5	8	8	24
Top-down allocations (opt-in)	0	0	8	17	17
PEBs/TEBs	4	4	8	17	17
EXE images	8	8	8	17*	17*
DLL images	8	8	8	19*	19*
Non-ASLR DLL images (opt-in)	0	0	8	8	24

* 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

Slide reference: Johnson, K., Miller, M. Microsoft Security Engineering Center. Exploit Mitigation Improvements in Windows 8. Black Hat USA 2012

Metric Requirements

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

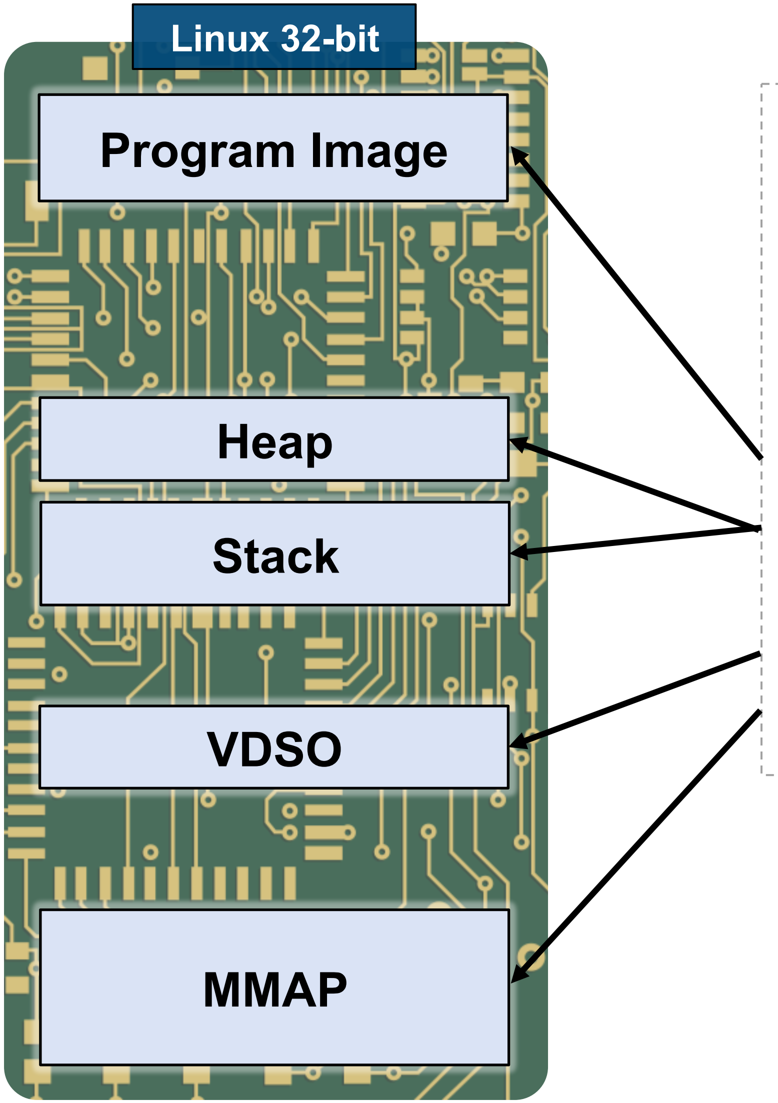
Exploits	Entropy
Pro	**Pro**
• "Real life," holistic test against adversary technology	• Quantitative, information theoretic • Easy to compare
Con	**Con**
• Anecdotal • Not comparable • Biased towards existing exploits	• Does not consider threat models • Not holistic

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

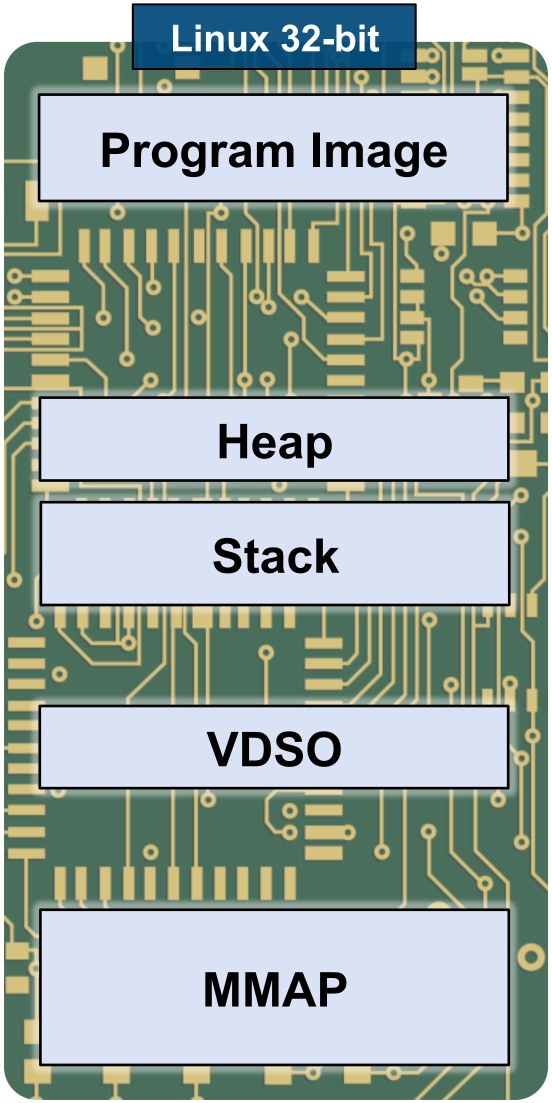


- **Multiple sections required to run a program:**
  - **Code to run ("Program Image")**
  - **Variables used in execution ("Heap" and "Stack")**
  - **Kernel functions ("VDSO")**
  - **Libraries ("MMAP")**

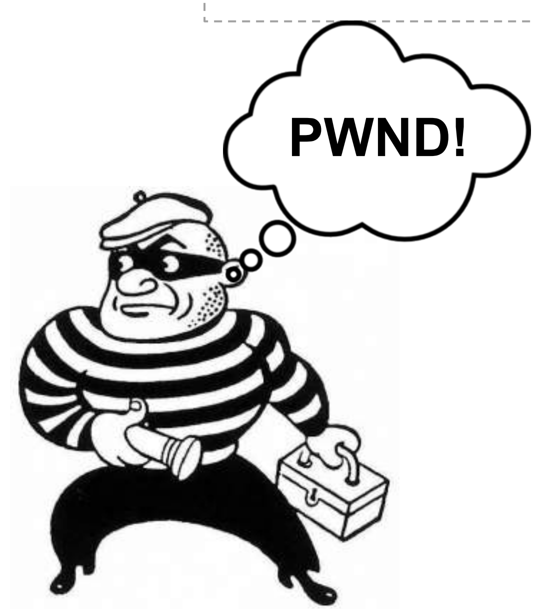
MMAP: Memory Map
VDSO: Virtual Dynamically-linked Shared Objects

