MemoryTrap: Booby Trapping Memory to Counter Memory Disclosure Attacks with Hardware Support

Chenke Luo[1,2], Jiang Ming[2], Dongpeng Xu[3], Guojun Peng[1], Jianming Fu[1]

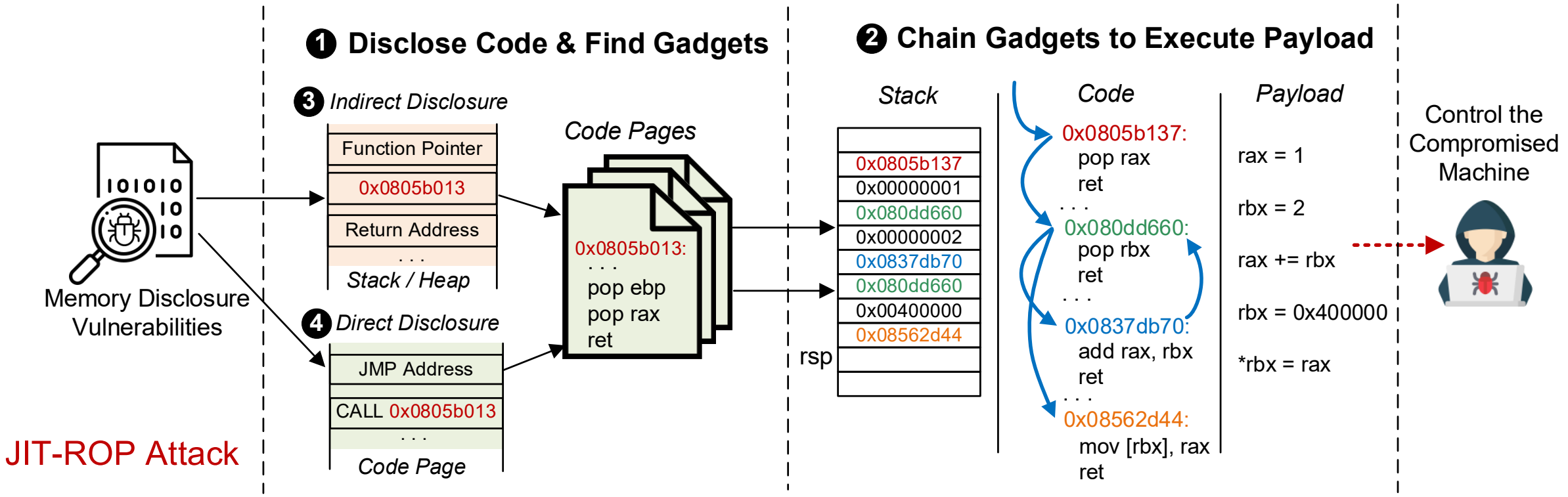
[1] Wuhan University [2] Tulane University [3] University of New Hampshire

