
# Dissecting complex code-reuse attacks with ROPMEMU

Mariano Graziano

Cisco Talos


# whoami

- Security researcher at Cisco Talos
- Ph.D. from Telecom ParisTech/Eurecom
- Hackademic
- Malware analysis / Memory forensics / Mitigations


# CODE INJECTION ATTACKS

OS KERNEL


# CODE INJECTION ATTACKS


0xc0c4c0c4 ?? ?? shellcode ins1 2

0xc0c4c0c8 ?? ?? shellcode ins2 3

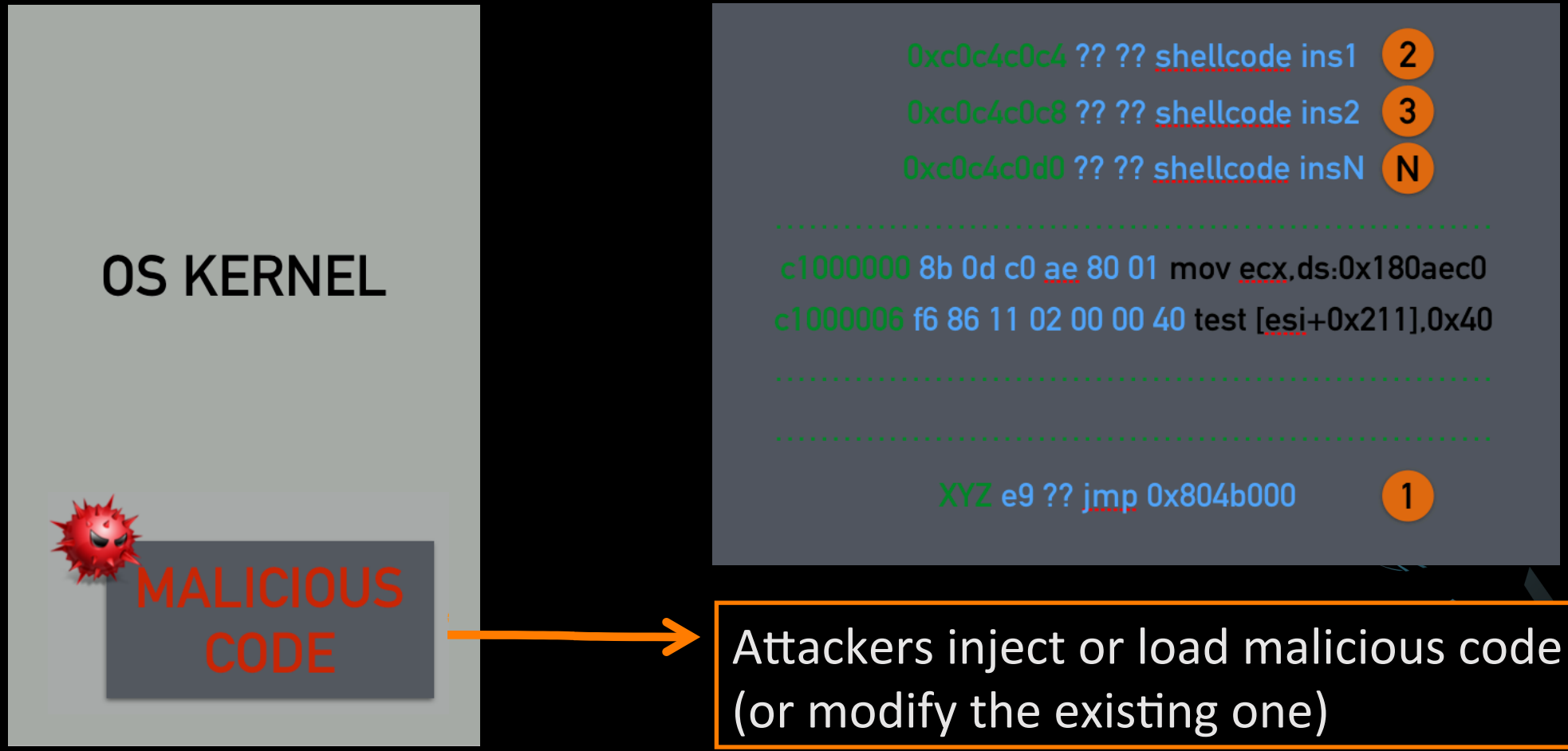
0xc0c4c0d0 ?? ?? shellcode insN N

c1000000 8b 0d c0 ae 80 01 mov ecx,ds:0x180aec0

c1000006 f6 86 11 02 00 00 40 test [esi+0x211],0x40

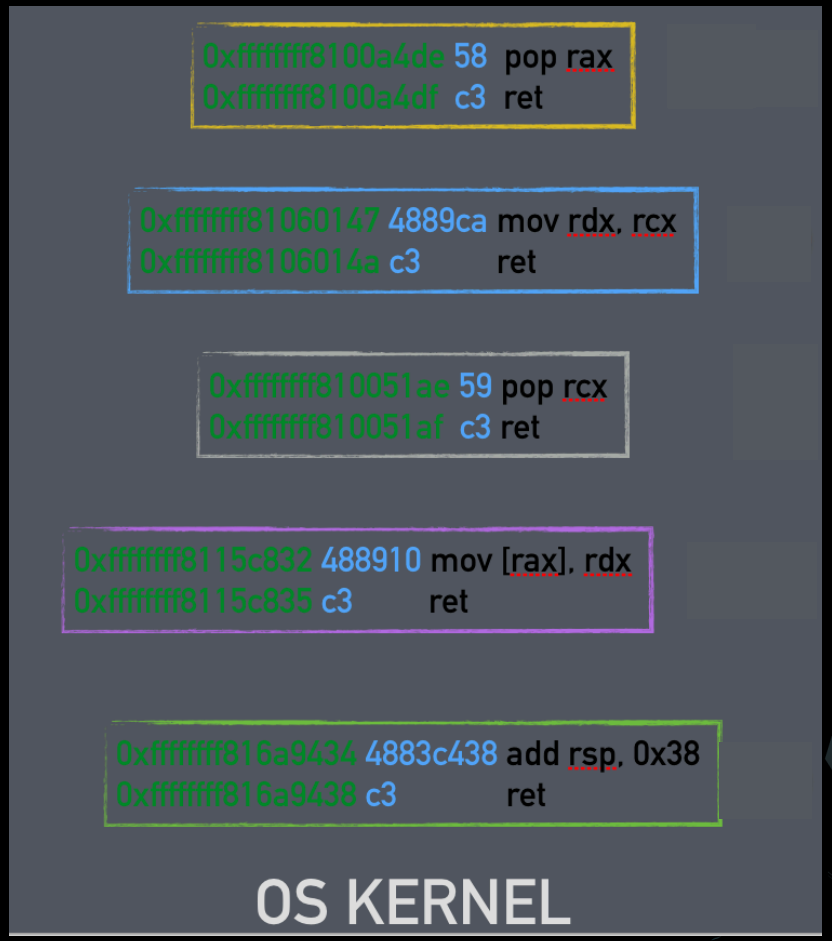
XYZ e9 ?? jmp 0x804b000 1

# CODE INJECTION ATTACKS




Attackers inject or load malicious code (or modify the existing one)