User Memory Layout

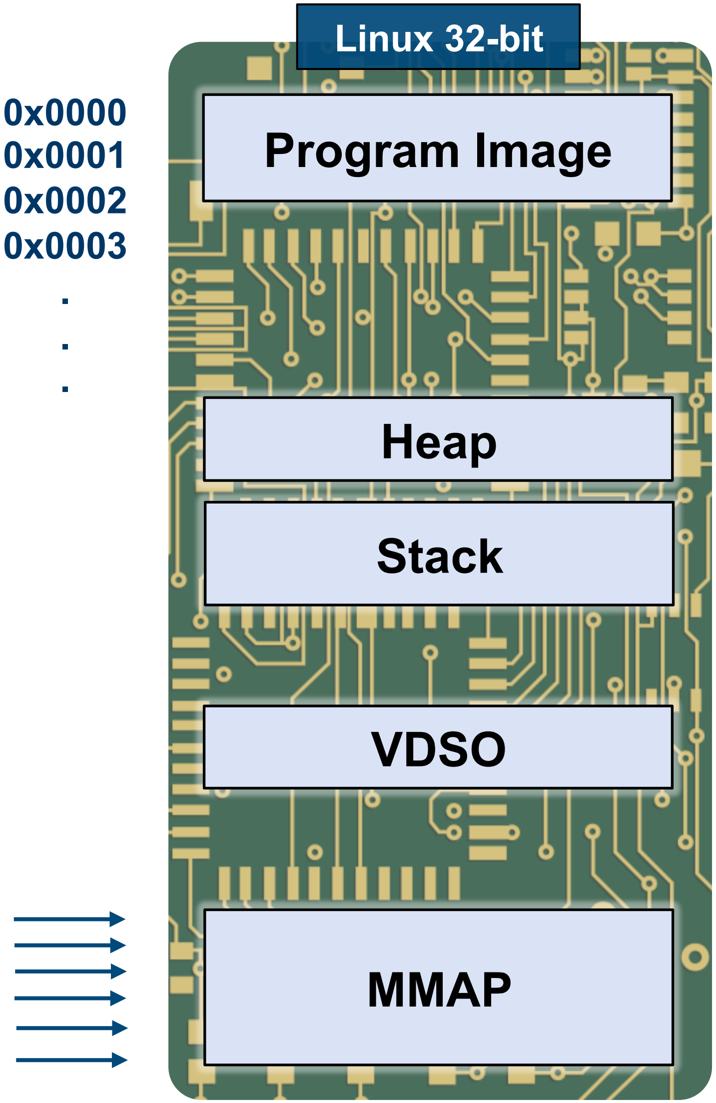
0x0000
0x0001
0x0002
0x0003
.
.
.
.



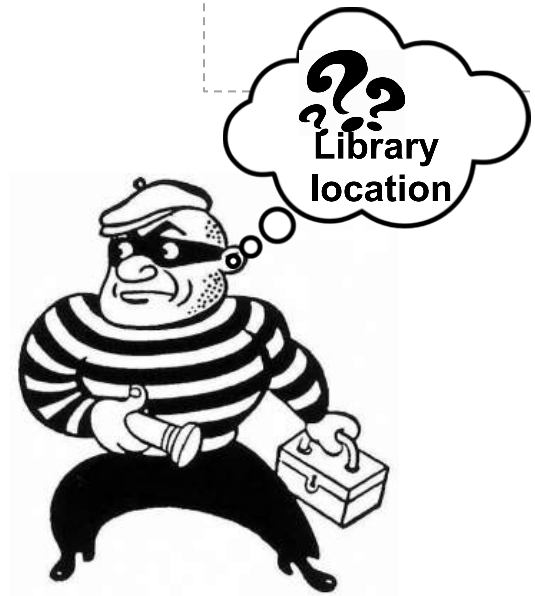
- In a static layout a variety of attacks are possible since an adversary can trivially know the location of objects in memory



