NC STATE UNIVERSITY Computer Science

HideM: Protecting the Contents of Userspace Memory in the Face of Disclosure Vulnerabilities

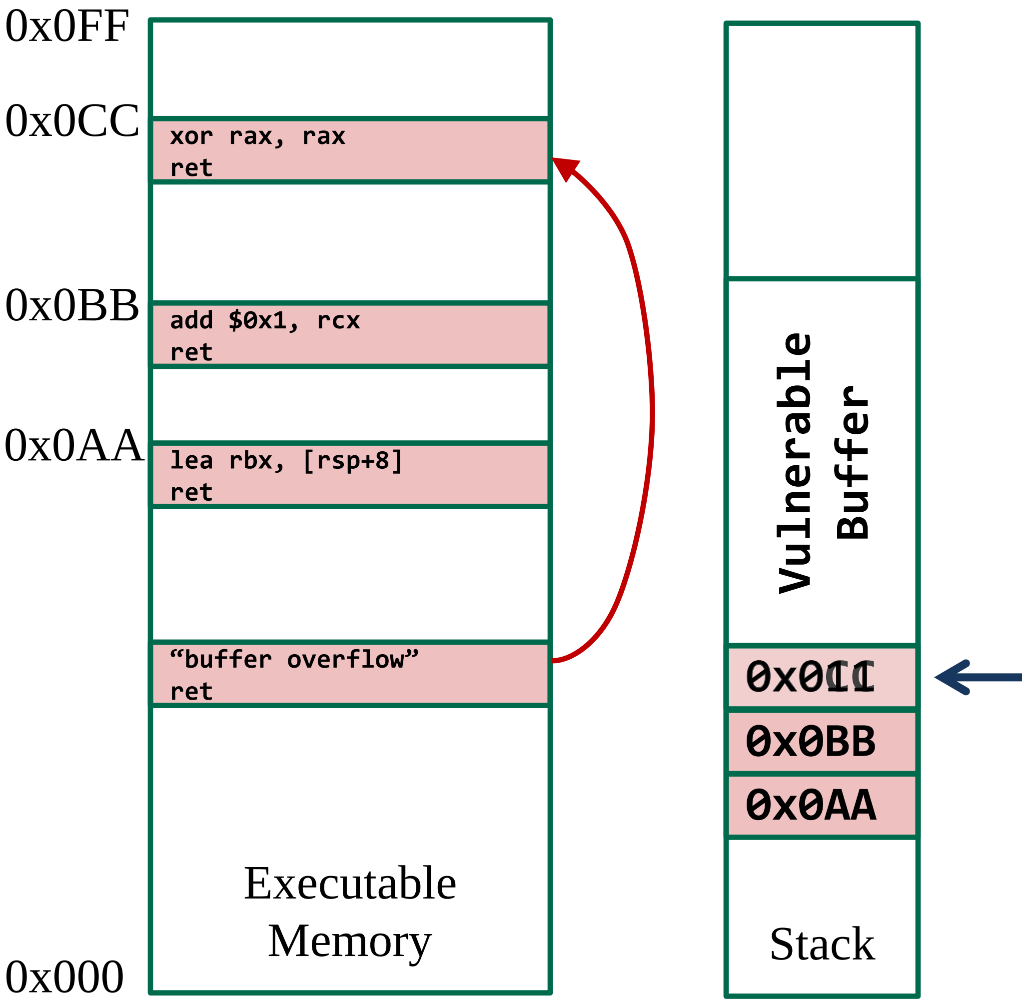
Jason Gionta, William Enck,
Peng Ning

JIT-ROP

Two Attack Categories

- Injection Attacks
  - Code Integrity
  - Data Execution Prevention
- Code-Reuse Attacks