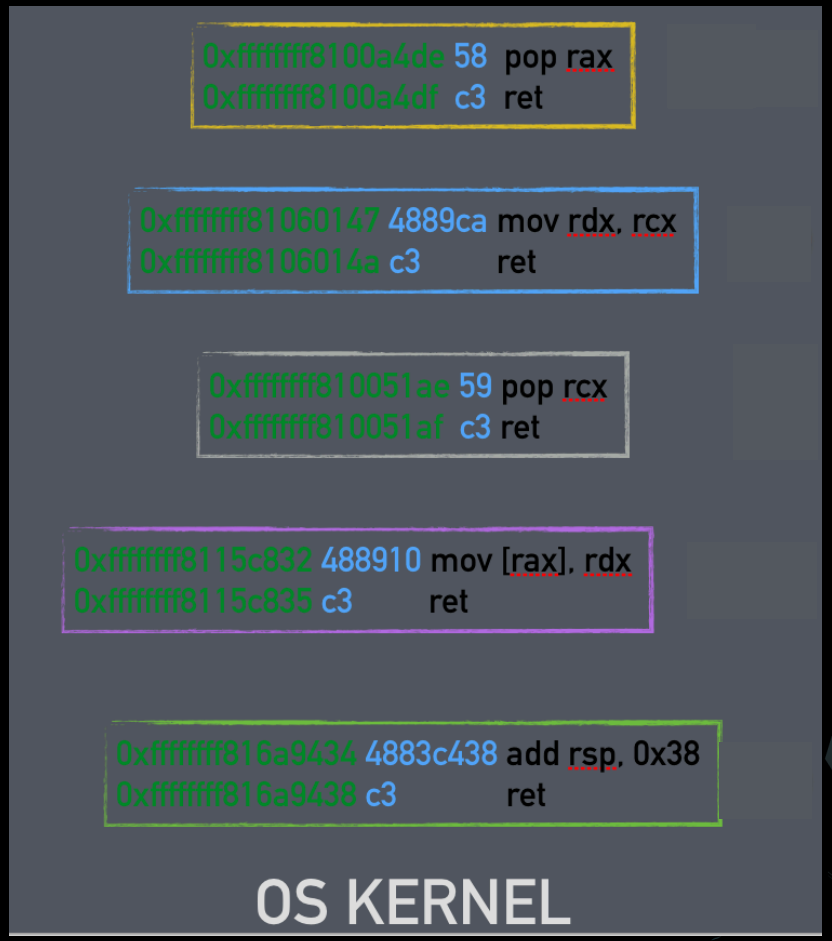

# CODE-REUSE ATTACKS - ROP



# CODE-REUSE ATTACKS - ROP

SP = 0xFFFF88001B800000

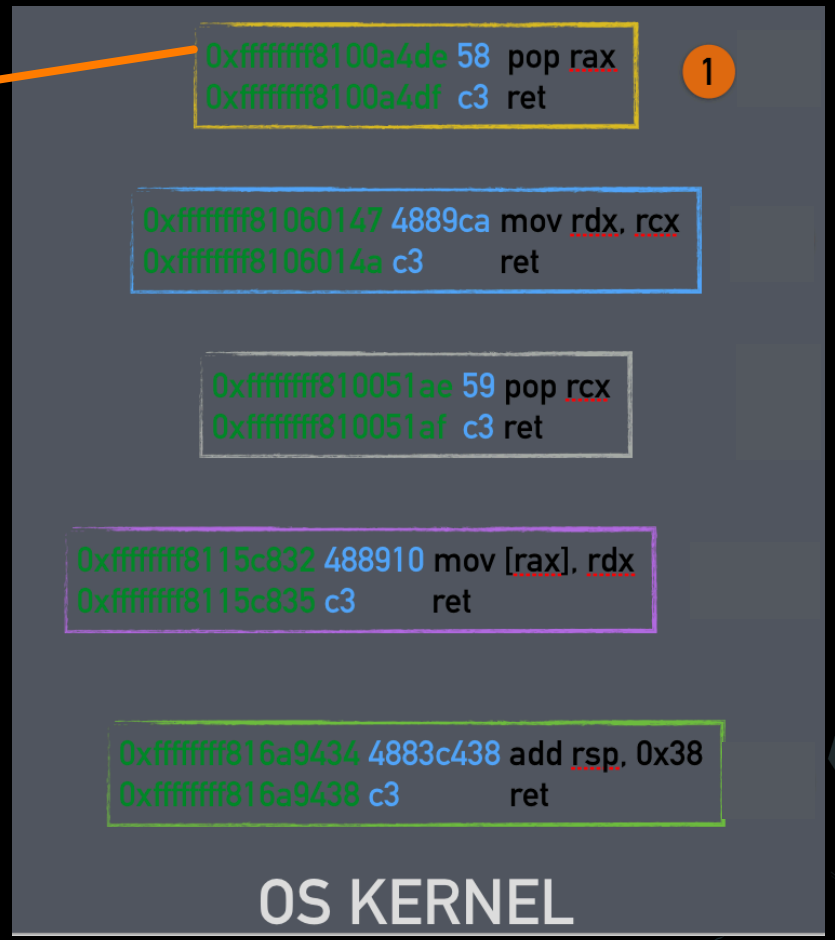
| | |
|---|---|
| 0x00) | 0xffffffff8100a4de |
| | 0xffff88001bc00000 |
| | 0xffffffff8115c832 |
| | 0xffffffff8100a4de |
| | 0xffff88001bc00008 |
| | 0xffffffff81060147 |
| 0x30) | 0xffffffff8115c832 |
| | 0xffffffff810051ae |
| | 0xffffffff816a9434 |



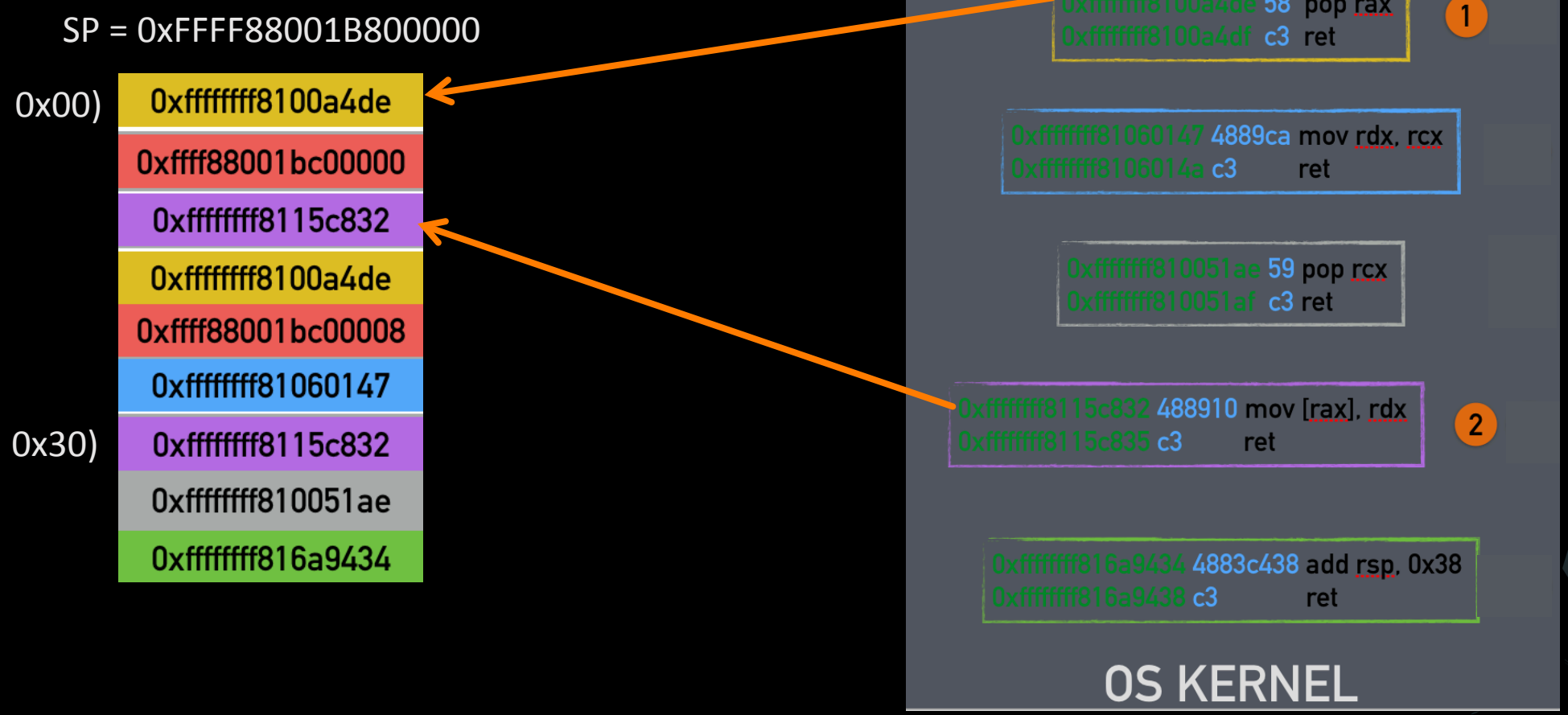
# CODE-REUSE ATTACKS - ROP

SP = 0xFFFF88001B800000

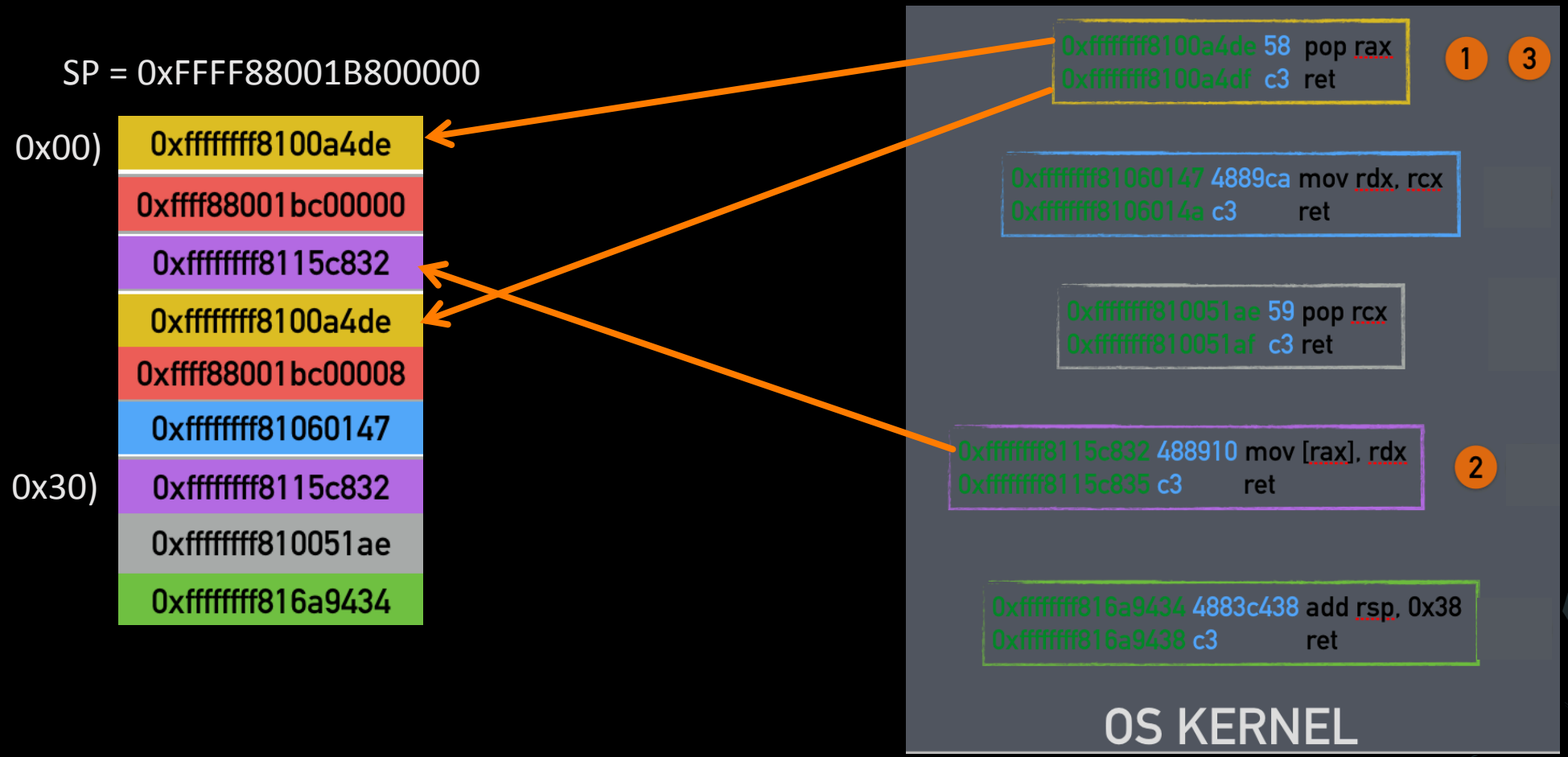
| | |
|---|---|
| 0x00) | 0xffffffff8100a4de |
| | 0xffff88001bc00000 |
| | 0xffffffff8115c832 |
| | 0xffffffff8100a4de |
| | 0xffff88001bc00008 |
| | 0xffffffff81060147 |
| 0x30) | 0xffffffff8115c832 |
| | 0xffffffff810051ae |
| | 0xffffffff816a9434 |