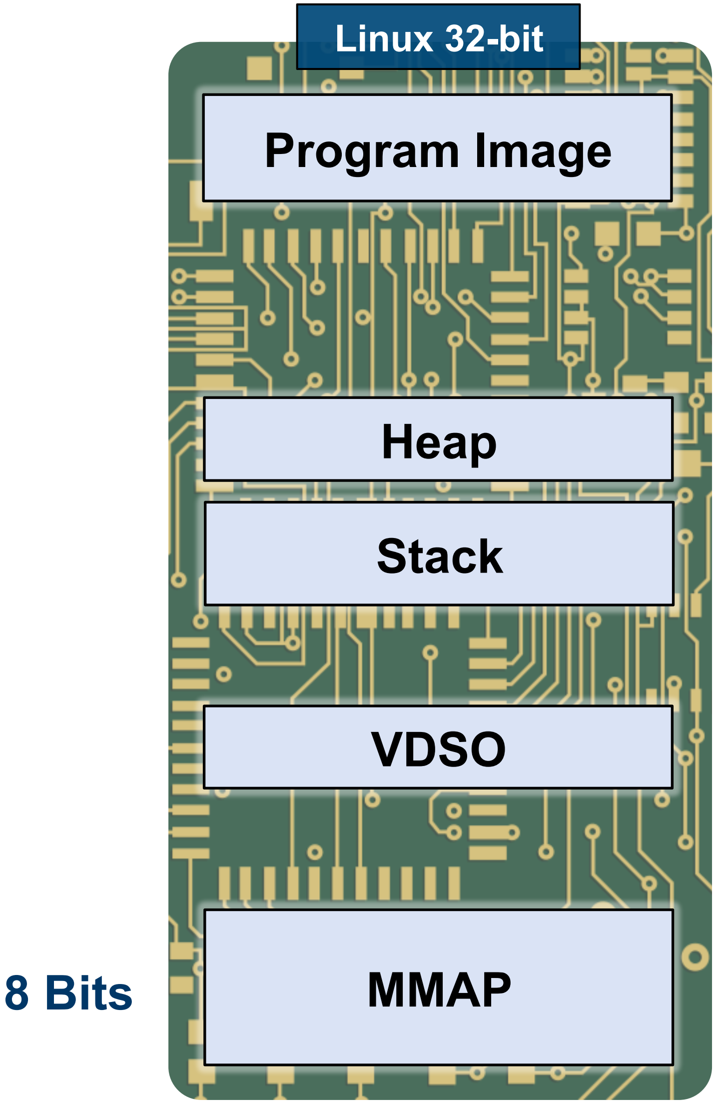
Entropy in User Memory Layout



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



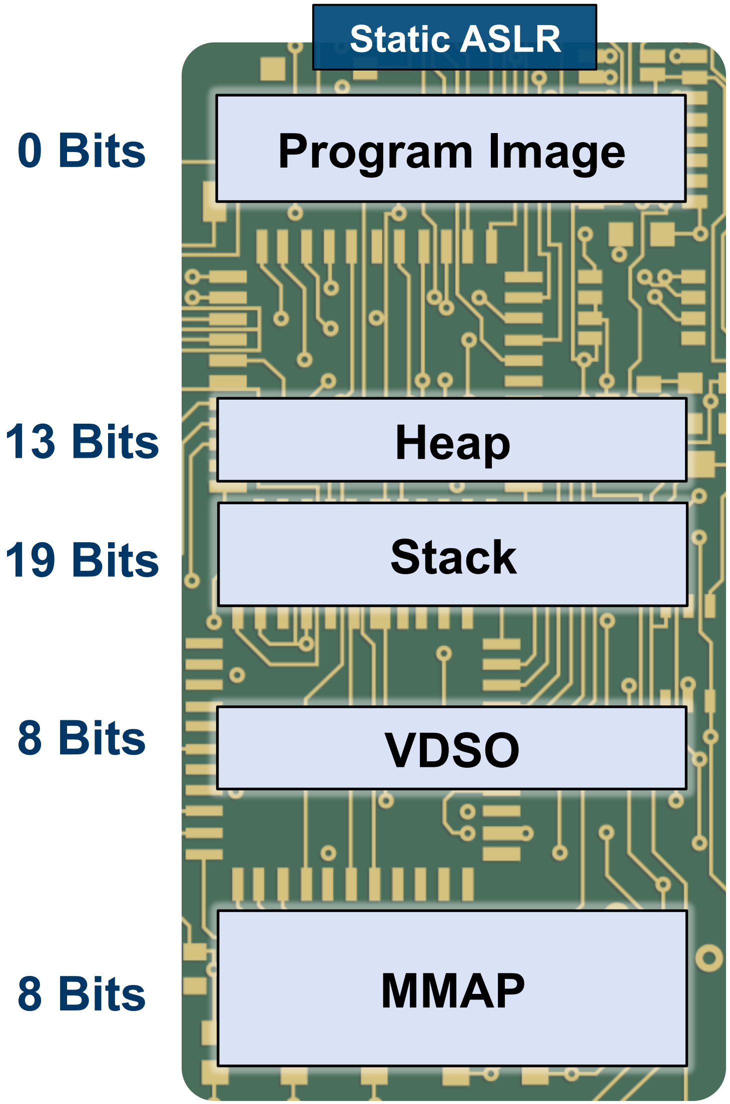
Entropy in User Memory Layout



- 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

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log p(x_i)$$

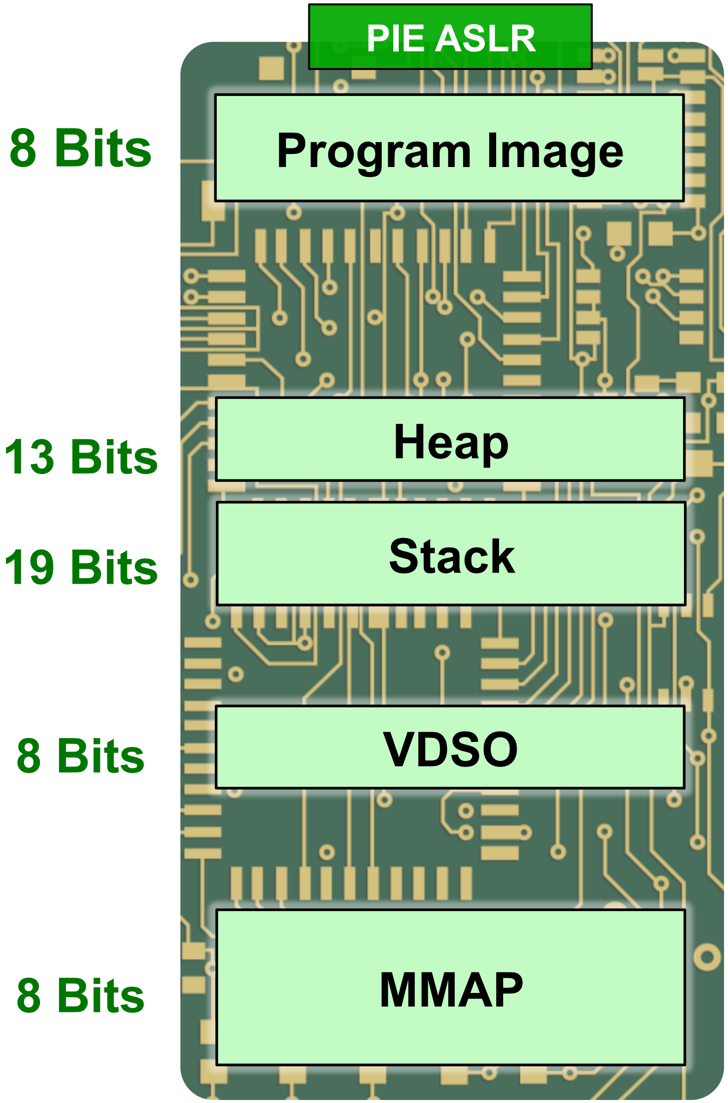
Static Address Space Layout Randomization (ASLR)