Code-Reuse Attacks – Simple ROP

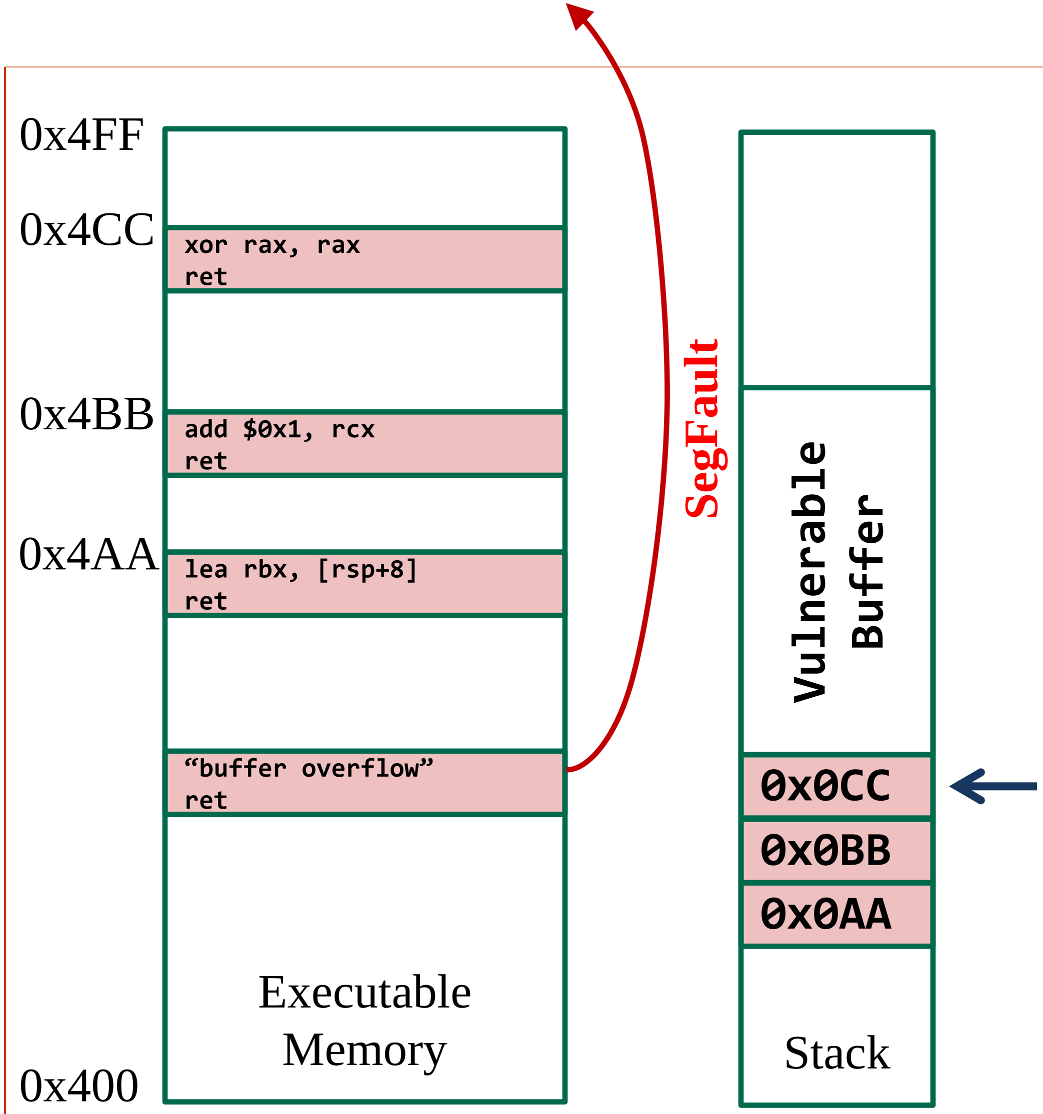


Overwritten Return Address

What an attacker knows?

- Buffer overflow
- Static binary load location
- Static location of gadgets

Code-Reuse Attacks – Break ROP



0x4FF

0x4CC

```
xor rax, rax
ret
```

0x4BB

```
add $0x1, rcx
ret
```

0x4AA

```
lea rbx, [rsp+8]
ret
```

```
“buffer overflow”
ret
```

Executable
Memory

0x400

SegFault

Vulnerable
Buffer

0x0CC

0x0BB

0x0AA

Stack

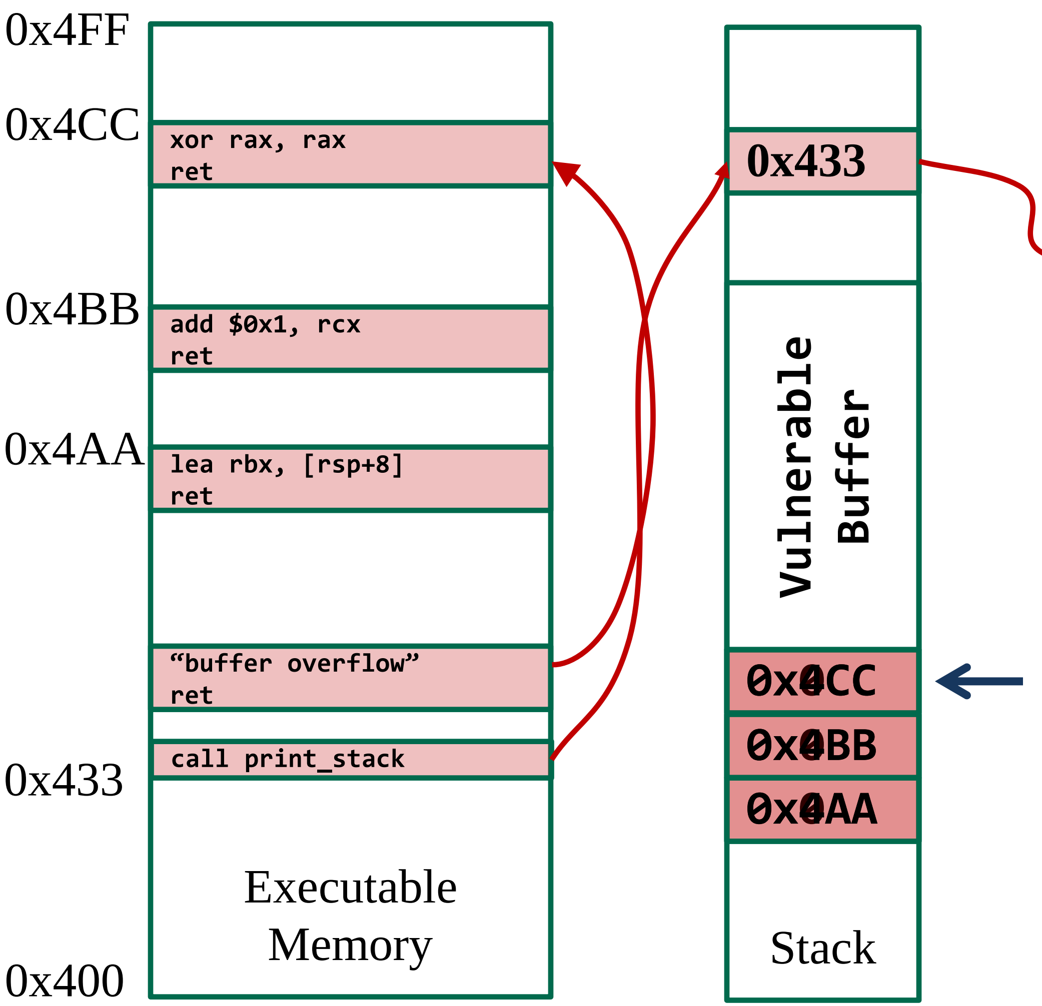
Overwritten
Return Address

What an attacker knows?