WUHAN UNIVERSITY

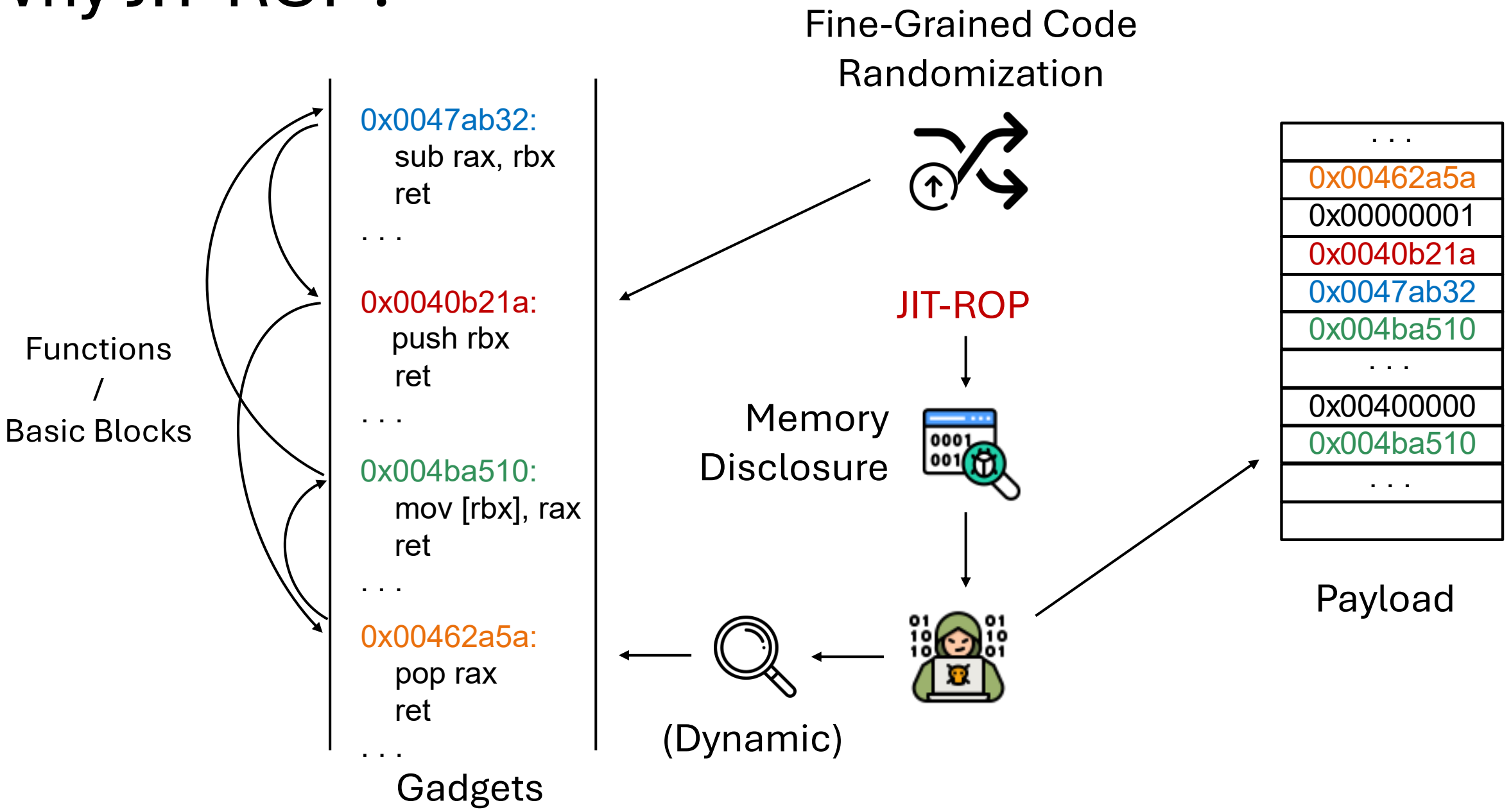
Tulane University

University of New Hampshire

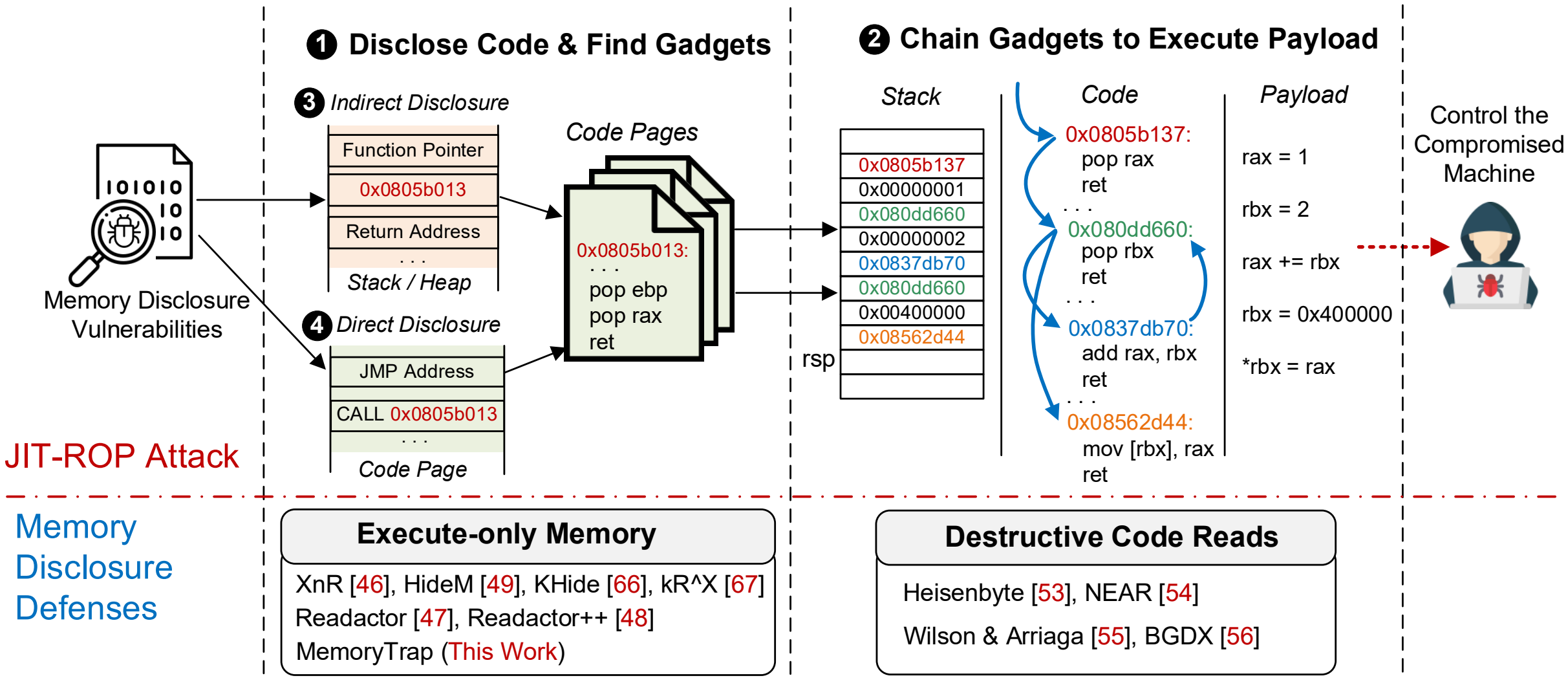
Just-In-Time Return-Oriented Programming (JIT-ROP)