- 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

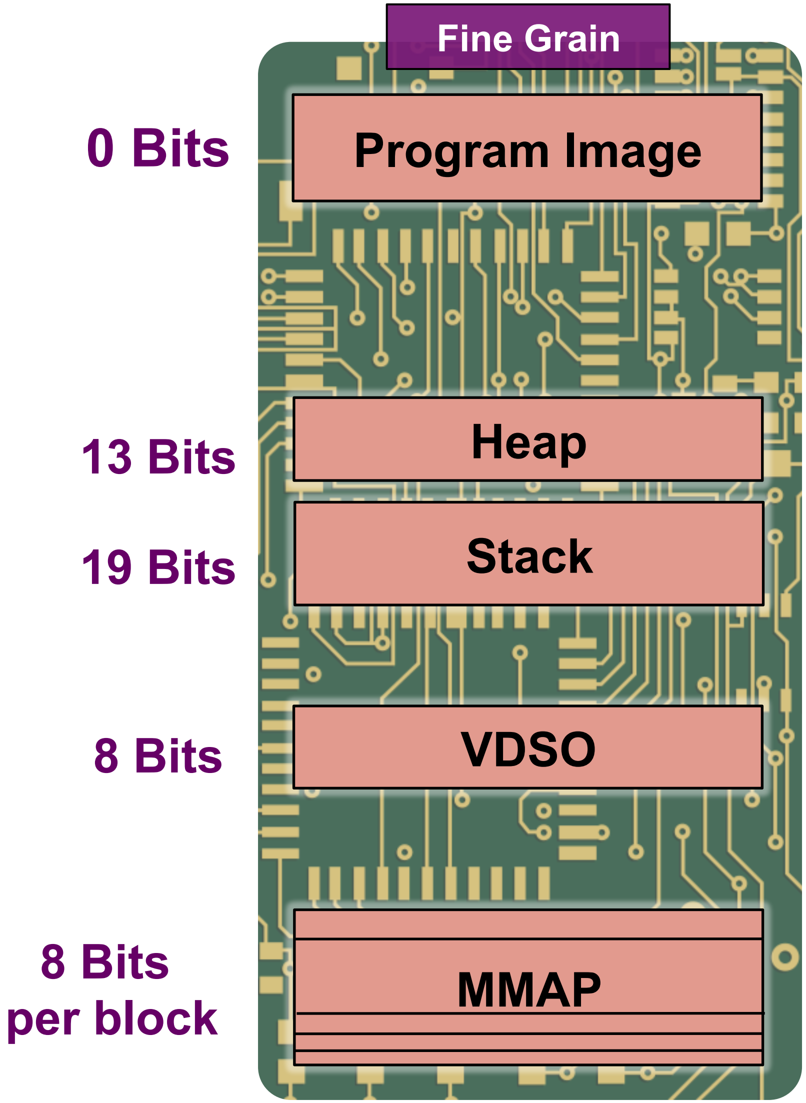
PIE ASLR

- **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

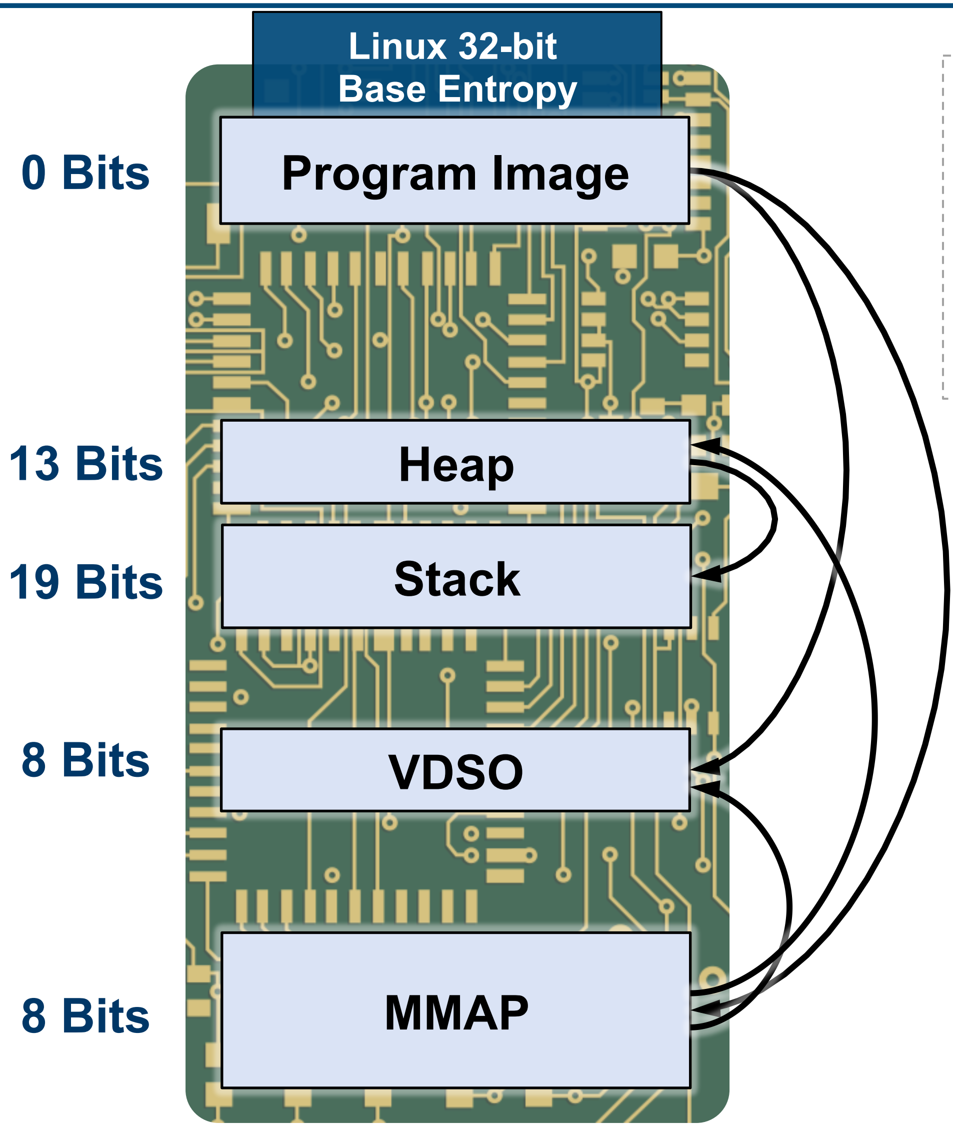
- Fine grain randomization
  - Randomize smaller blocks, not only section base addresses
  - E.g. Independent library randomization
  - “Address Space Layout Permutation (ASLP): Towards Fine-Grained Randomization of Commodity Software”, Kil et al.



Outline

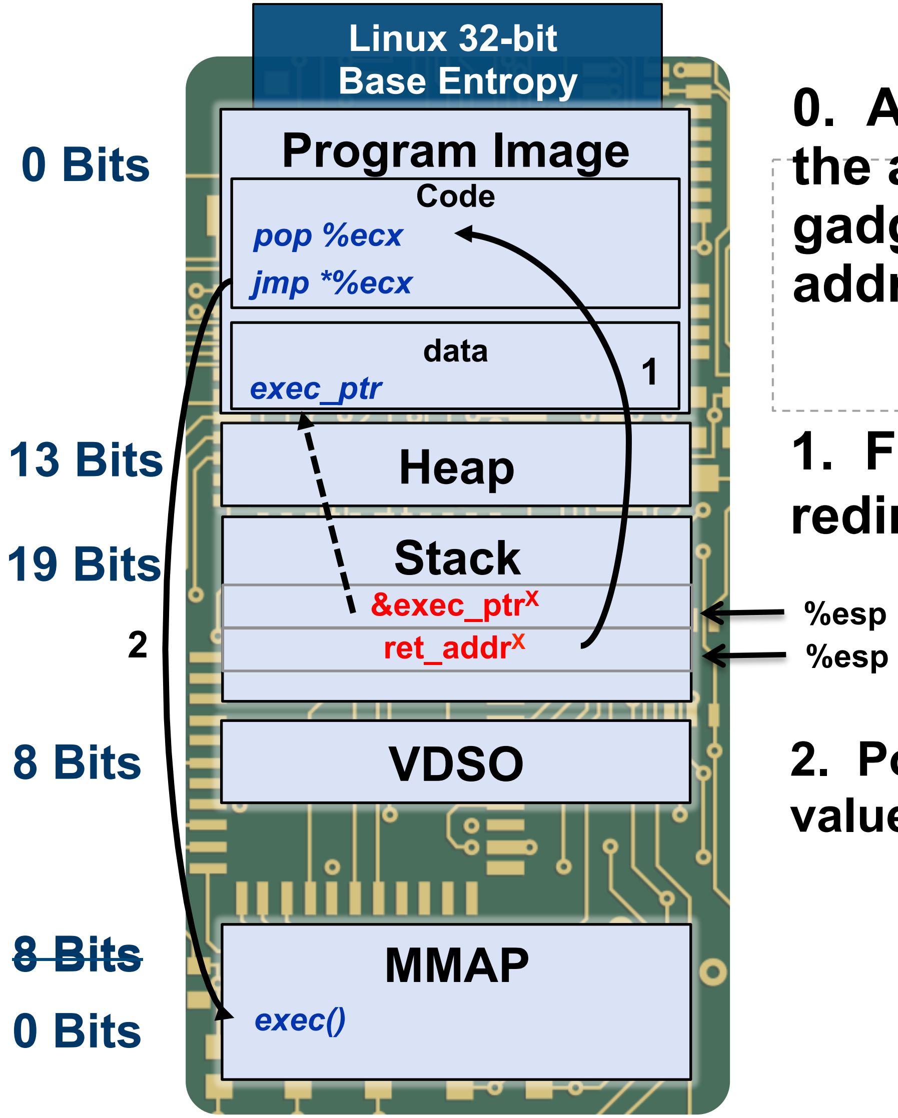
- Background on Memory Randomization
- ➡ Effective Entropy
- Evaluation