- Buffer overflow
- ~~Static binary load location~~
- Static location of gadgets

Protection: ASLR

Code-Reuse Attacks – ASLR Bypass ROP



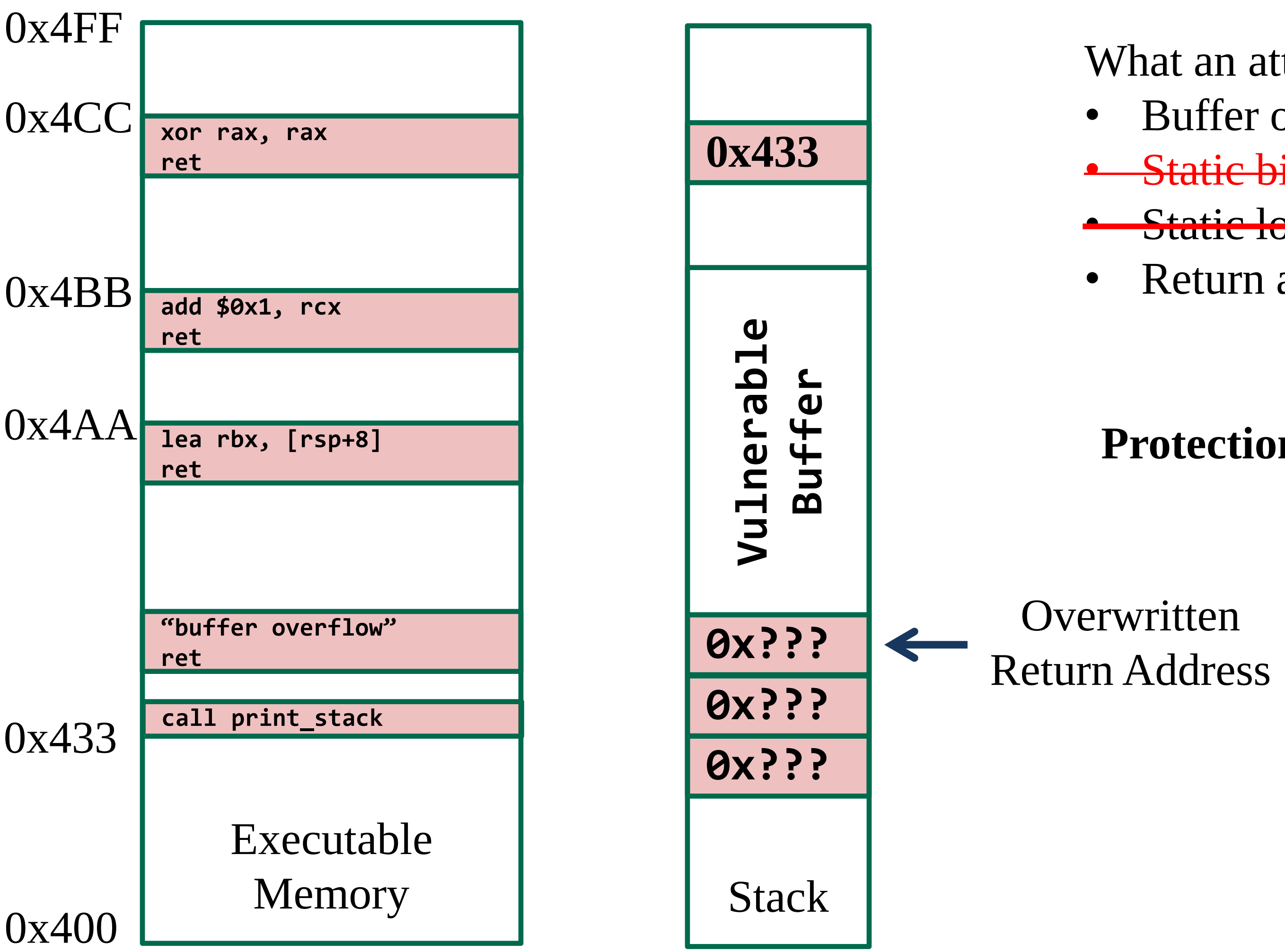
What an attacker knows?

- Buffer overflow
- ~~Static binary load location~~
- Static location of gadgets
- Return address leak

Protection: ASLR

Overwritten Return Address

Code-Reuse Attacks – Break ROP (Again)



0x4FF

0x4CC
```
xor rax, rax
ret
```

0x4BB
```
add $0x1, rcx
ret
```

0x4AA
```
lea rbx, [rsp+8]
ret
```

```
“buffer overflow”
ret
```

0x433
```
call print_stack
```

0x400

Executable Memory

0x433

Vulnerable Buffer

0x???

0x???

0x???