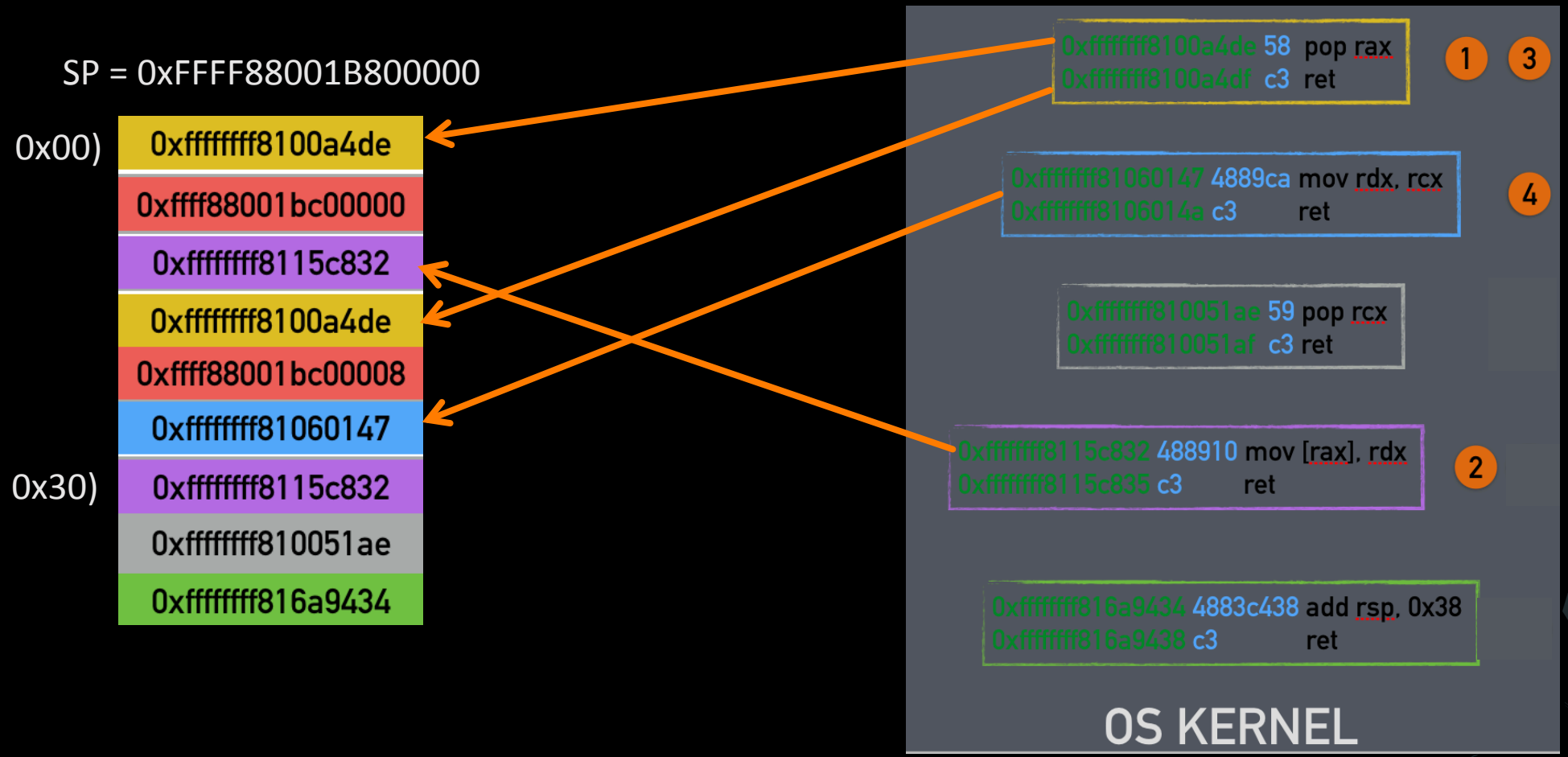
# CODE-REUSE ATTACKS - ROP



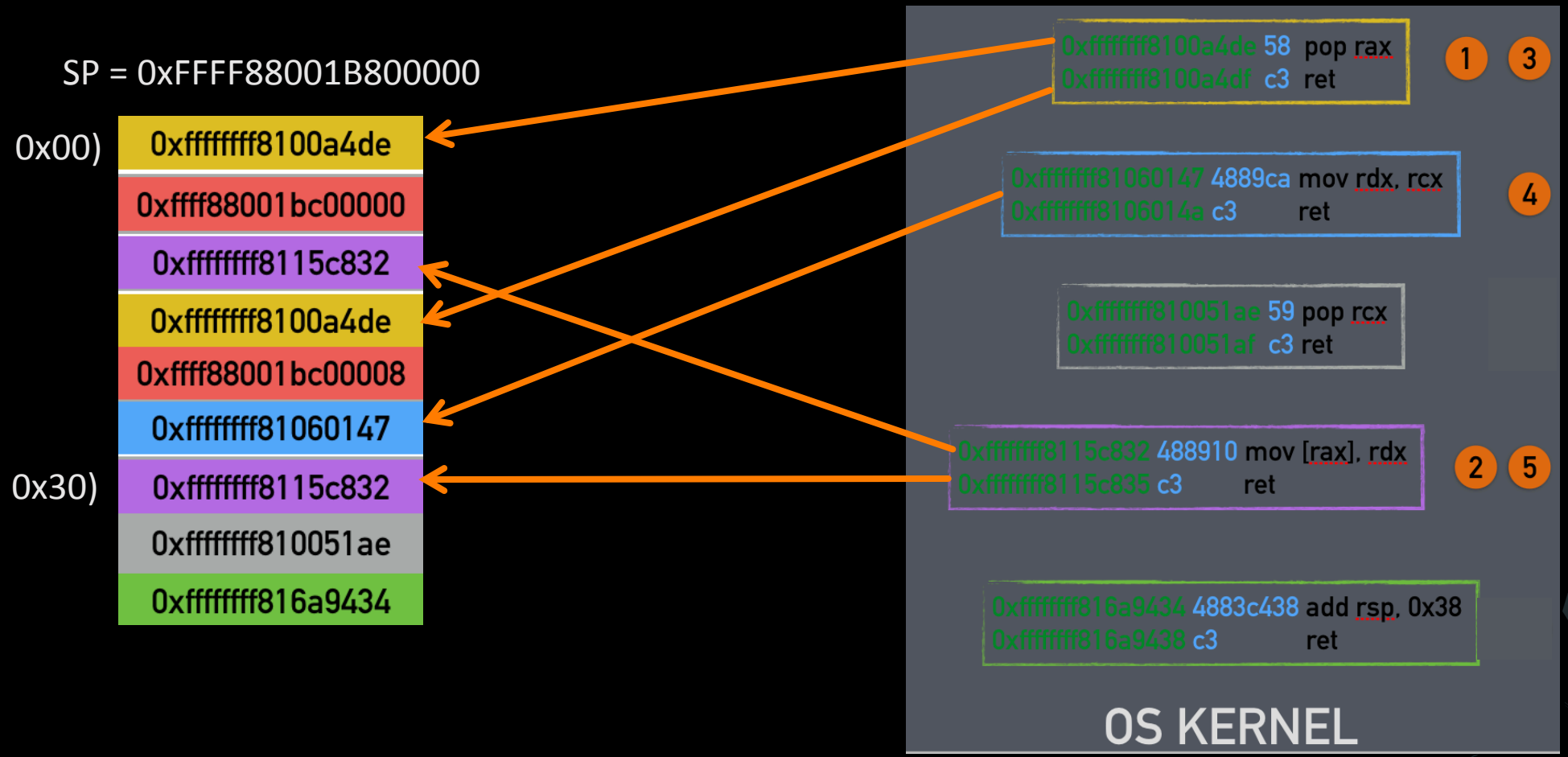
# CODE-REUSE ATTACKS - ROP



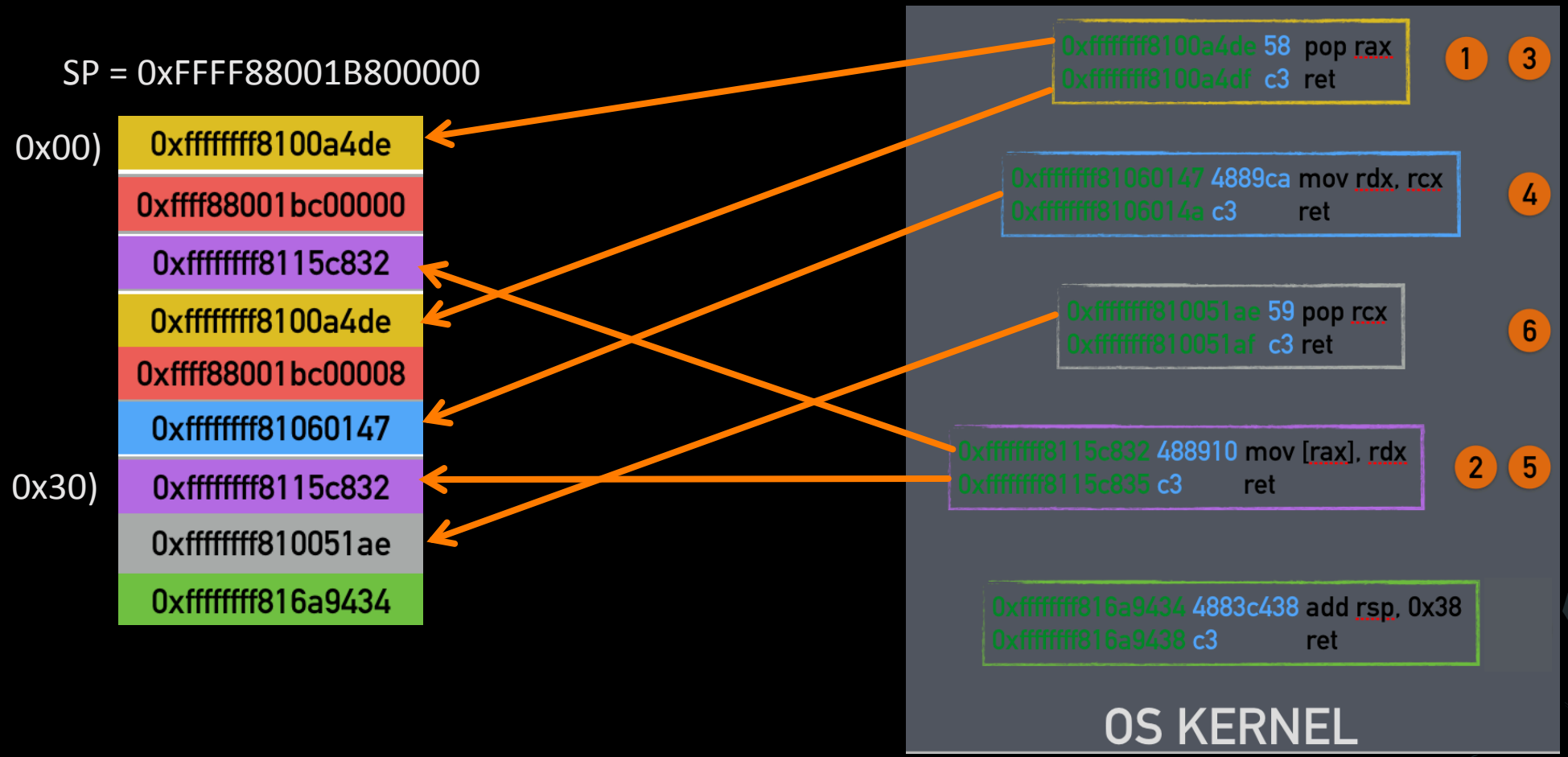
# CODE-REUSE ATTACKS - ROP



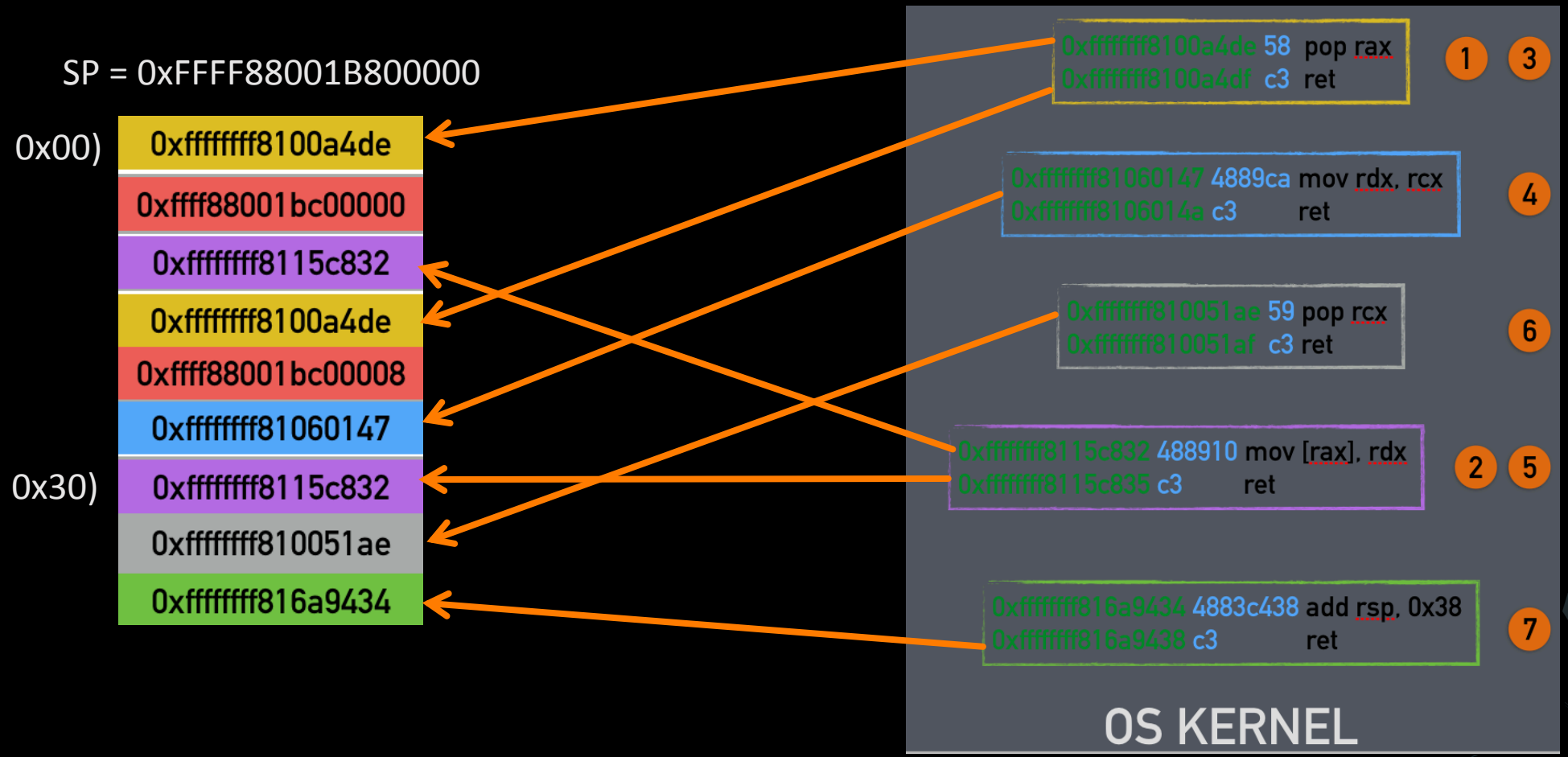
# CODE-REUSE ATTACKS - ROP



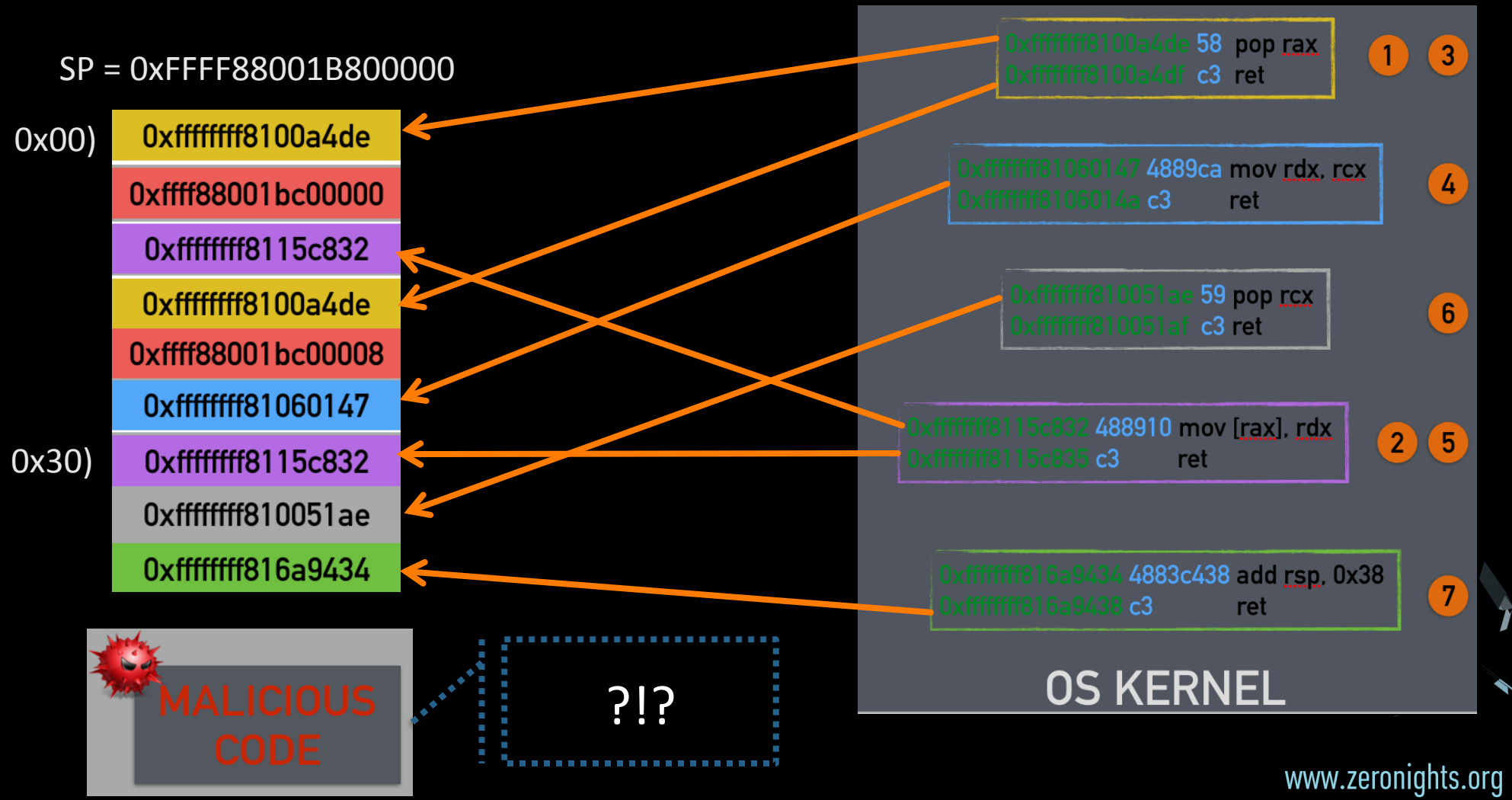
# CODE-REUSE ATTACKS - ROP



# CODE-REUSE ATTACKS - ROP



# CODE-REUSE ATTACKS - ROP

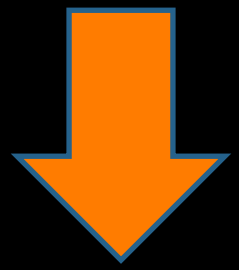



# MOTIVATION

HW and OS countermeasures forced ROP adoption

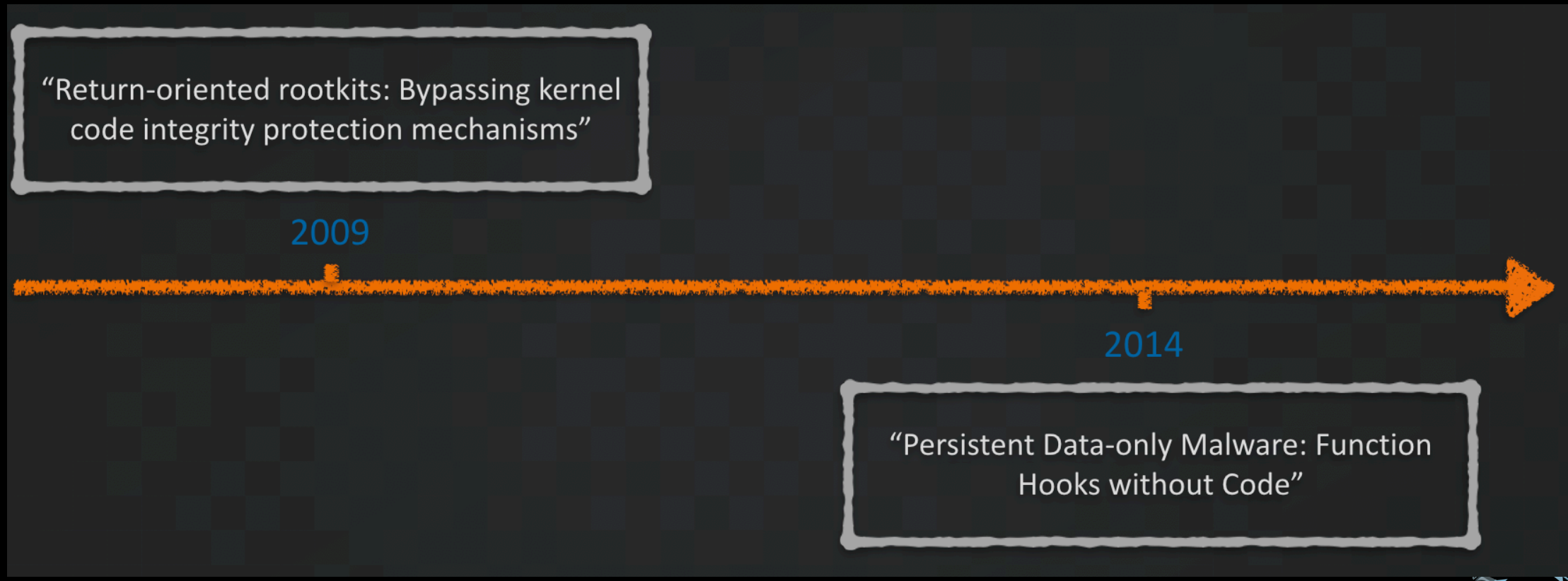

# MOTIVATION

HW and OS countermeasures forced ROP adoption



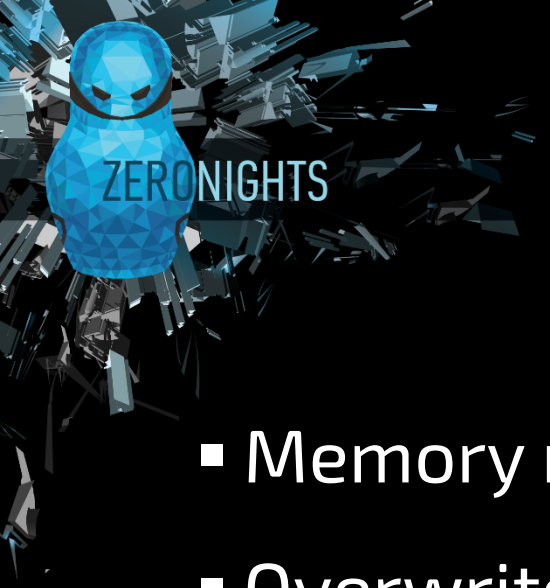
- Persistent ROP rootkit – Vogl et al. [NDSS 14]
- ROP as an obfuscation technique (malware in the wild)
- All existing tools cope with injected code
- No tools to dissect ROP payloads

# ROP ROOTKITS




# CHUCK - CHALLENGES

- Memory region for the persistent control structure