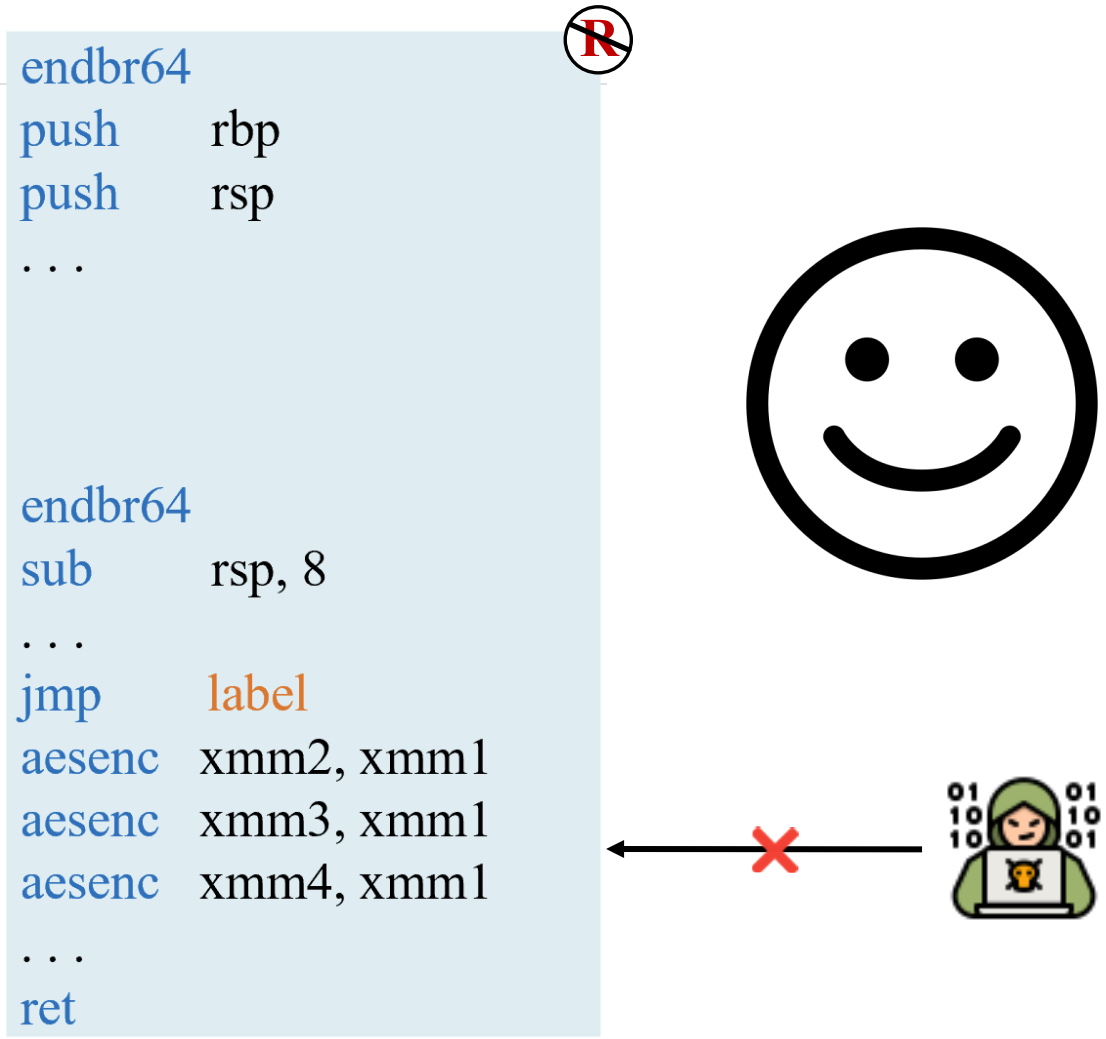
Why JIT-ROP?


Fine-Grained Code Randomization
Functions / Basic Blocks
0x0047ab32:
sub rax, rbx
ret
. . .
0x0040b21a:
push rbx
ret
. . .
0x004ba510:
mov [rbx], rax
ret
. . .
0x00462a5a:
pop rax
ret
. . .
Gadgets
JIT-ROP
Memory Disclosure
(Dynamic)
. . .
0x00462a5a
0x00000001
0x0040b21a
0x0047ab32
0x004ba510
. . .
0x00400000
0x004ba510
. . .
Payload

JIT-ROP Defenses

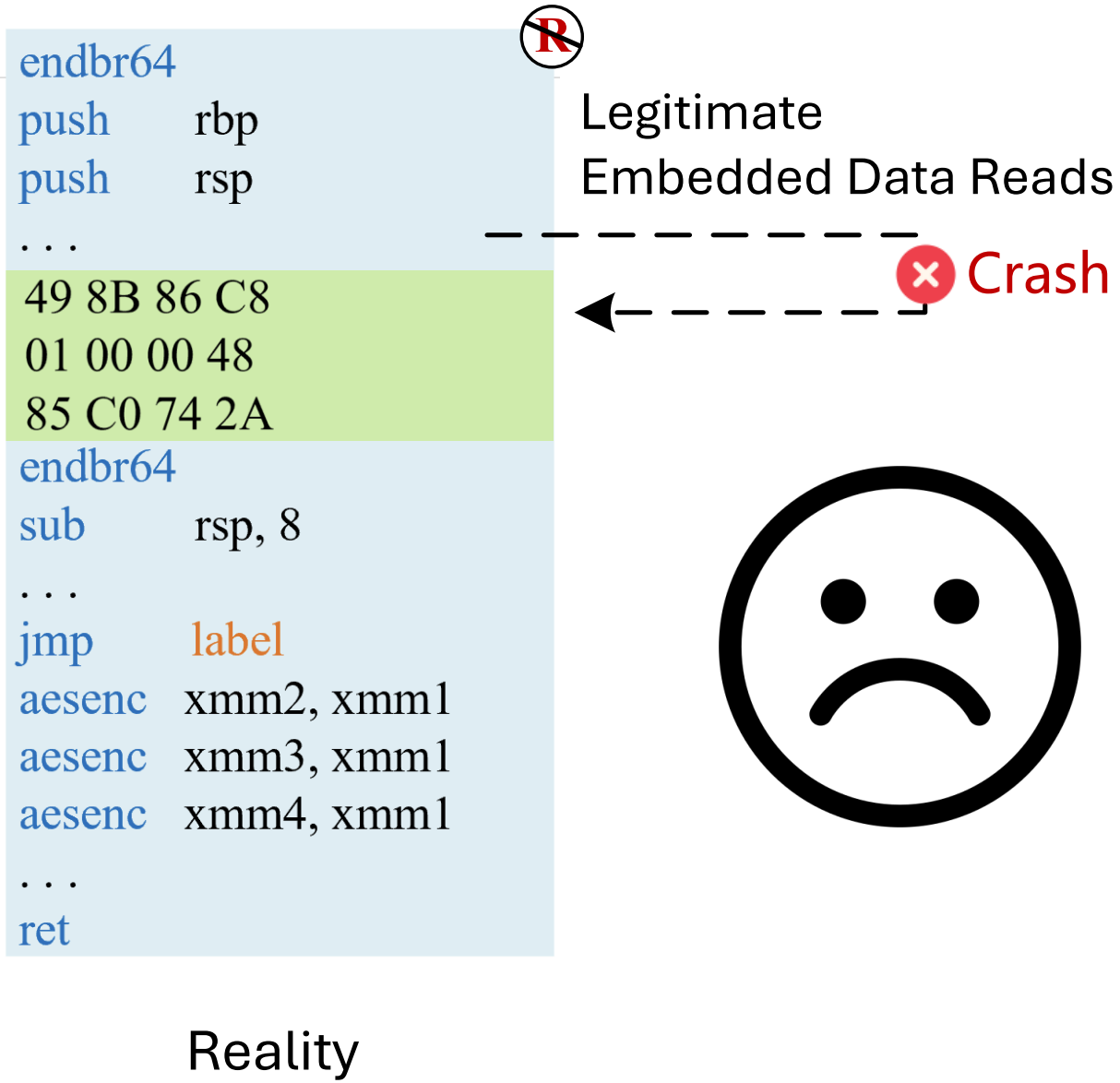


Execute-only Memory (XoM)