Stack

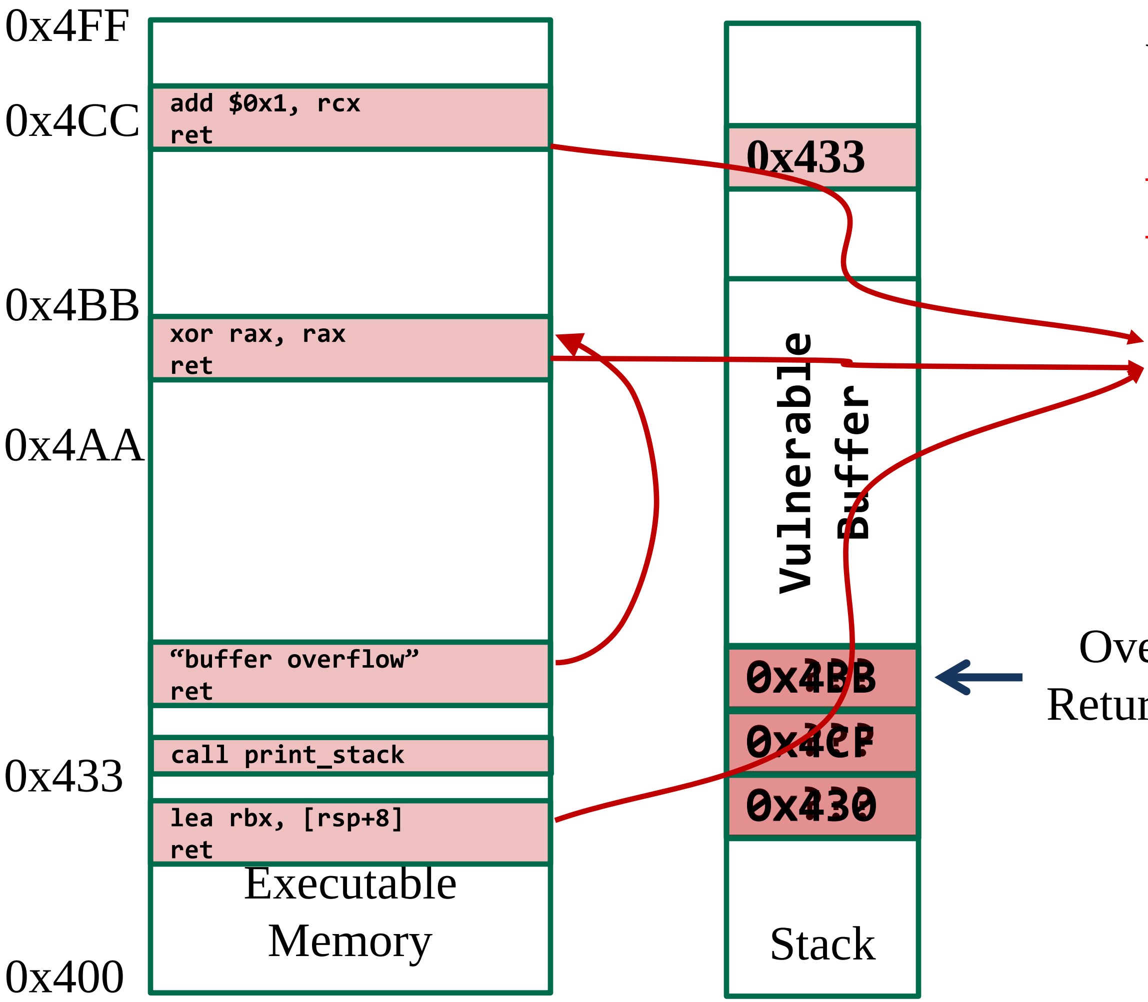
Overwritten Return Address

What an attacker knows?

- Buffer overflow
- ~~Static binary load location~~
- ~~Static location of gadgets~~
- Return address leak

Protection: Fine-Grained ASLR

Code-Reuse Attacks – ROP + Disclosure



What an attacker knows?

- Buffer overflow
- ~~Static binary load location~~
- ~~Static location of gadgets~~
- Return address leak
- Code Disclosure

Protection: Fine-Grained ASLR

Overwritten Return Address

Memory Disclosure Vulnerabilities

- Leak raw memory contents
- Used for bypassing modern protections
  - ASLR - Pwn2Own2013, Pwn2Own2014
  - Fine-grained ASLR - Just-In-Time Code Reuse [Snow et al. 2013]
  - CFI - Out-of-Control: Overcoming Control Flow Integrity [Gotkas et al. 2014]
- Disclosure Protections
  - XnR – [Backes et al. 2014]
    - Limit code reads so small set of memory
    - Does not handle legitimate reads
    - Heuristic based detection
- Observation: *commodity systems lack of fine-grained read permissions*

HideM: Protect Userspace Code from Disclosure

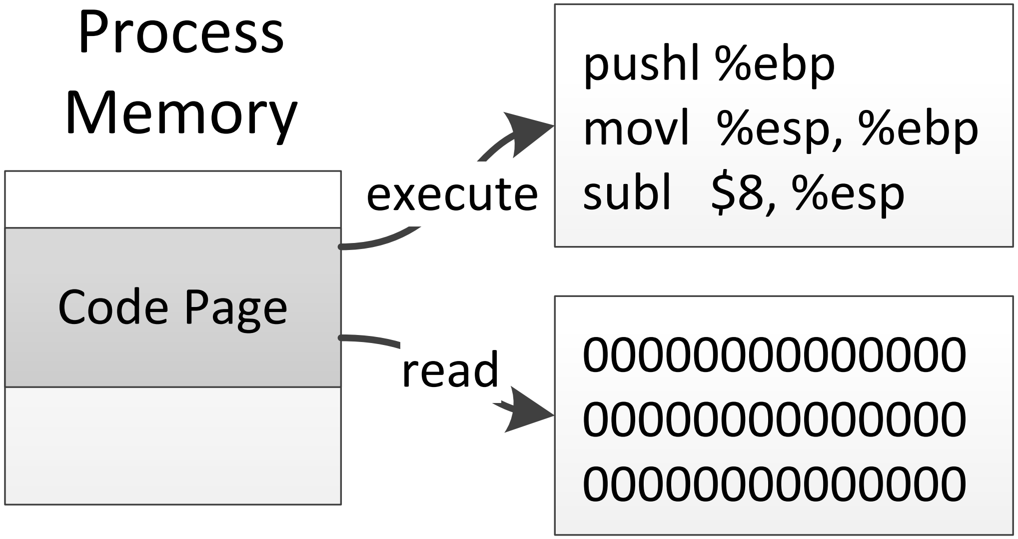
- Assumption: Fine-Grained ASLR Deployed
- Enable fine-grained read permissions on executable memory
  - Prevent the majority of code from being read
- Enforce permissions seamlessly on Commercial-Off-The-Shelf (COTS) binaries
- Target commodity systems to ease adoption
- GOAL: Unreliable exploitation
  - Adversary must guess contents of code