- Overwrites protection:
  - Interrupts
  - Self-overwriting
- Resume original/normal control flow
- Activate the persistent component




# CHUCK - CHALLENGES

- Memory region for the persistent memory region
- Overwrites protection:
  - Interrupts
  - Self-overwriting
- Resume original/normal control flow
- Activate the persistent component


# CHUCK - PERSISTENCE

- CVE-2013-2094
- `sysenter`
  - `IA32_SYSENTER_ESP (0x175)`
  - `IA32_SYSENTER_EIP (0x176)`


# CHUCK - PERSISTENCE

- CVE-2013-2094
- `sysenter`
  - `IA32_SYSENTER_ESP (0x175)` – Copy chain
  - `IA32_SYSENTER_EIP (0x176)` – ret


# CHUCK

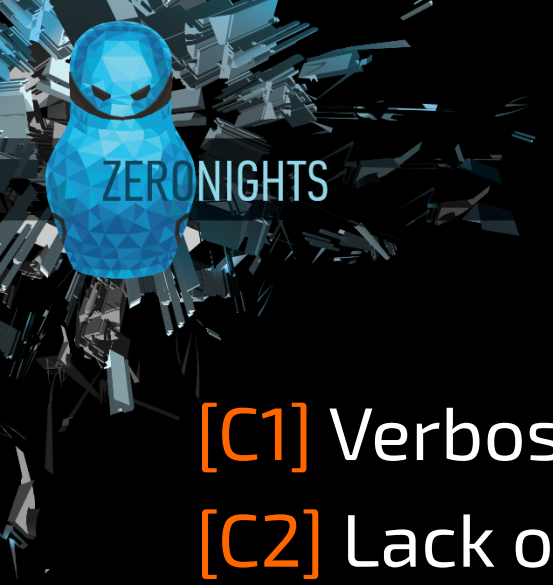
- PoC[1]:
  - LPE: CVE-2013-2094
  - OS: Ubuntu Server 3.8 with secure boot
- Hooks :
  - `sys_read`
  - `sys_getdents`
- Chains:
  - Copy chain (persistent)
  - Dispatcher chain
  - Payload chain

[1] https://www.sec.in.tum.de/persistent-data-only-malware/