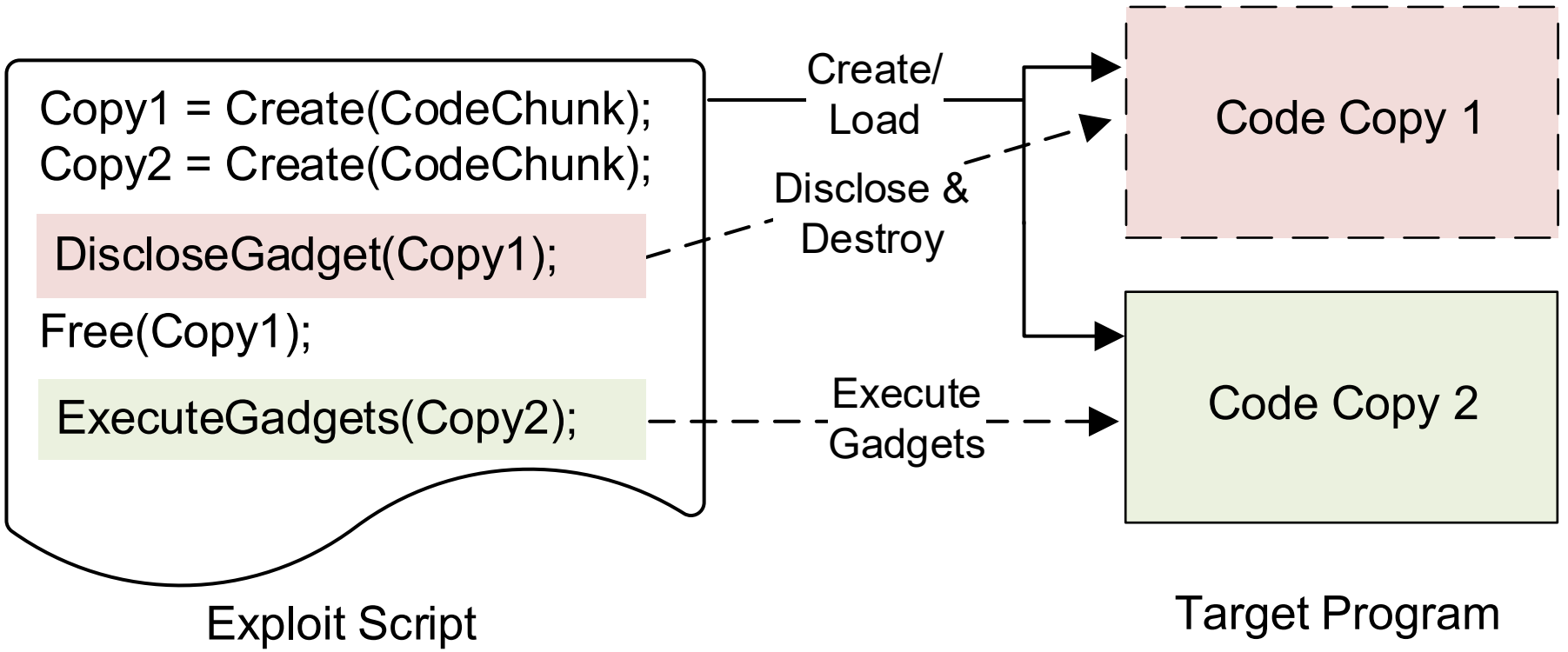
Ideal

Execute-only Memory (XoM)



Destructive Code Reads

DCR destroys memory that has been read, preventing it from being executed as code.



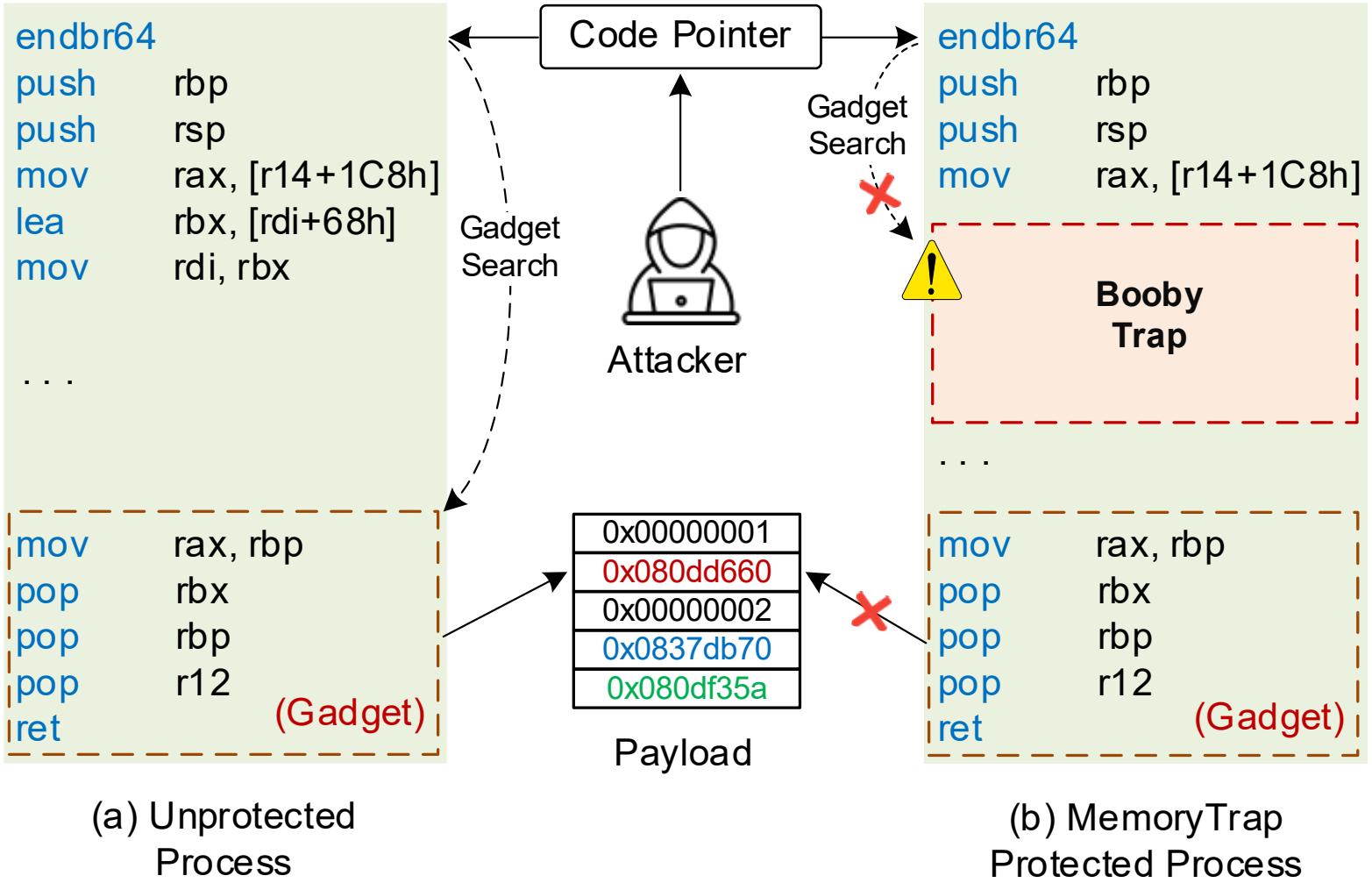
MemoryTrap (This Work)