Connections in User Memory Layout



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

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

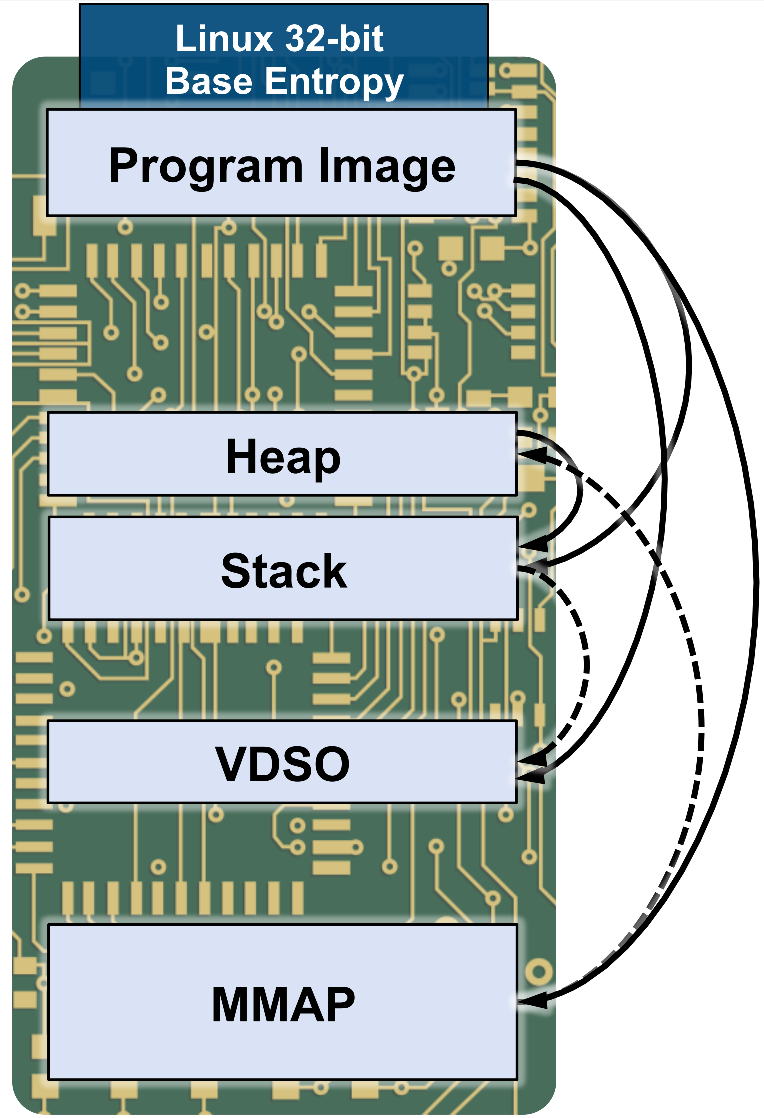
0 Bits

~~13 Bits~~ ? Bits

~~19 Bits~~ ? Bits

~~8 Bits~~ ? Bits

~~8 Bits~~ ? Bits



Connections in User Memory Layout

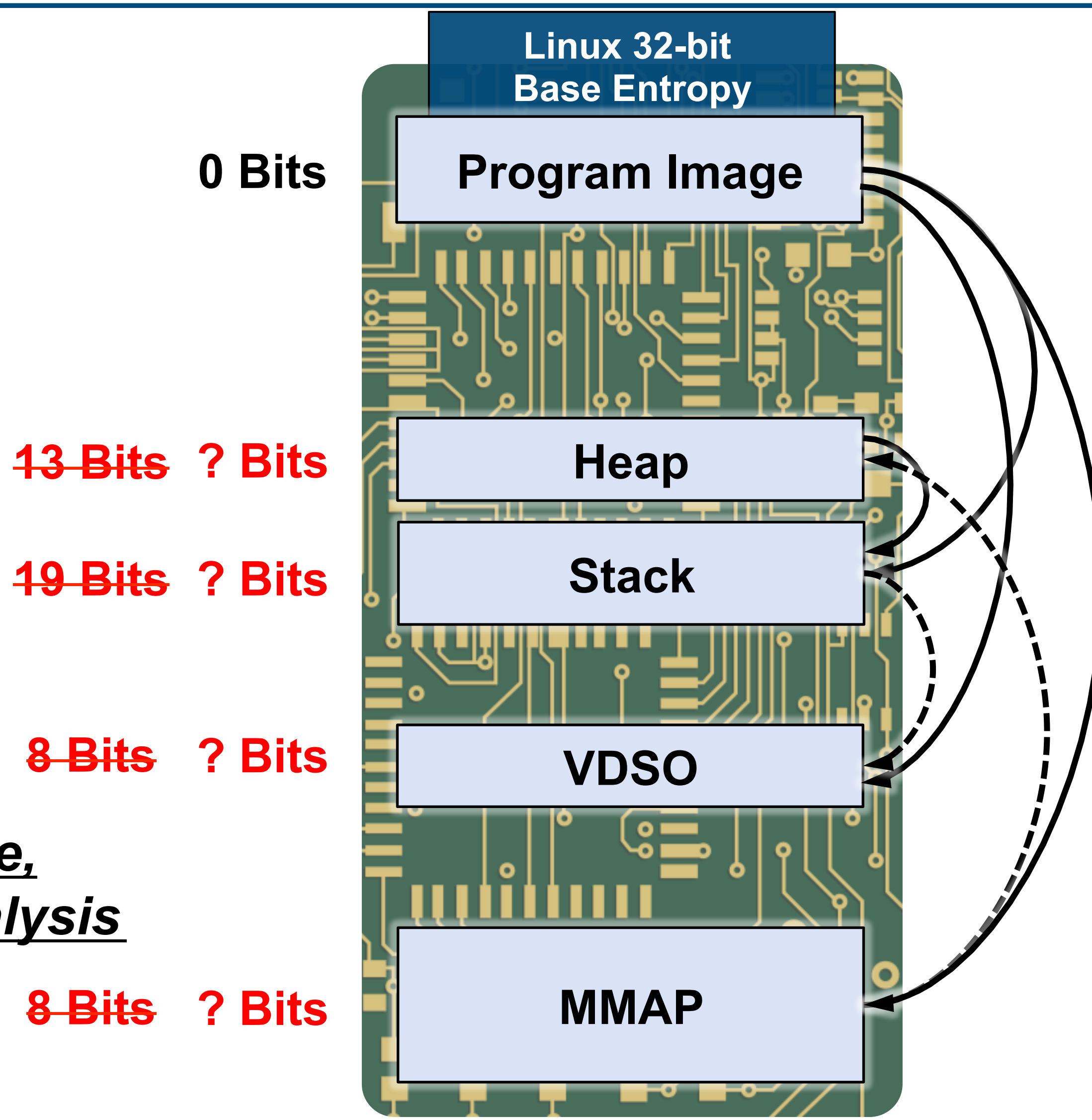
— Absolute connections

- Read-only pointers
- Direct jumps

--- Dynamic connections

- Writable pointers
- Indirect branches

<u>Difficult to determine, requires runtime analysis</u>



Identifying Dynamic Pointers

Run deterministic execution path twice:

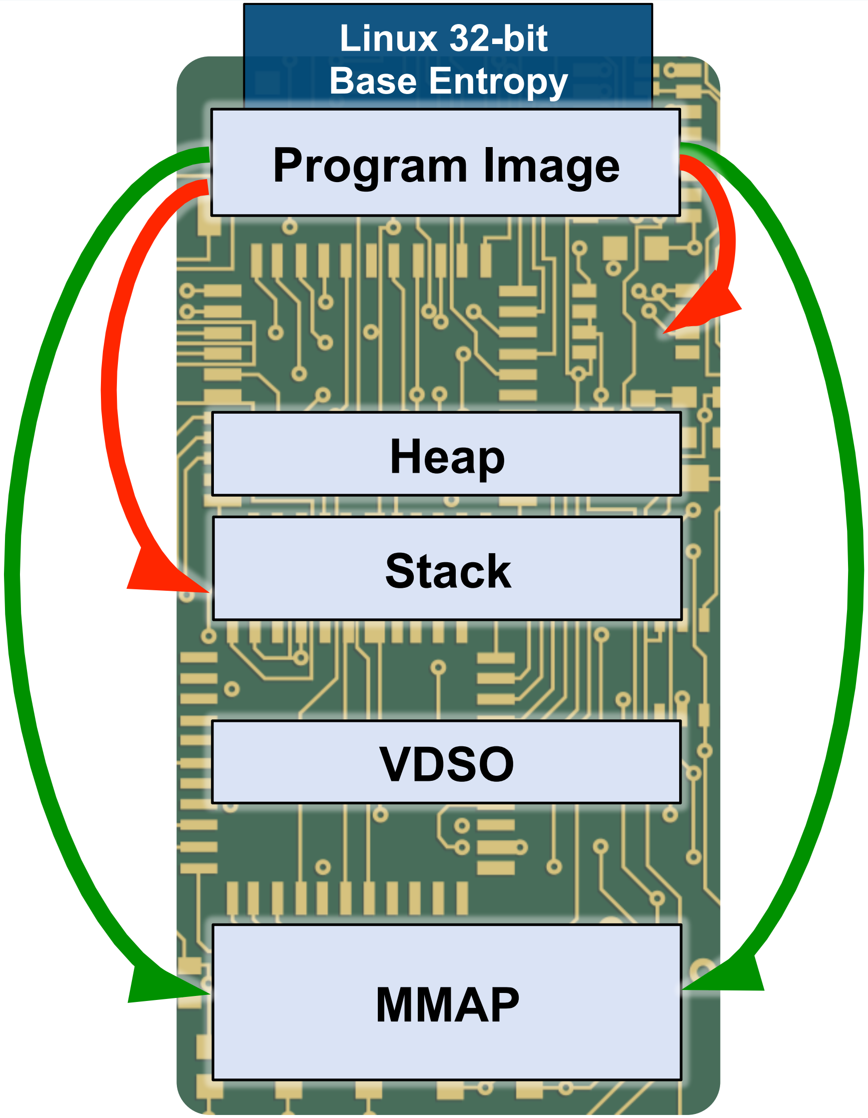
Examine writeable memory at every control flow statement