# CHALLENGES


# CHALLENGES

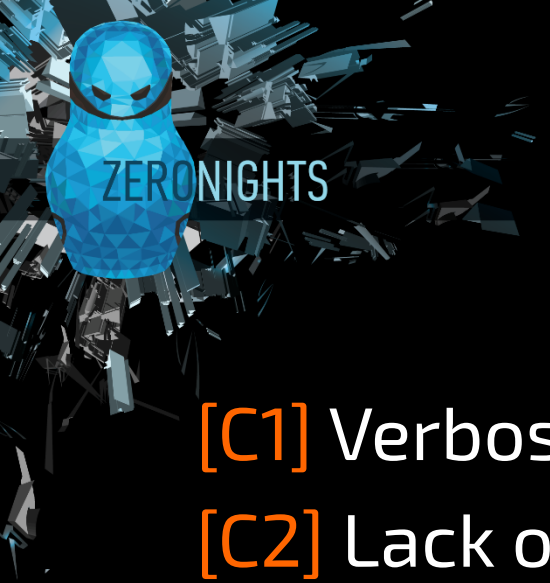
## [C1] Verbosity




# CHALLENGES

[C1] Verbosity

[C2] Lack of immediate values




# CHALLENGES

[C1] Verbosity

[C2] Lack of immediate values

[C3] Stack based instruction chaining


# CHALLENGES

[C1] Verbosity

[C2] Lack of immediate values

[C3] Stack based instruction chaining

[C4] Conditional branches


# CHALLENGES

[C1] Verbosity

[C2] Lack of immediate values

[C3] Stack based instruction chaining

[C4] Conditional branches

[C5] Return to functions


# CHALLENGES

[C1] Verbosity

[C2] Lack of immediate values

[C3] Stack based instruction chaining

[C4] Conditional branches

[C5] Return to functions

[C6] Dynamically generated chains


# CHALLENGES

[C1] Verbosity
[C2] Lack of immediate values
[C3] Stack based instruction chaining
[C4] Conditional branches
[C5] Return to functions
[C6] Dynamically generated chains
[C7] Stop condition