We present a new cyber deception idea, MemoryTrap, to detect memory disclosure attempts and eventually prevent JIT-ROP attacks.

MemoryTrap tolerates legitimate data reads in executable memory while seamlessly defending against powerful code inference attacks.

We take advantage of Intel's MPK feature to develop an efficient, fine-grained memory permission control mechanism

Basic Idea of MemoryTrap



(a) Unprotected Process

(b) MemoryTrap Protected Process

Hide Booby Traps into Execution Paths

- We randomly generate a variable-size code snippet as a booby trap

- We hide the booby traps from the protected program's normal execution paths by inserting a *JMP* instruction in front of inserted code snippets

- We set the whole booby trap, including the prefixed *JMP* instruction, as **unreadable**

Booby Trap Insertion Strategy

1. We insert at least one booby trap in each function;

2. For the function whose size is larger than 4KB, we insert an additional booby trap for every 4KB code.

Booby Trap Insertion Strategy

Table 1: The distance distribution of every two adjacent booby traps in the hardened applications' binary code.

	0∼100B	100B∼1KB	1∼2KB	2∼4KB	>4KB
Nginx	0%	73.2%	16.5%	10.3%	0%
Apache	27.5%	65.3%	3.6%	3.6%	0%
Lighttpd	16.0%	73.7%	4.7%	5.6%	0%
MySQL	30.6%	57.2%	6.5%	5.7%	0%
MongoDB	25.5%	62.3%	6.4%	5.8%	0%
Redis	26.1%	65.9%	4.1%	3.9%	0%
SQLite	20.9%	55.2%	7.1%	16.8%	0%
SPEC 2017	47.4%	44.6%	3.9%	4.1%	0%
Average	24.3%	62.2%	6.6%	6.9%	0%

Detecting Disclosure Attempts

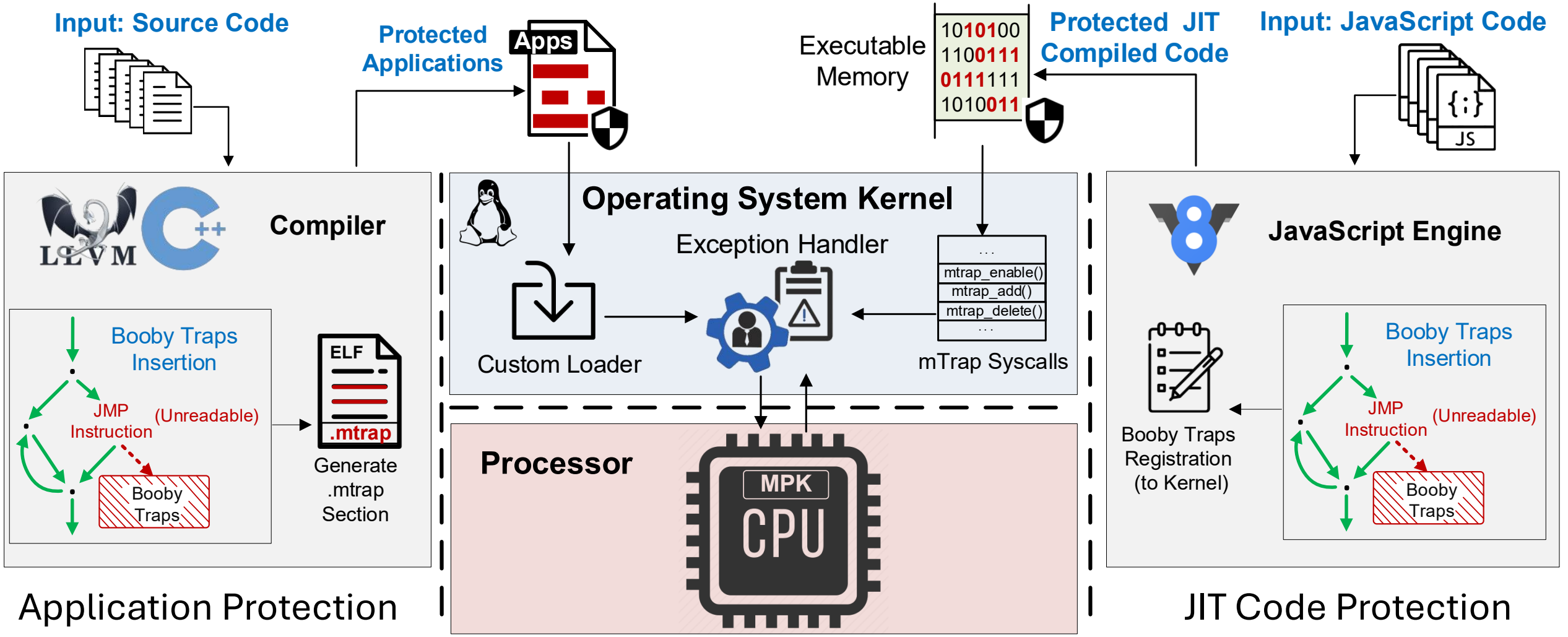
1. We log the positions of all planted booby traps;