↓

Eliminate inconsistencies

↓

Identify dynamic pointers



Effective Entropy (EffH)

$$EffH_s = min \begin{cases} h_s \\ min(H^x_{conn}) \\ min(H^p_{conn}) + min(H^x) \end{cases}$$

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

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

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

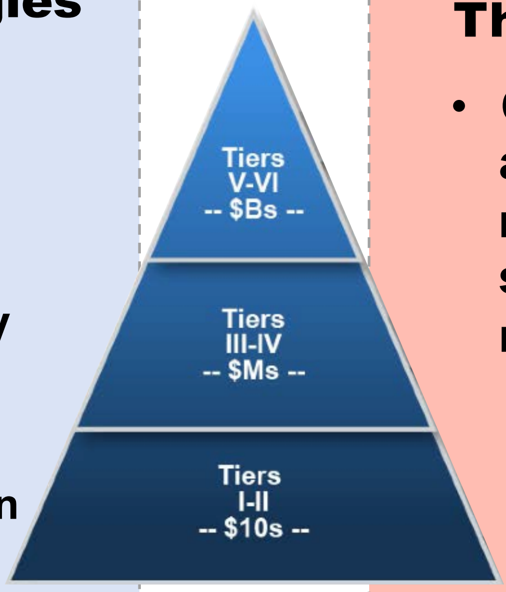
- Background on Memory Randomization
- Effective Entropy
- Evaluation

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

Image Reference: Defense Science Board, Jan 2013: Resilient Military Systems and the Advanced Cyber Threat

Threat Model

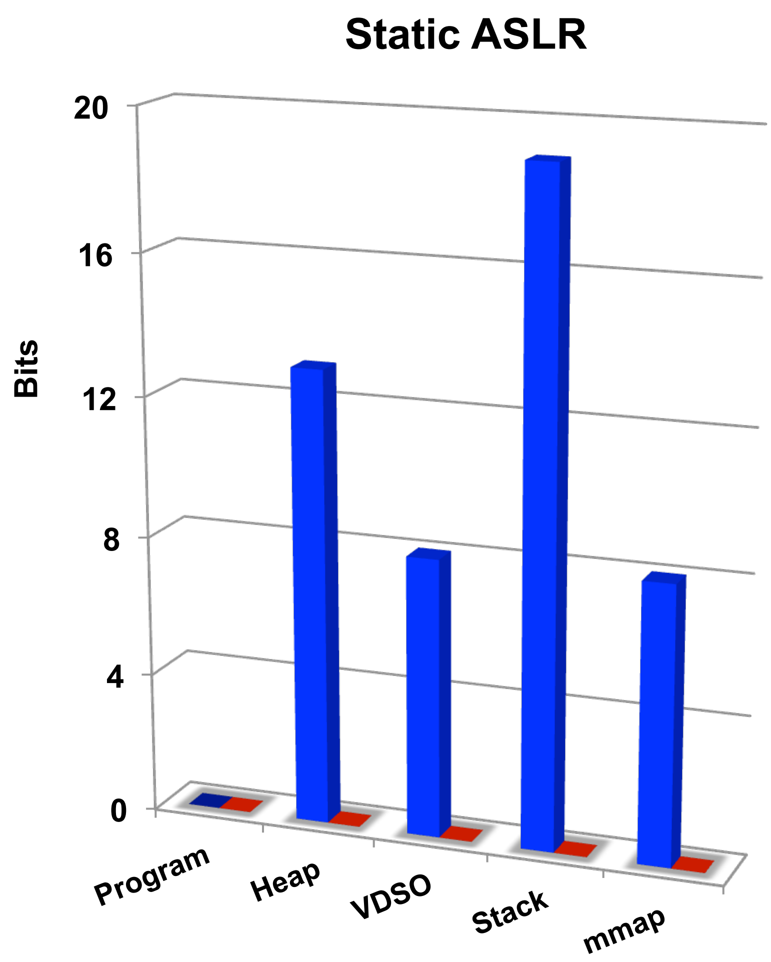
- **Moderate Adversary**
  - **Control flow hijacking vulnerability**
  - **Modern exploitation methods including Return Oriented Programming (ROP)**
- **Memory Disclosure Adversary**
  - **Control flow hijacking vulnerability**
  - **Modern exploitation methods including ROP**
  - ***Memory disclosure vulnerability that reveals location of one memory section***

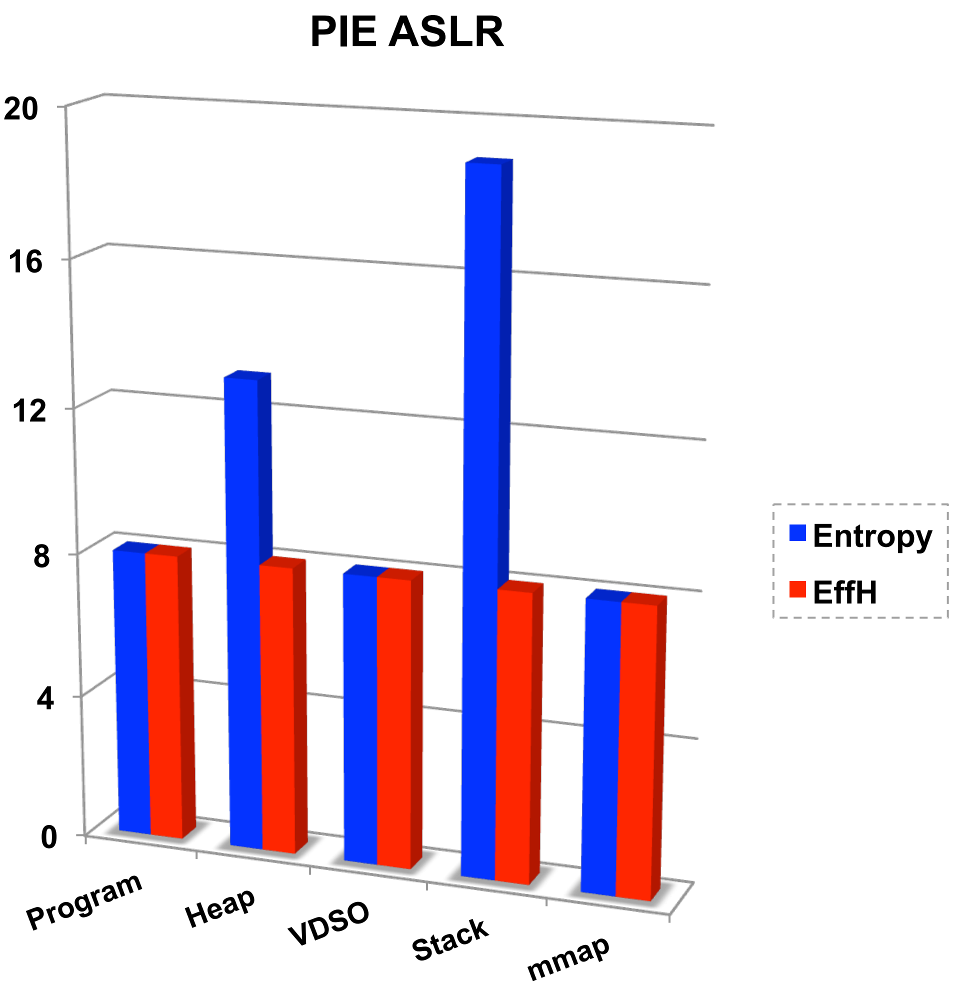
Return Oriented Programming

- **Use snippets of executable code called "ROP gadgets"**
- **Combine gadgets to create a custom exploit**