# CHALLENGES

[C1] Verbosity ✓

[C2] Lack of immediate values ✓

[C3] Stack based instruction chaining ✓

[C4] Conditional branches ✗

[C5] Return to functions ✗

[C6] Dynamically generated chains ✗

[C7] Stop condition ✗


# CHALLENGES

[C1] Verbosity
- Lu et al. – ACSAC12
- Yadegari et al. – S&P15

[C2] Lack of immediate values

[C3] Stack based instruction chaining

[C4] Conditional branches

[C5] Return to functions
- Stancill et al. – RAID13
- Lu et al. – ACSAC12

[C6] Dynamically generated chains

[C7] Stop condition


# APPROACH

- ROPMEMU framework adopts many techniques:
  - Memory forensics
  - Code emulation
  - Multi-path exploration
  - CFG recovery
  - Compiler transformations

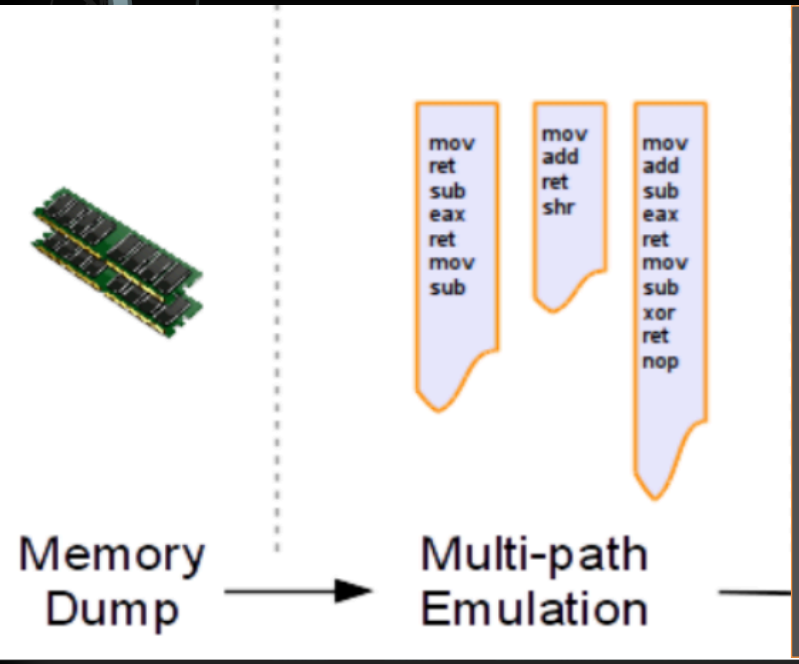

# IMPLEMENTATION

- Volatility plugins (`ropemu` and `unchain`) depend on:
  - Capstone
  - Unicorn
  - nasm
- Standalone scripts (bash and python):
  - GDB python
  - pygraphviz

# ROPMEMU

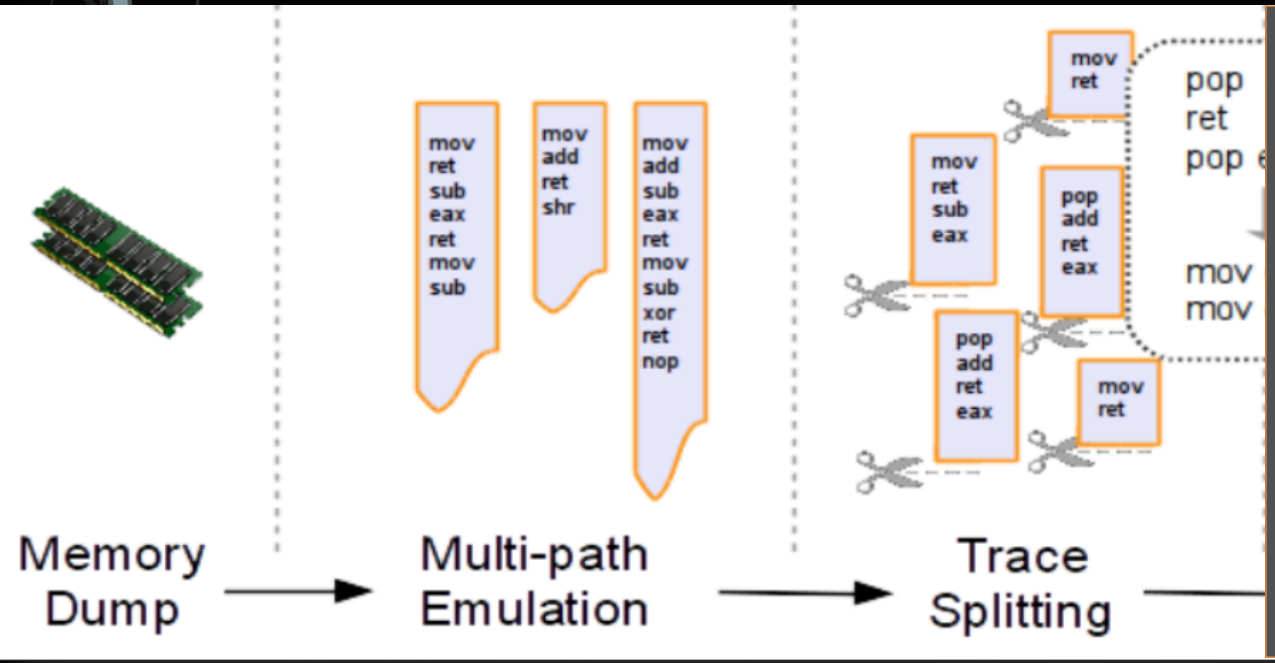
Memory Dump

# ROPMEMU


mov
ret
sub
eax
ret
mov
sub
mov
add
ret
shr
mov
add
sub
eax
ret
mov
sub
xor
ret
nop
Memory Dump
Multi-path Emulation

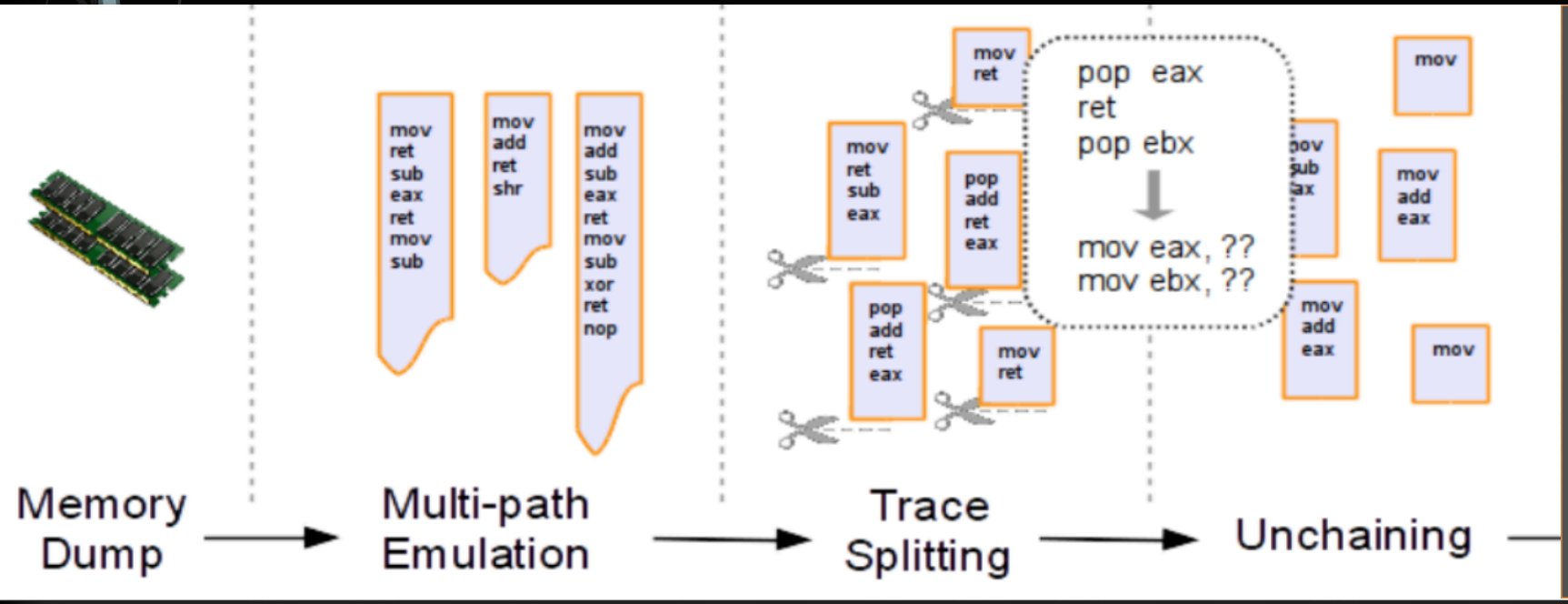
# DEMO 0x01

# ROPMEMU


mov ret sub eax ret mov sub
mov add ret shr
mov add sub eax ret mov sub xor ret nop
mov ret
mov ret sub eax
pop add ret eax
pop add ret eax
mov ret
pop ret pop
mov mov
Memory Dump
Multi-path Emulation
Trace Splitting