2. We mark all code pages as unreadable;

3. When a code page read occurs, we detect the target address;

4. If the target address is within booby trap areas, we consider it as an attack.

Architecture of MemoryTrap



Security Evaluation

```
1. 0x0804b884: pop esi; pop ebp; ret;
... 192 Booby Traps ...
2. 0x0805ddbb: push ecx; ret;
... 162 Booby Traps ...
3. 0x0806e948: add ecx, [esi]; ret;
... 103 Booby Traps ...
4. 0x0807b586: mov [ecx], eax; mov [ecx+4], edx; mov eax, 0; ret;
... 47  Booby Traps ...
5. 0x08080a57: pop ecx; add al, 0x89; ret;
... 279 Booby Traps ...
6. 0x0809a65b: pop eax; add al, 0x89; ret;
... 50  Booby Traps ...
7. 0x0809ef79: push eax; add al, 0x83; ret;
```

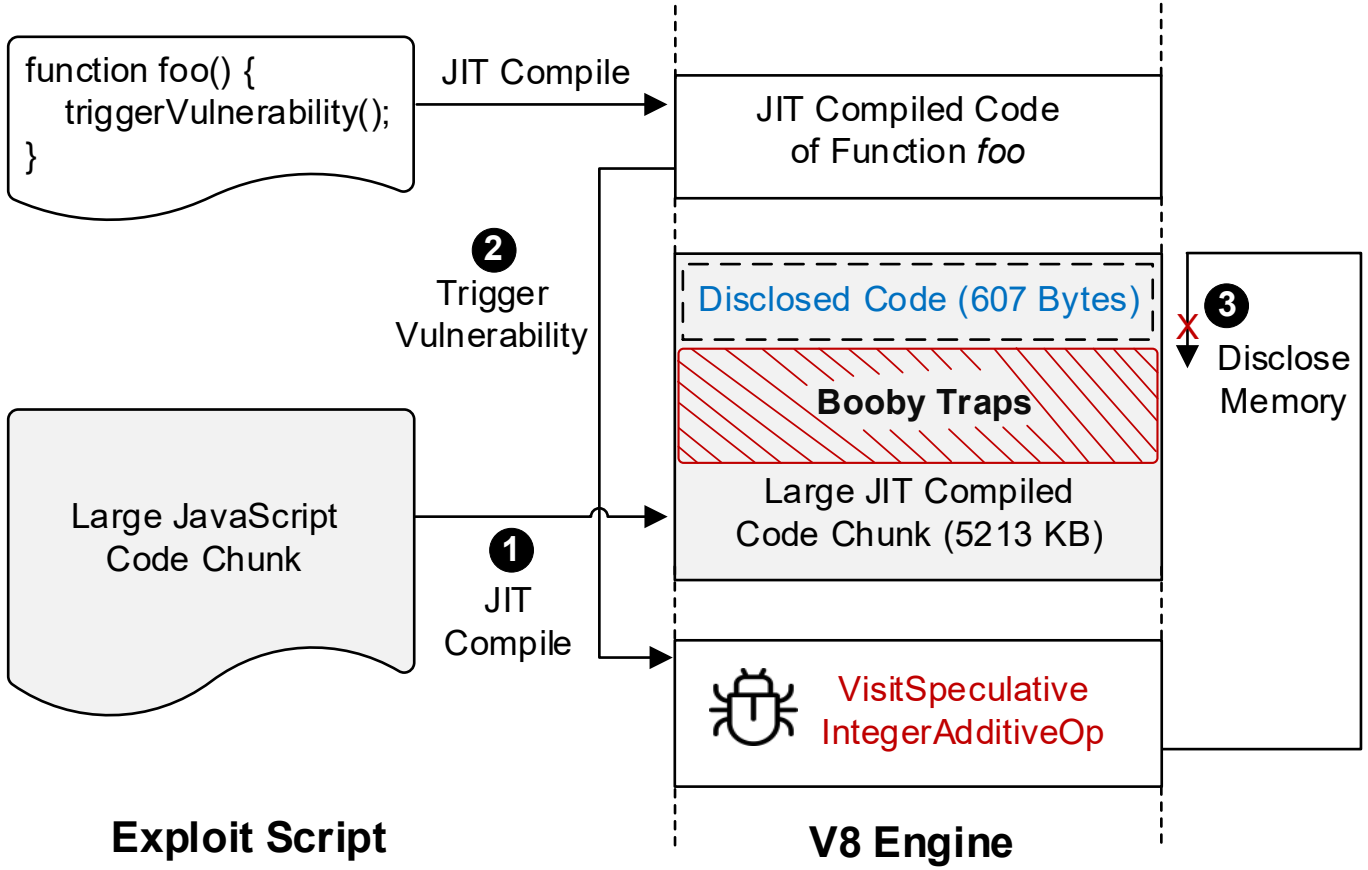
Gadgets and booby traps distribution for the CVE-2013-2028 exploit.

Security Evaluation

Table 2: Attackers' ability to find potential gadgets when MemoryTrap is enabled. The vulnerable program is Nginx.