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

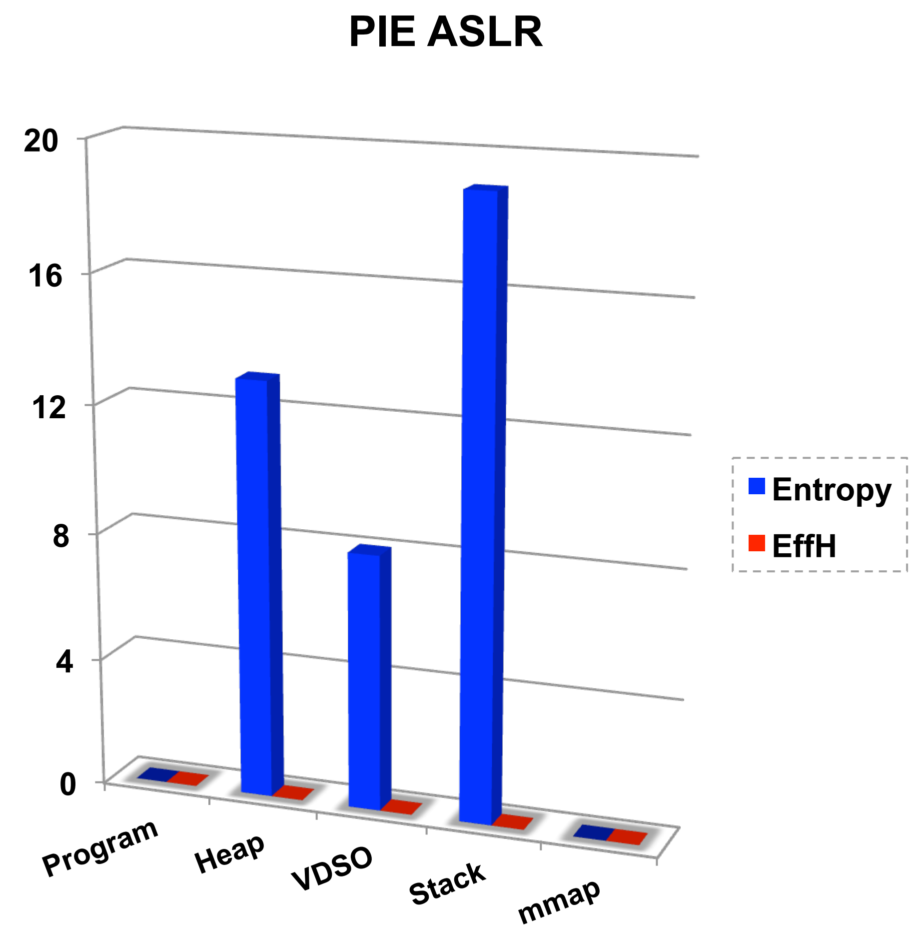
Moderate Adversary - ASLR





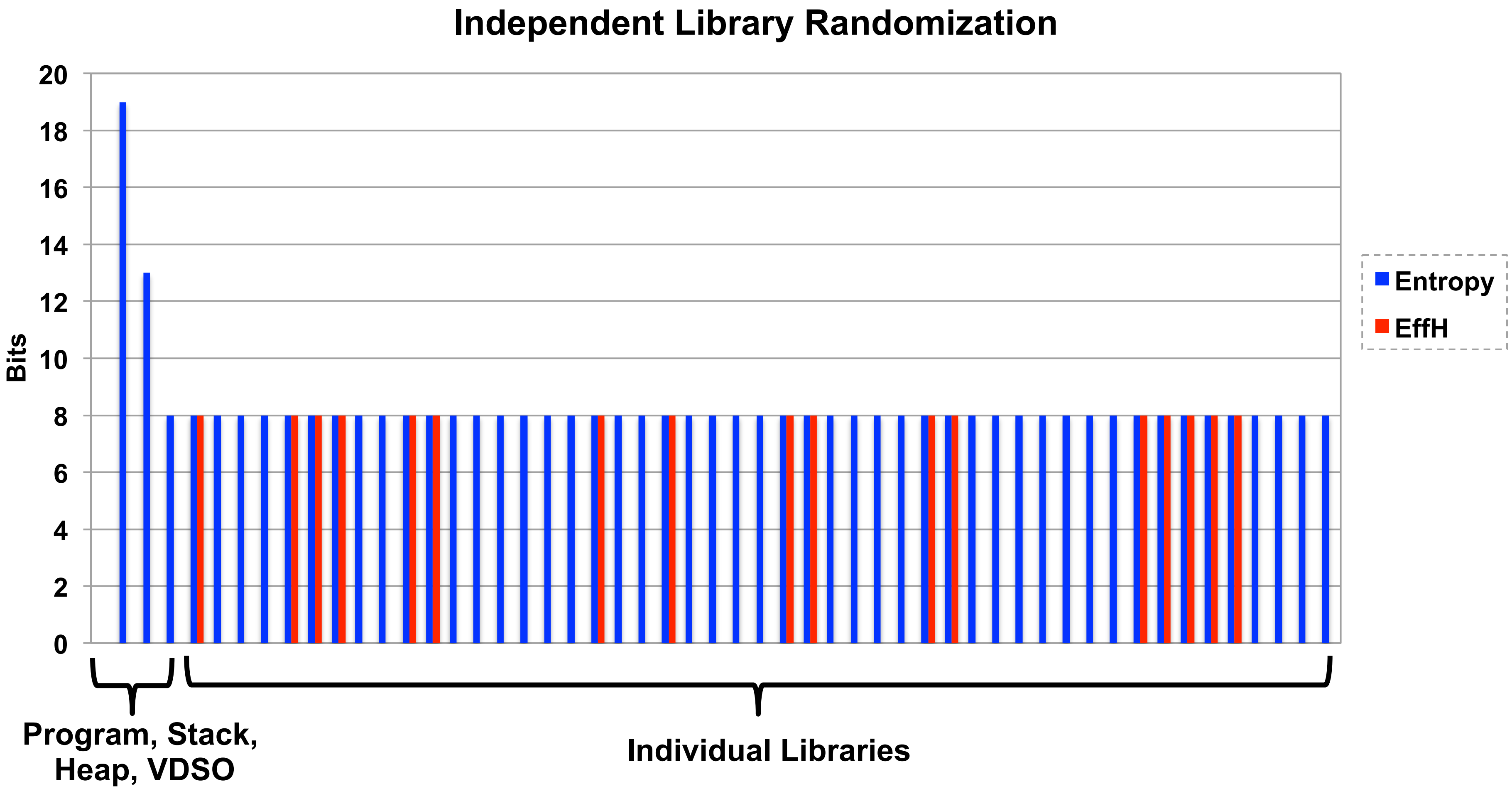
Static ASLR provides zero bits of EffH to Moderate Adversary