Challenges

- Execute permissions imply read permissions
  - A *present* userspace page can always be read
  - **Solution**: apply *code hiding* to differentiate memory access based on CPU operation
- Executable pages often contain read-only data
  - Allow legitimate reads of executable-pages
  - **Solution**: generate and apply *code reading* policy per executable page
- Protecting COTS binaries without symbols
  - **Solution**: light-weight binary analysis to identify data embedded in code pages

Code Hiding: Primitive

- Enables execute-only permissions on memory
  - Access based on CPU operation
- Based on PaX and advanced rootkit hiding
- Leverage split TLB architecture

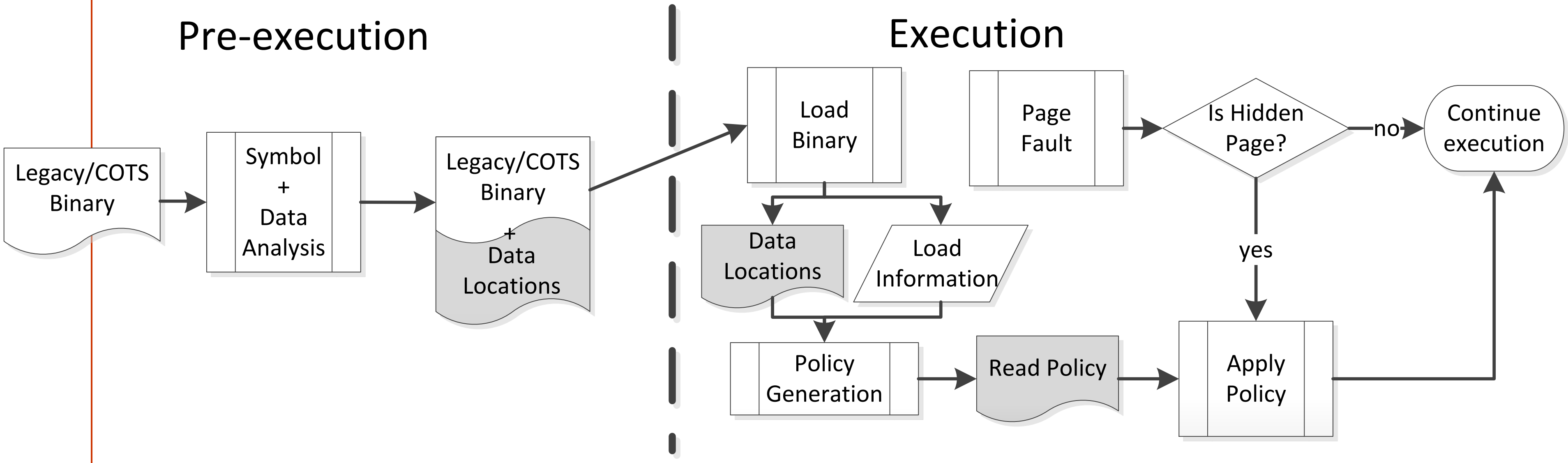


```
pushl %ebp
movl  %esp, %ebp
subl   $8, %esp
```

```
000000000000000
000000000000000
000000000000000
```

Fine-grained Read Permissions

- Generate and enforce code reading policy
  - Identify read data in code prior to execution
  - Embed as part of COTS binary
- Associate binary data locations with load time memory
- Apply code reading policy per page



Identifying Data in Code - Types

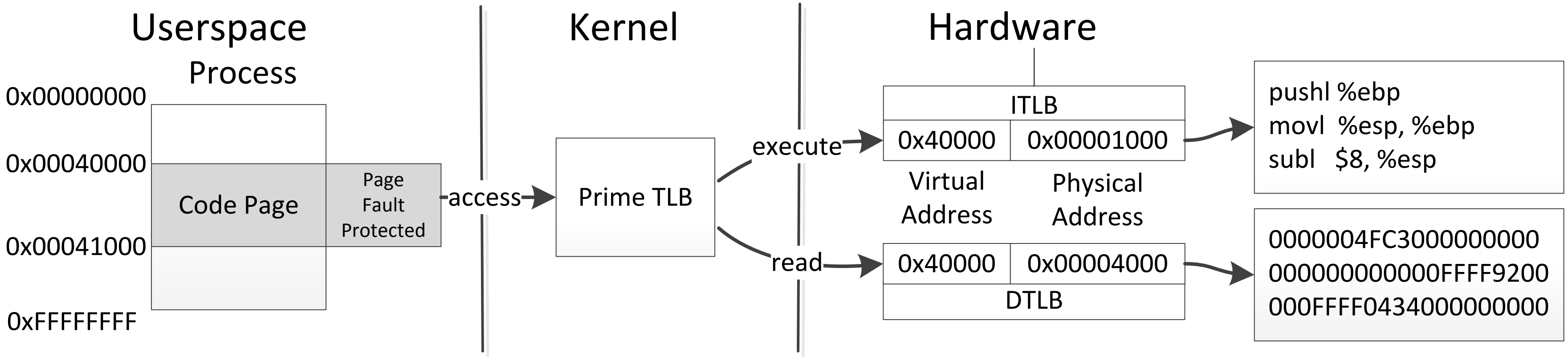
- Two types of data read in code pages
  - DT-1: read-only data never executed
  - DT-2: executed data that needs to be read
- Provide DT-1 and DT-2 ranges with binary

Identifying Data in Code – How to Identify

- DT-1: read-only data never executed
  - Binary structure
  - Recursive disassembly to identify ICF targets
    - Based on Zhang and Sekar [Usenix Sec’13]
    - Identify jump-tables
    - Disassembly errors identify gaps (unknown regions)
- DT-2: executed data that needs to be read
  - Binary analysis – identify DT-2
    - Immediate values that result in a valid code address
    - Instruction pointer relative values