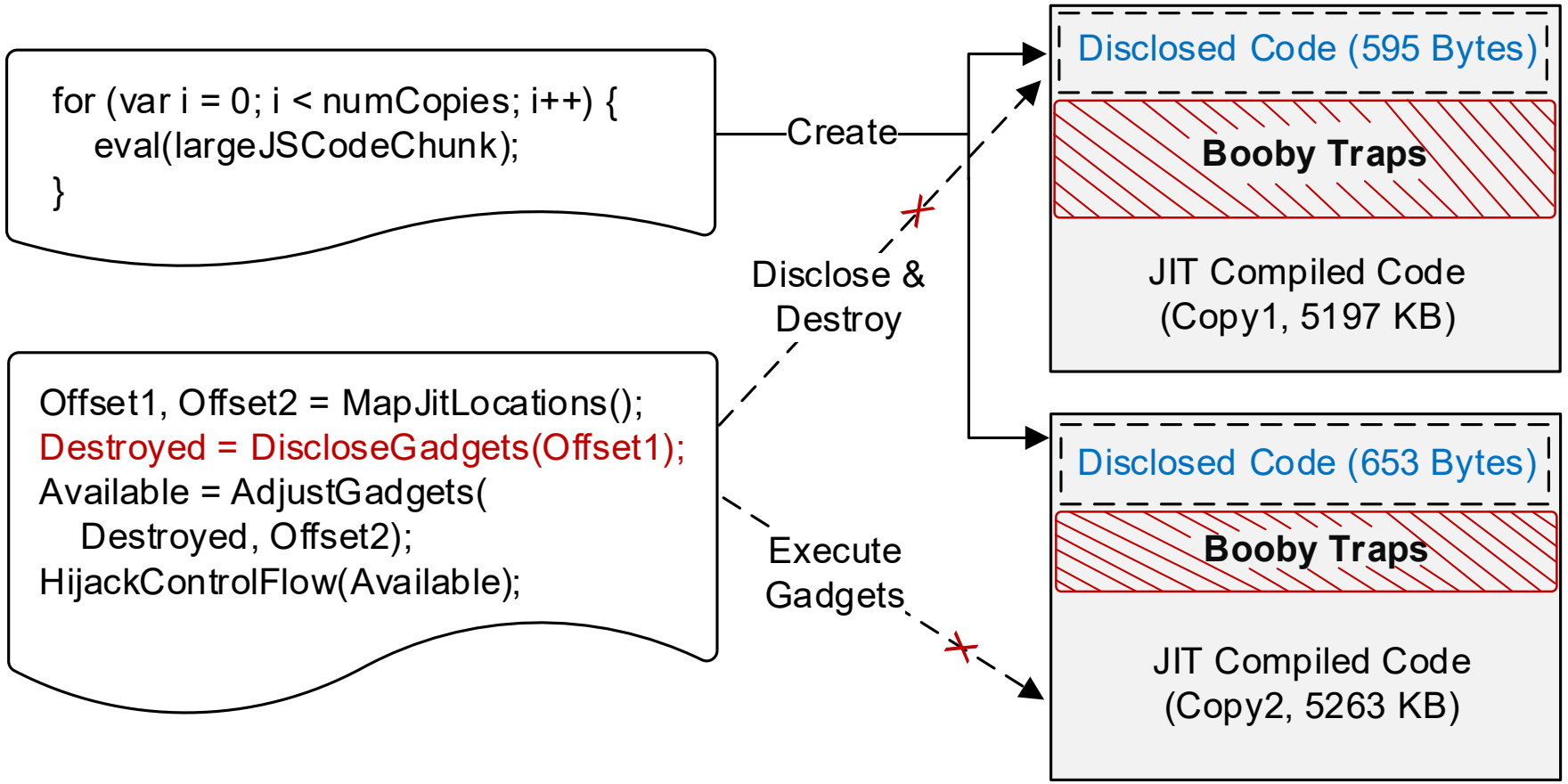
Code Page Traversal Strategy	Average Bytes Disclosed	# of Attempts to Find One Gadget
Breadth-first	657	25
Depth-first	299	14

Security Evaluation



Using exploit of CVE-2020-16040 to disclose memory of JIT compiled code protected by MemoryTrap.

Security Evaluation



JavaScript JIT cloning attack maintains two copies,
but our booby traps spread over in each copy.

Performance Evaluation

LMBench:

Table 3: Time for kernel operations related to process creation and page fault handling (in μs). Smaller is better. In the first row are the names of benchmarks in lmbench.

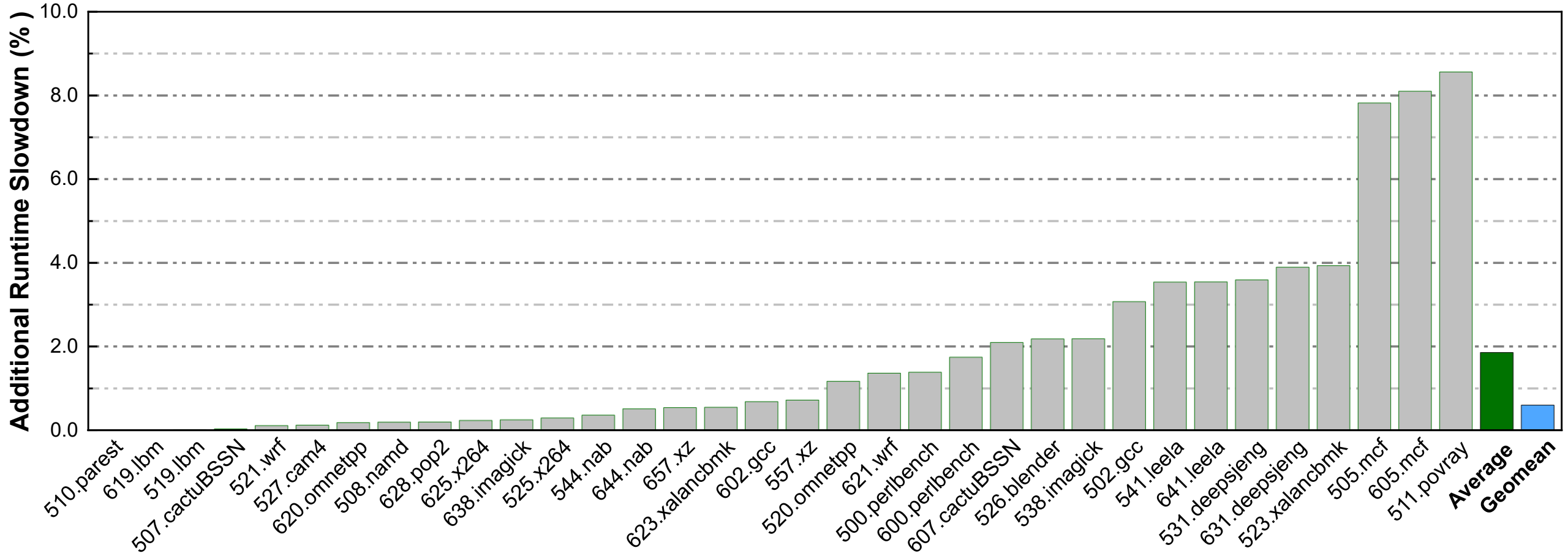
Kernel	Fork Proc	Exec Proc	Page Fault	Prot Fault
Standard	113	347	0.776	0.481
MemoryTrap	114	349	0.780	0.485
Overhead	0.88%	0.58%	0.52%	0.83%

Table 4: Context switch time (in μs). Smaller is better.