Memory Disclosure Adversary – PIE ASLR



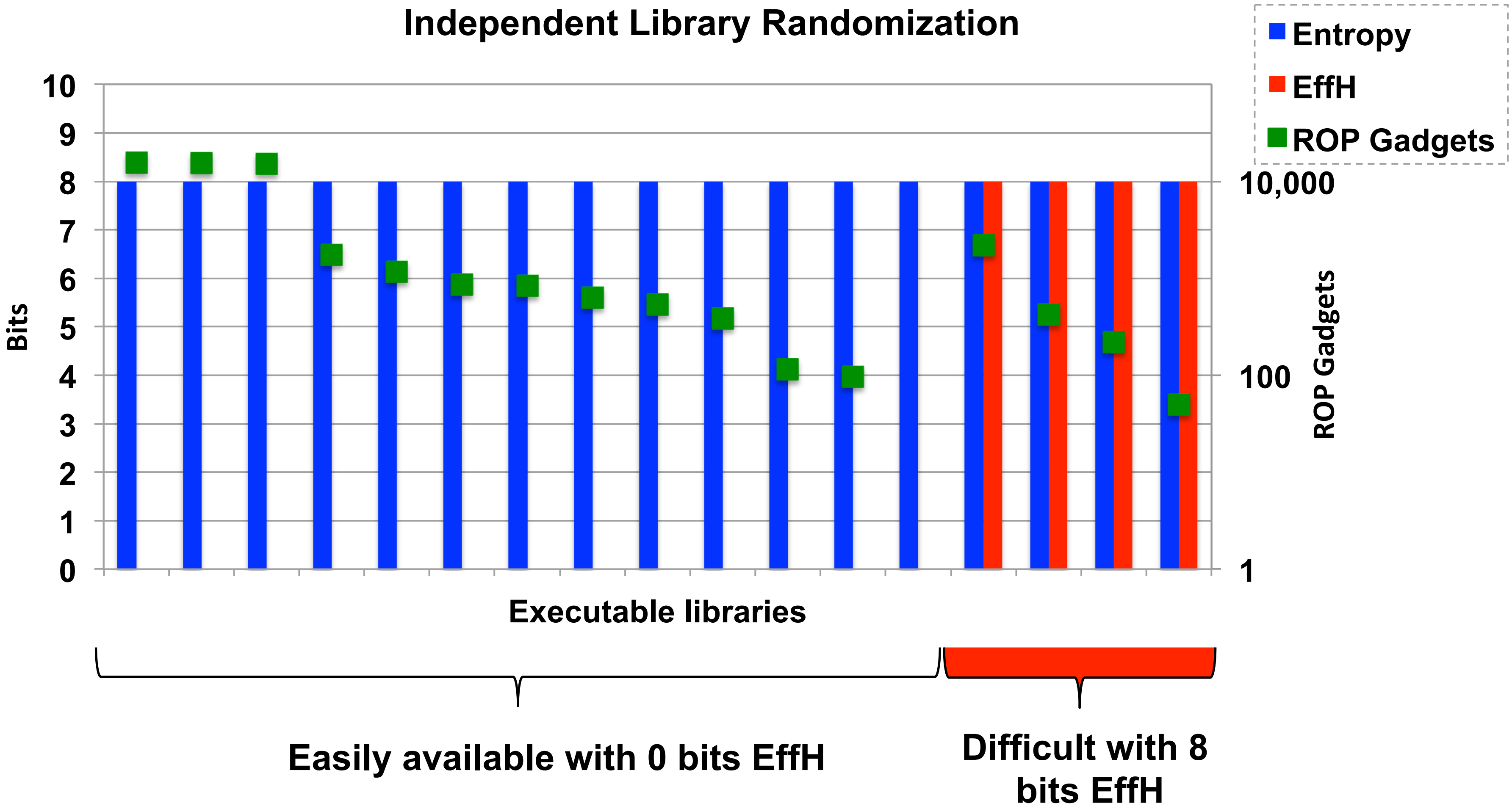
PIE ASLR provides zero bits of EffH to Adversary disclosing Program Image

Memory Disclosure Adversary - Fine Grain



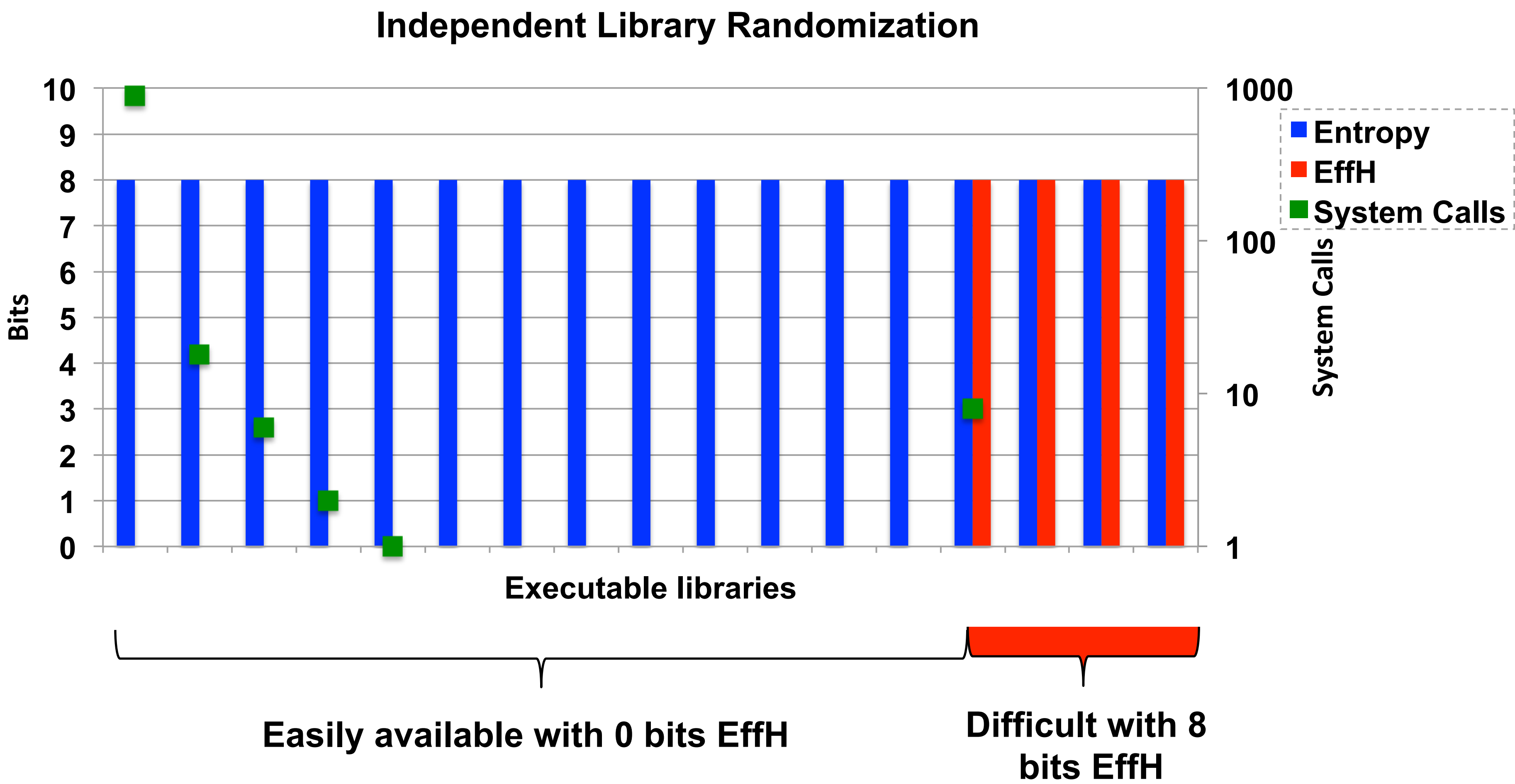
Fine Grain provides 8 bits of EffH for some but not all libraries

Memory Disclosure Adversary - Fine Grain



Protecting only some libraries does not mitigate attacks

Memory Disclosure Adversary - Fine Grain



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**