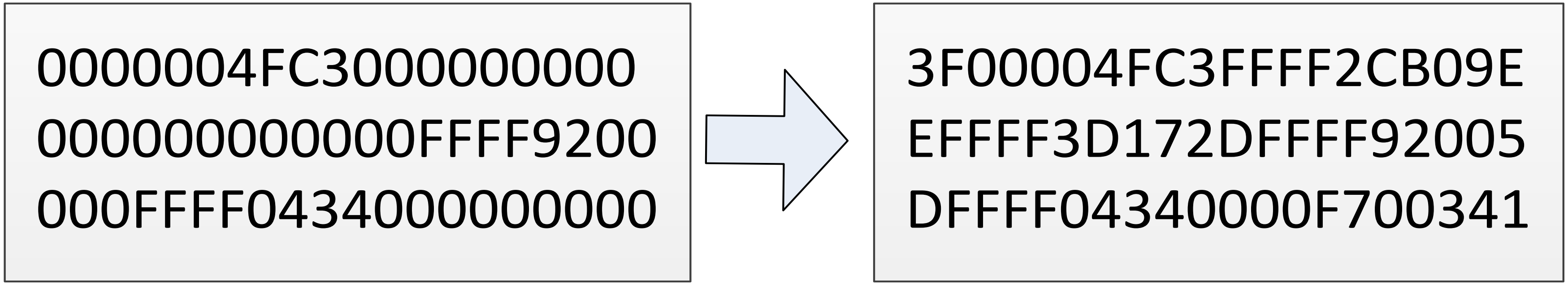
Applying Read Policy

- All HideM protected pages have userspace access denied by default
- Generate shadow read page based on policy
  - First page fault copy DT-1 and DT-2 to read page
  - Remove DT-1 from code page
- Apply shadow page with code hiding
  - Prime TLB



Hardening Against ROP Exploits

- Readable pages contain DT-2 data
- Adversary can build exploits from only DT-2
  - Limited to 4 bytes in length
    - Identifiable by ROP Runtime Protections
- Add noise to readable pages in place of non-read code
  - Mimics DT-2 data



```
0000004FC3000000000
000000000000FFFF9200
000FFFF0434000000000
```

```
3F00004FC3FFFF2CB09E
EFFFF3D172DFFFF92005
DFFFF04340000F700341
```

Empirical Evaluation

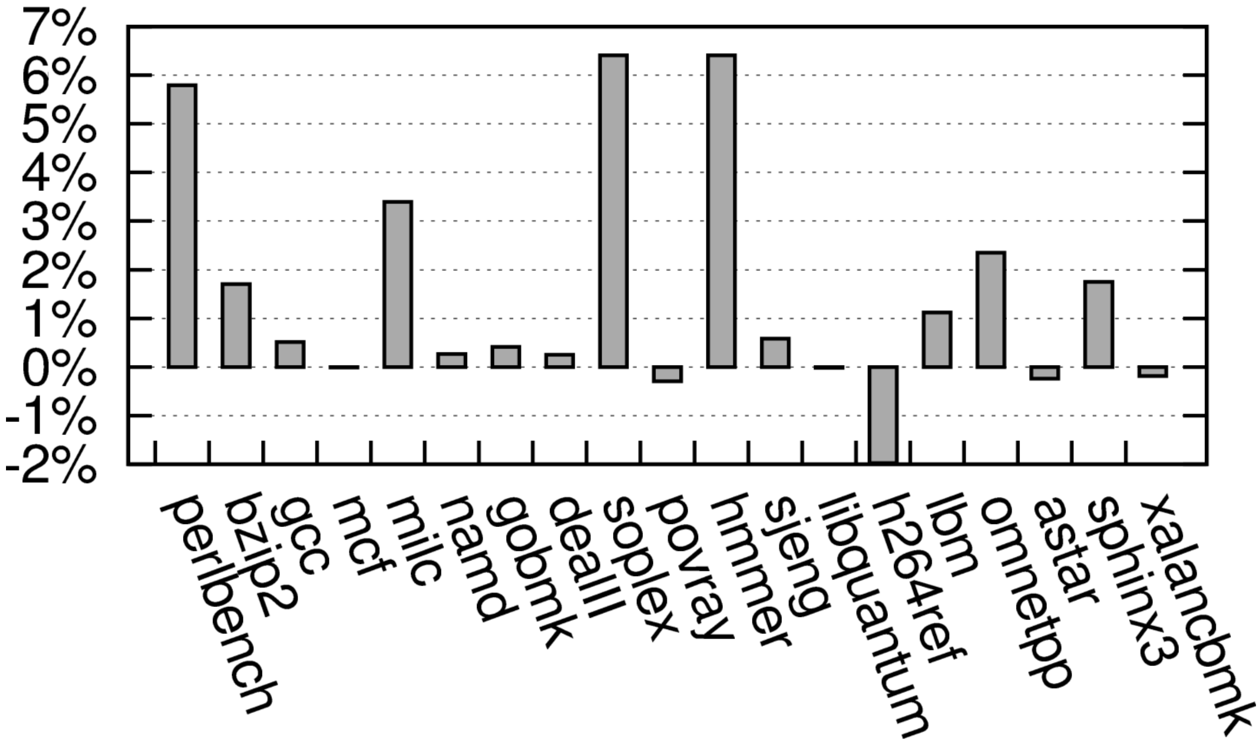
- Implementation of HideM
  - Linux kernel 3.10.12
    - Intel x86 64-bit
  - Dynamic library loading support
    - glib 2.18
- Platform
  - IBM LS22 blade server
    - Two Quad-Core AMD Operton 2384 processors
    - 32GB of RAM
    - 1024 4KB page TLB entries per core
  - Ubuntu 12.04.4 LTS 64-bit