Kernel	2p	2p	2p	8p	8p	16p	16P
	0K	16K	64K	16K	64K	16K	64K
Standard	2.09	2.10	2.61	2.43	2.59	2.64	2.83
MemoryTrap	2.13	2.15	2.48	2.49	2.62	2.66	2.79
Overhead	1.91%	2.38%	-4.24%	2.47%	1.16%	0.76%	-1.43%

Performance Evaluation

SPEC CPU 2017:



Average: 1.85% Geomean: 0.60%

Performance Evaluation

JIT-Engine Protection:

	A*	Audio	Imaging	JSON	SJCL
Original	149.7ms	285.0ms	360.3ms	145.0ms	508.9ms
MemoryTrap	151.6ms	290.7ms	365.1ms	146.1ms	520.8ms
Overhead	1.27%	2.00%	1.33%	0.76%	2.34%

Performance Evaluation

Worst-Case Benchmark:

Continuously reading embedded data, causing 3.23X performance overhead

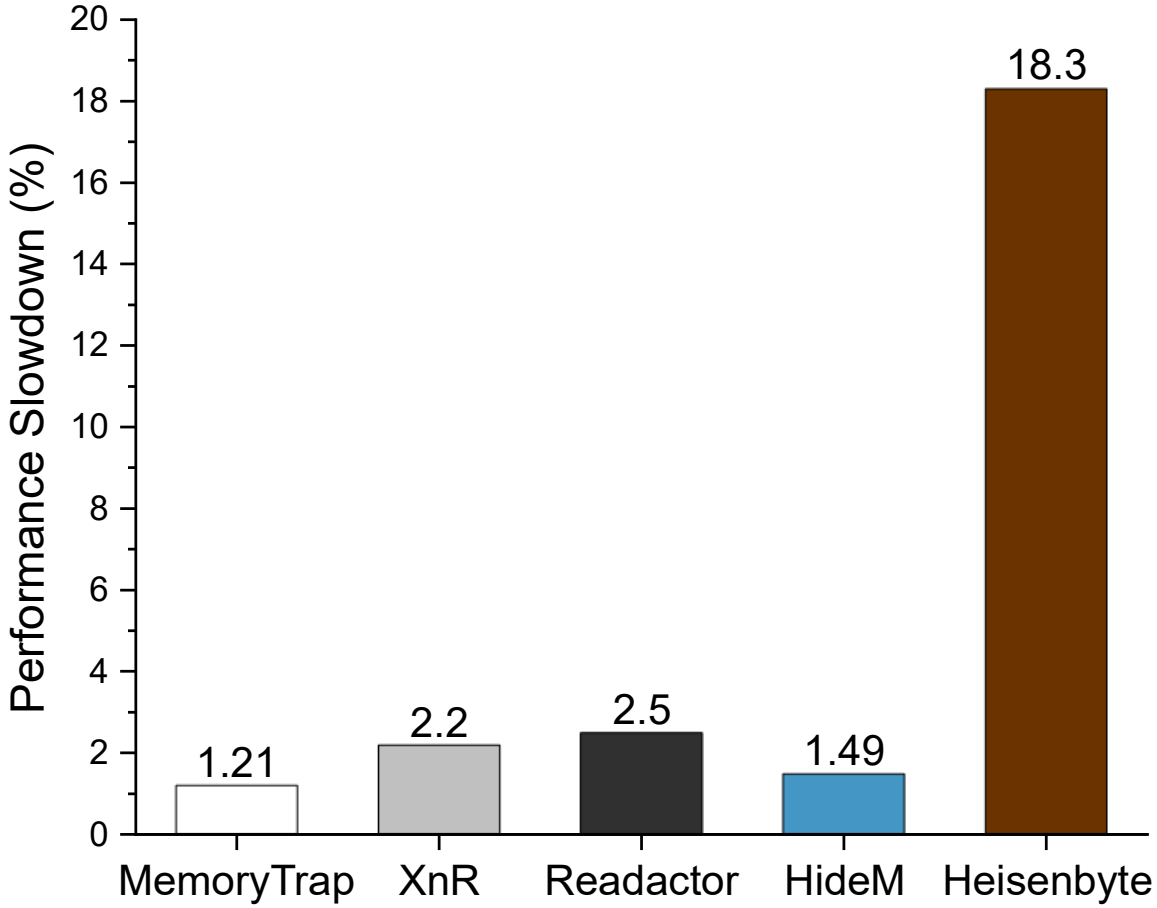
$$Read\ Intensity = \frac{\#\ of\ Legitimate\ Reads}{\#\ of\ Executed\ Instructions}$$

SPEC CPU 2017: $7.0E^{-11}$ Web Servers: $1.64E^{-10}$ Databases: $4.8E^{-12}$

OpenSSL: $1.4E^{-7}$

Performance Evaluation

Performance Comparison:



Conclusion

- This paper introduces MemoryTrap, a hardware-assisted technique to counter memory disclosure attacks

- MemoryTrap inserts unreadable, variable-size code snippets (booby traps) at strategic points to block unauthorized code page reads, utilizing Intel's MPK for efficient, fine-grained memory permission control.

- Our experiments demonstrate that MemoryTrap reveals a strong resistance to various memory disclosure attempts and negligible runtime overhead.

Thank You!