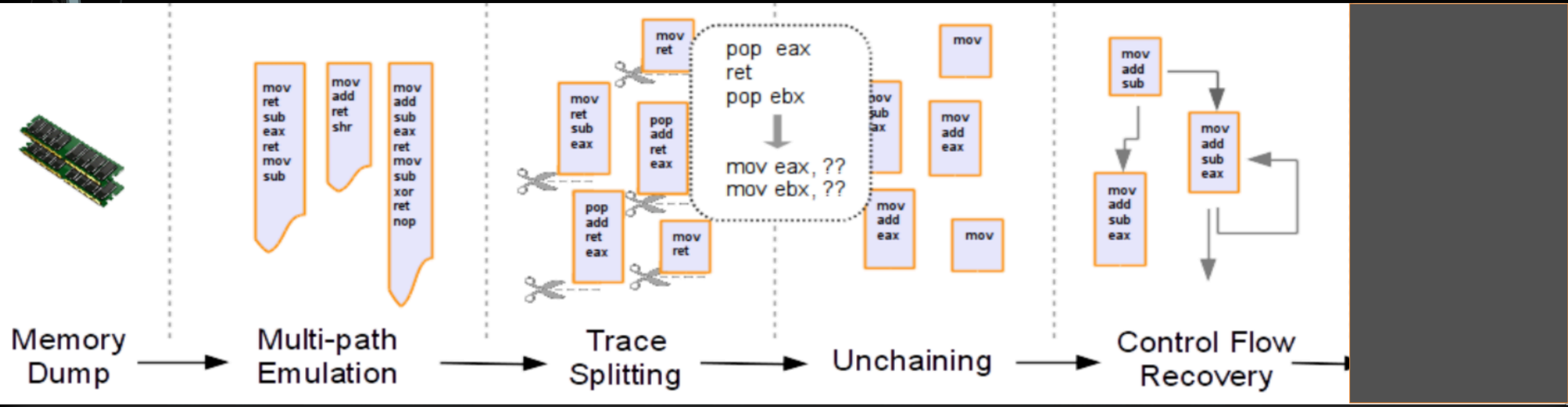
# DEMO 0x02

# ROPMEMU

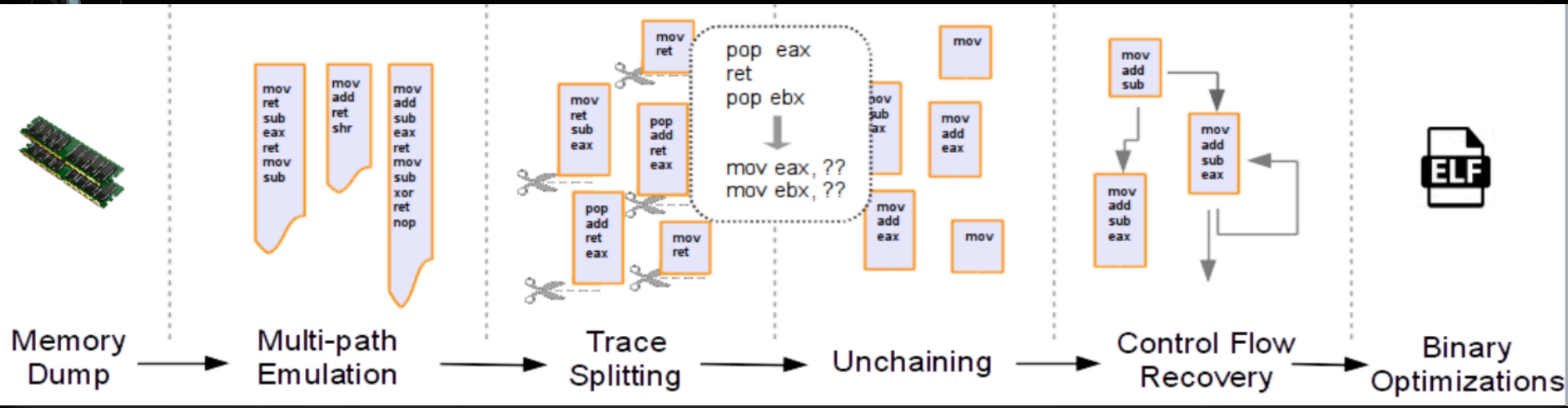

mov ret sub eax ret mov sub
mov add ret shr
mov add sub eax ret mov sub xor ret nop
mov ret
mov ret sub eax
pop add ret eax
pop add ret eax
mov ret
pop eax
ret
pop ebx
mov eax, ??
mov ebx, ??
mov
mov add eax
mov add eax
mov
Memory Dump
Multi-path Emulation
Trace Splitting
Unchaining

# DEMO 0x03

# ROPMEMU


mov ret sub eax ret mov sub
mov add ret shr
mov add sub eax ret mov sub xor ret nop
mov ret
mov ret sub eax
pop add ret eax
pop add ret eax
mov ret
pop eax
ret
pop ebx
mov eax, ??
mov ebx, ??
mov
mov add eax
mov add eax
mov
mov add sub
mov add sub eax
mov add sub eax
Memory Dump
Multi-path Emulation
Trace Splitting
Unchaining
Control Flow Recovery

# ROPMEMU

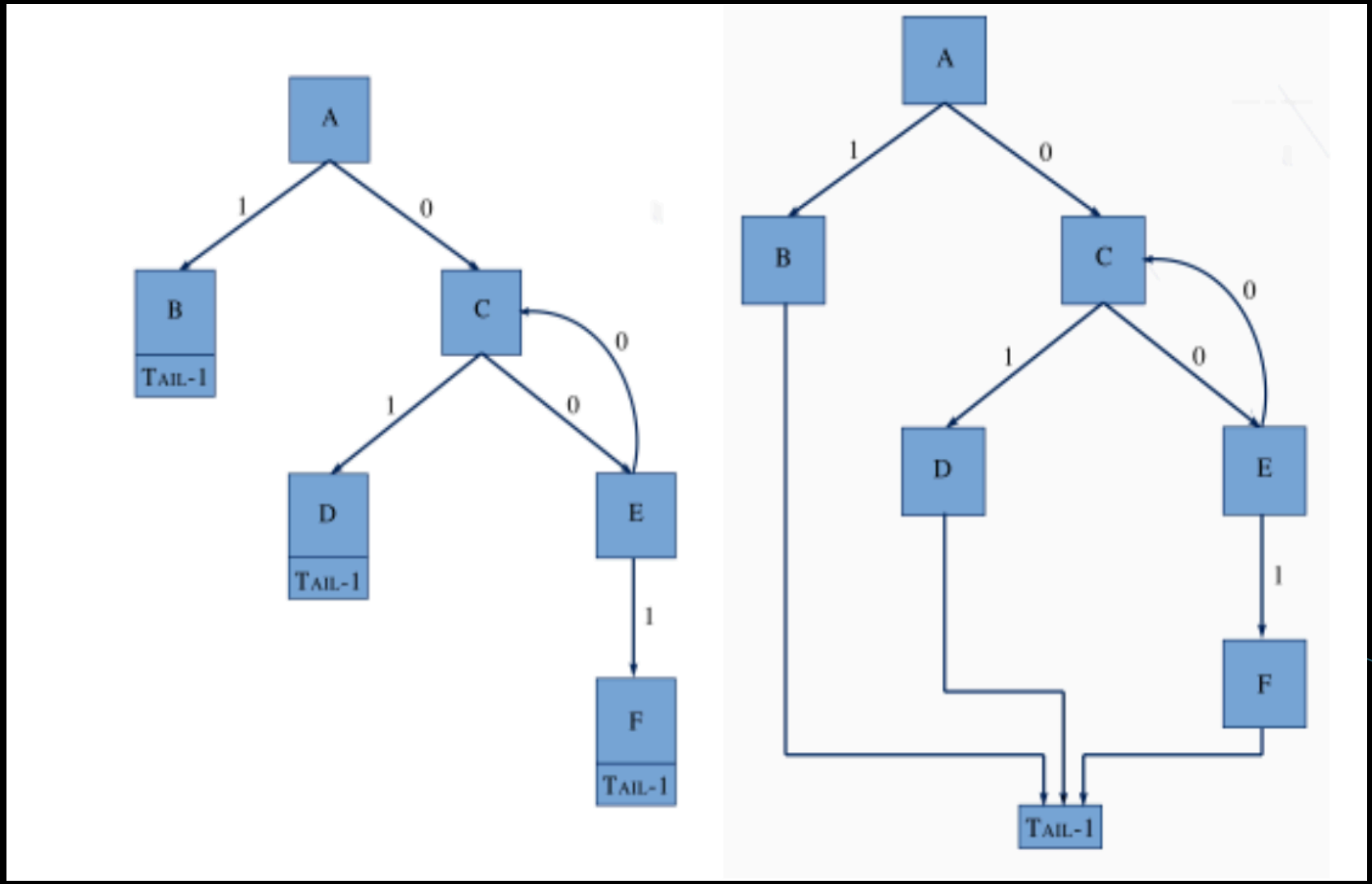

mov ret sub eax ret mov sub
mov add ret shr
mov add sub eax ret mov sub xor ret nop
mov ret
mov ret sub eax
pop add ret eax
pop add ret eax
mov ret
pop eax
ret
pop ebx
mov eax, ??
mov ebx, ??
mov
mov add eax
mov add eax
mov
mov add sub
mov add sub eax
mov add sub eax
ELF
Memory Dump
Multi-path Emulation
Trace Splitting
Unchaining
Control Flow Recovery
Binary Optimizations