Empirical Evaluation: Application Set

- 28 applications and required shared libraries converted to HideM
  - 442 total binaries converted / 441 MB
  - 13 binaries required manual analysis of data in code
- 9 non-trivial applications
  - Wireshark, dumpcap, gimp, gedit, lynx, python, emacs, lynx, smplayer
- 19 SpecCPU 2006 applications
  - Perlbench, bzip2, gcc, mcf, gobmk, hmmer, sjeng, libquantum, h264ref, omnetop, astar, xalancbmk, milc, namd, dealII, soplex, povray,lbm, sphinx3

Empirical Evaluation: Performance

- Runtime overhead: Percent increase in runtime
  - Maximum: 6.5% increase; Minimum: 2% decrease
  - Average 1.49%, median 0.51%



Empirical Evaluation: Security Evaluation

- Model the probability of exploitation based on knowledge of HideM
  - Adversaries dump memory and search for gadgets
  - Identify unique gadgets required for exploit
  - Choose a location for each unique required gadget
    - Duplicate gadgets at different locations

Empirical Evaluation: Security Evaluation

- Probability of exploitation against HideM
  - Based on “Unordered sampling without replacement”
    - N gadgets for an exploit
    - U_g total unique gadgets
    - U_{vg} number of unique valid gadgets
    - S_{vg} number of valid unique gadgets for a specific gadget
    - S_g total number of gadgets for a specific valid unique gadget

$$\prod_{n=1}^{N}\left(\left(\frac{U_{vg}-n-1}{U_g-n-1}\right)\left(\frac{S_{vg}}{S_g}\right)\right)$$

Empirical Evaluation: Security Evaluation

- Probability of exploitation against HideM
  - $\frac{S_{vg}}{S_g}$ is gadget specific
  - Replace $\frac{S_{vg}}{S_g}$ with an observed average for the distribution

$$\prod_{n=1}^{N}\left(\left(\frac{U_{vg}-n-1}{U_g-n-1}\right)\left(\frac{S_{vg}}{S_g}\right)\right)$$

Empirical Evaluation: Security Evaluation

- Use two tools to find ROP gadgets in memory
  - ROPGadget
  - RP++
- Gadgets limited to 4 bytes in length
- Calculate probability of exploitation for tested binaries given N=1
  - Only one valid gadget required to exploit

Empirical Evaluation: Security Evaluation

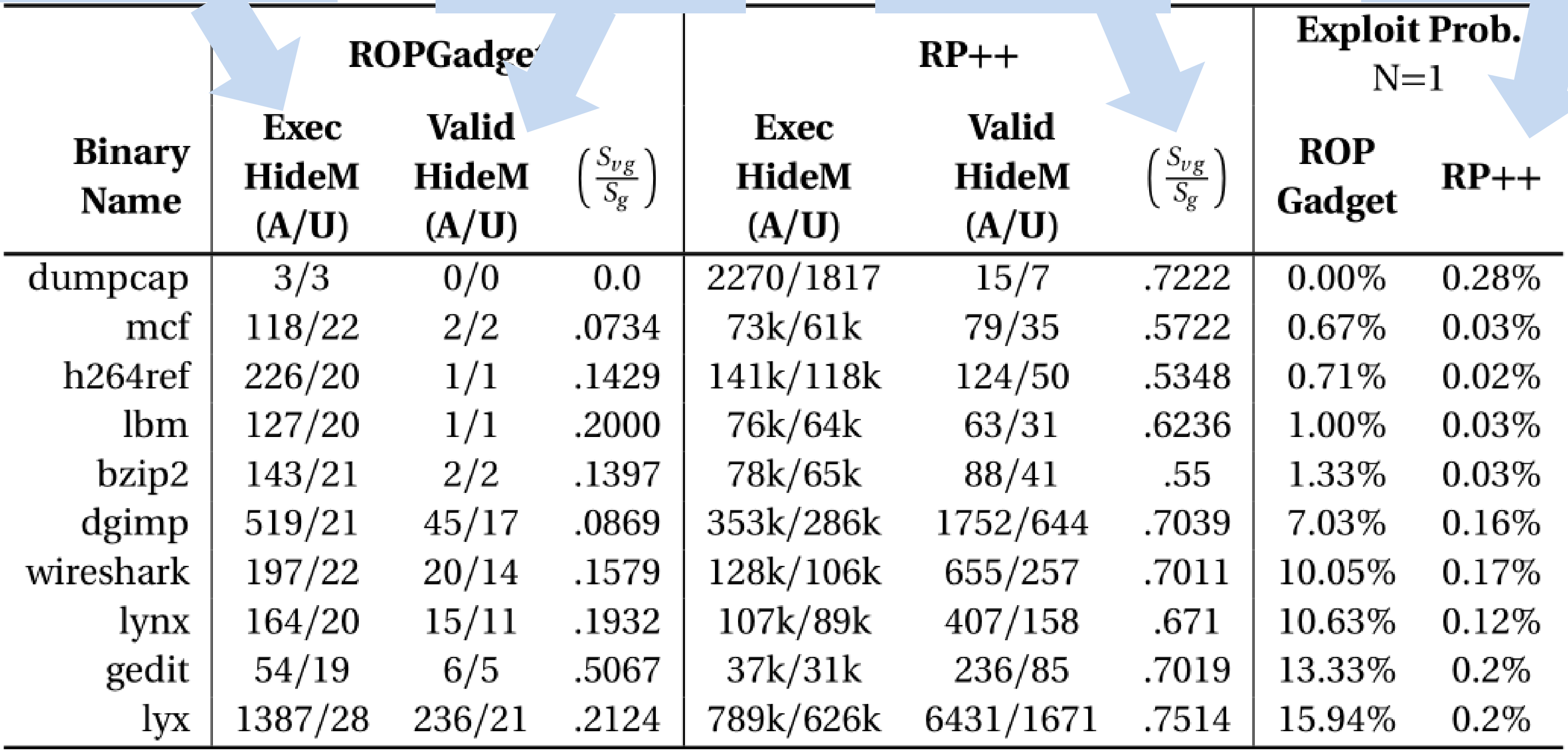
- 5 highest and lowest exploit probability (N=1)

Before HideM

After HideM

Distribution

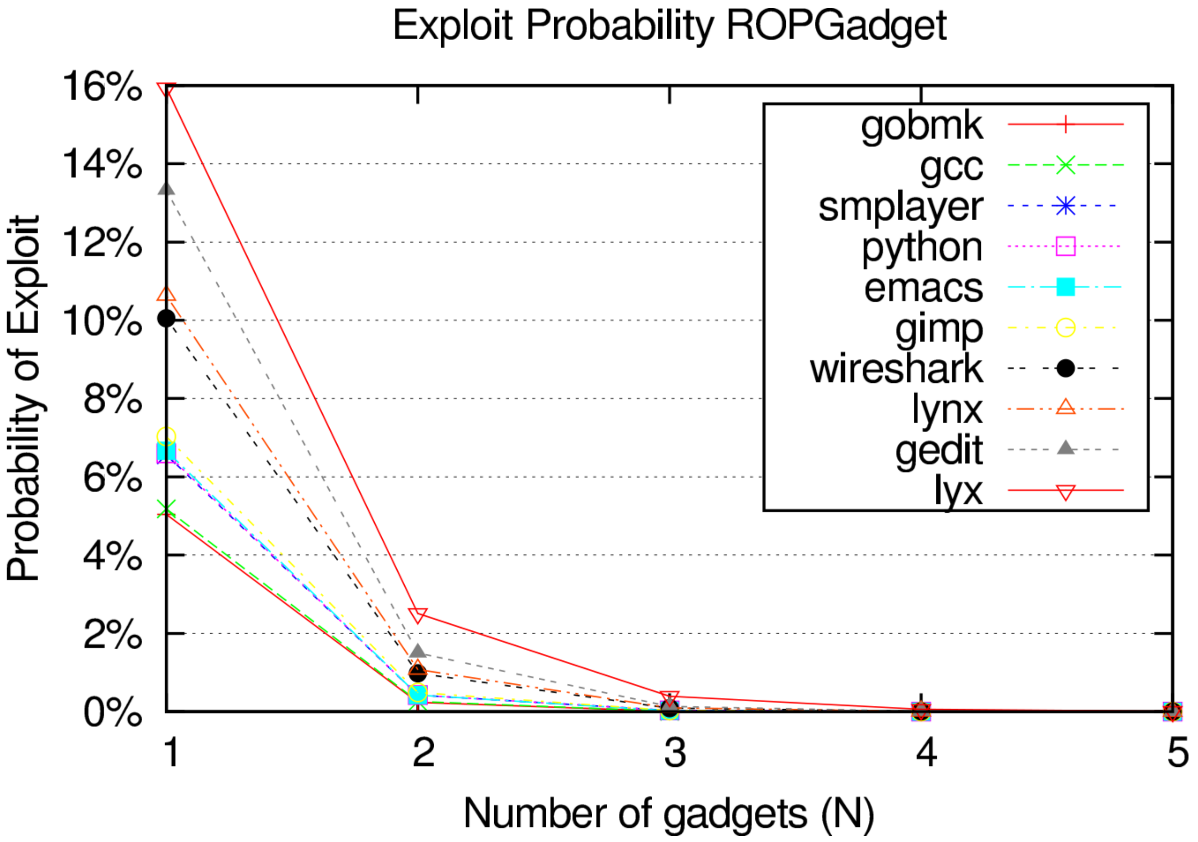
Guessing



Binary Name	ROPGadge			RP++			Exploit Prob. N=1	
	Exec HideM (A/U)	Valid HideM (A/U)	$\left(\frac{S_{vg}}{S_g}\right)$	Exec HideM (A/U)	Valid HideM (A/U)	$\left(\frac{S_{vg}}{S_g}\right)$	ROP Gadget	RP++
dumpcap	3/3	0/0	0.0	2270/1817	15/7	.7222	0.00%	0.28%
mcf	118/22	2/2	.0734	73k/61k	79/35	.5722	0.67%	0.03%
h264ref	226/20	1/1	.1429	141k/118k	124/50	.5348	0.71%	0.02%
lbm	127/20	1/1	.2000	76k/64k	63/31	.6236	1.00%	0.03%
bzip2	143/21	2/2	.1397	78k/65k	88/41	.55	1.33%	0.03%
dgimp	519/21	45/17	.0869	353k/286k	1752/644	.7039	7.03%	0.16%
wireshark	197/22	20/14	.1579	128k/106k	655/257	.7011	10.05%	0.17%
lynx	164/20	15/11	.1932	107k/89k	407/158	.671	10.63%	0.12%
gedit	54/19	6/5	.5067	37k/31k	236/85	.7019	13.33%	0.2%
lyx	1387/28	236/21	.2124	789k/626k	6431/1671	.7514	15.94%	0.2%

Empirical Evaluation: Security Evaluation

- 10 highest exploit probability
- Gadgets limited to 4 bytes



Conclusion

- HideM provides protection against code disclosure
  - Hides codes from being read
  - Applies code reading policy to enable selective fine-grained reads of code
    - Supports C++ exception handling
- Supports COTS binaries
  - Identifies data locations through offline static analysis, minimal manual verification
- Existing systems can be retrofitted for protection
- Limited impact on performance

Thanks

- Questions?

✉ jjgionta@ncsu.edu
💻 gionta.org