# EXPERIMENT

| CHAINS | INSTRUCTIONS | GADGETS | BLOCKS | BRANCHES | FUNCTIONS | CALLS |
|---|---|---|---|---|---|---|
| COPY | 414275 | 184126 | 1 | - | - | - |
| DISPATCHER | 63515 | 18874 | 7 | 3 | 1 | 5 |
| PAYLOAD | 6320 | 2913 | 34 | 26 | 9 | 17 |

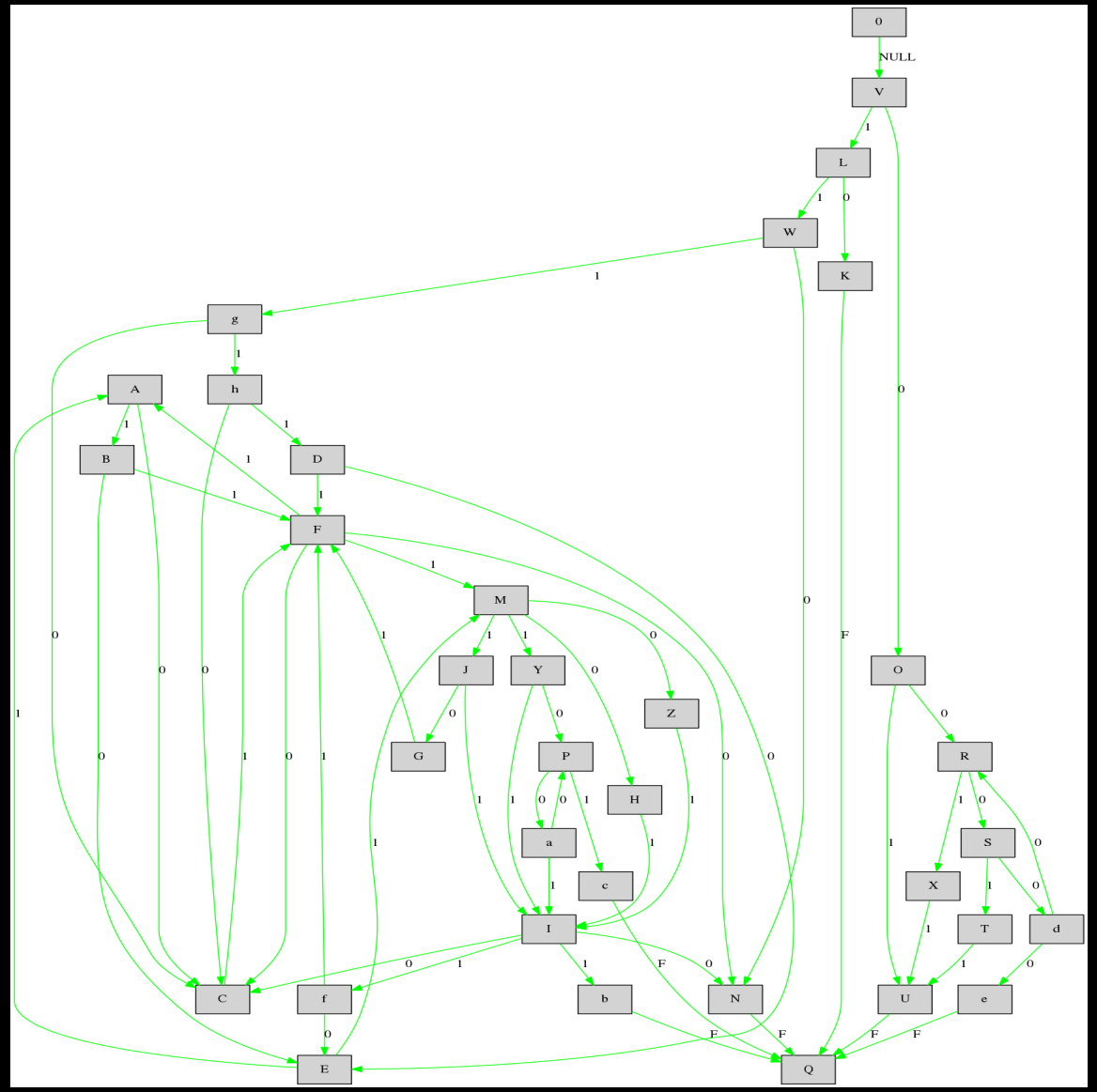
ZERONIGHTS

# RESULTS - CFG

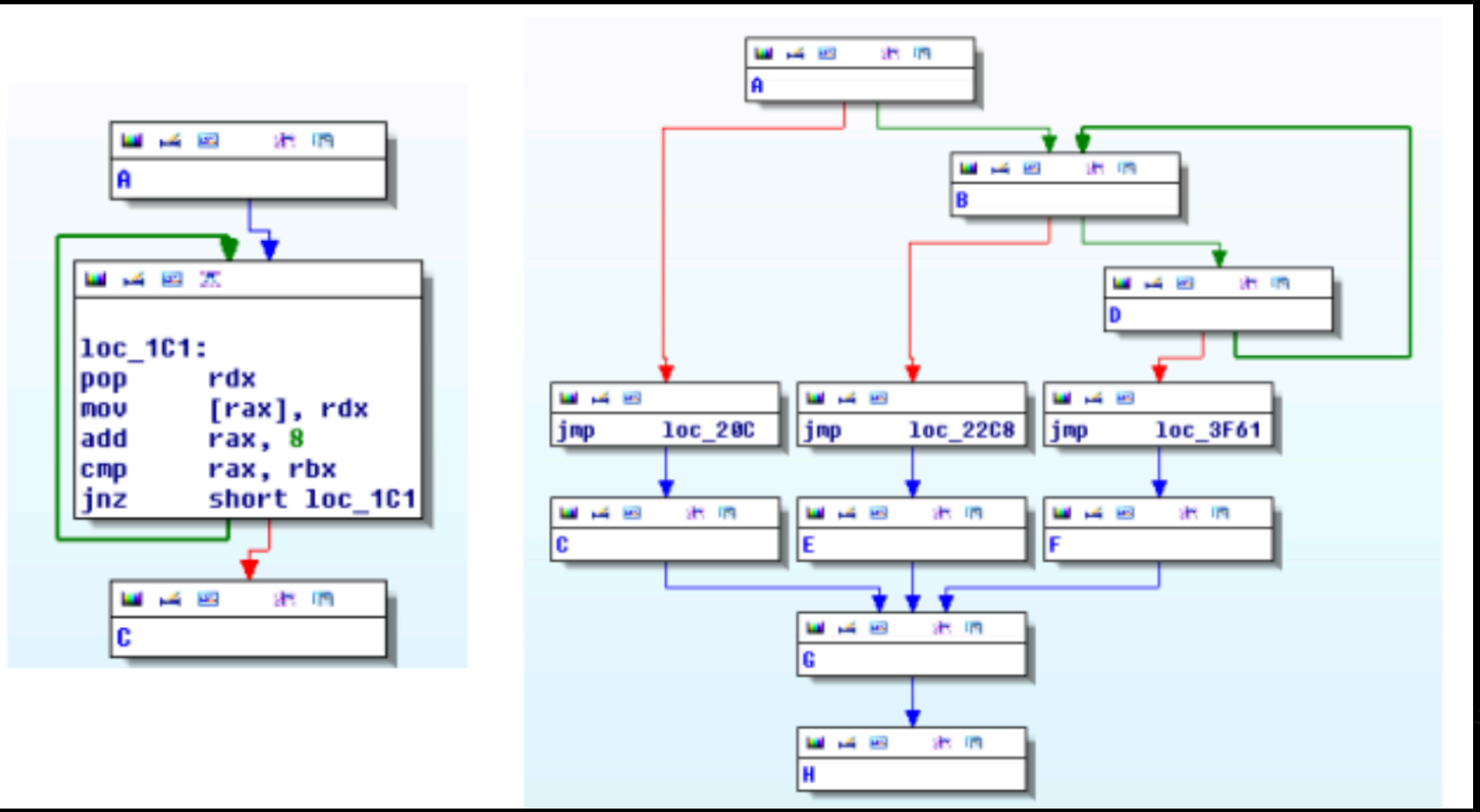


ZERONIGHTS

# RESULTS - CFG

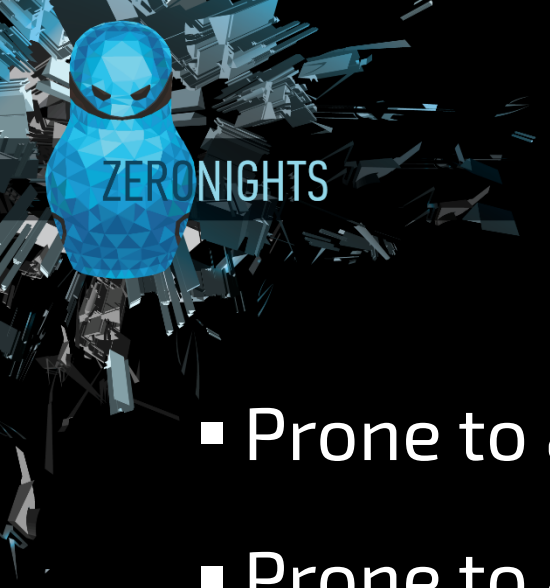


# RESULTS - IDA


A
loc_1C1:
pop rdx
mov [rax], rdx
add rax, 8
cmp rax, rbx
jnz short loc_1C1
C
A
B
D
jmp loc_20C
jmp loc_22C8
jmp loc_3F61
C
E
F
G
H




# DEMO 0x04




# LIMITATIONS

- Prone to anti-acquisition
- Prone to anti-emulation
- Lack of completeness on arbitrary inputs


# CONCLUSIONS

- Source code: https://github.com/vrtadmin/ROPMEMU
- First public tool to analyze code-reuse payloads
- Tested on the most complex public threat


# QUESTIONS?

Mariano Graziano

@emd3l

magrazia@cisco.com