ECE 6504

Embedded and Controls Systems Security: Stack Smashing V

Reading:

T:*kBouncer, ROPGuard, ROPecker,* and *Evaluating the Effectiveness of Current Anti-ROP Defenses*
N:*Security Based on Physical Unclonability and Disorder*

“Learn from the mistakes of others. You can never live long enough to make them all yourself.”

–Groucho Marx

stack protection methods:

1. make the overflow difficult to exploit
2. make overflow noticeable
3. make it impossible

stack protection methods:

1. make the overflow difficult to exploit
2. make overflow noticeable
3.make it impossible

detect ROP activity

stack protection methods:

1. make the overflow difficult to exploit
2. make overflow noticeable
3.make it impossible

assuming failure

detect ROP activity

1. fixed load addresses
2. shared libraries incompatible with ASLR
3. base address of a DLL can be either calculated via leaked pointer or brute-forced

an ideal ROP detector:

1. operates w/binaries (no source needed)
2. w/o debug symbols
3. does not insert code into calls
4. low overhead (time, memory)

an ideal ROP detector:

1. operates w/binaries (no source needed)
2. w/o debug symbols
3. does not insert code into calls
4. low overhead (time, memory)

	ROP Types	No Source Code	No Binary Rewriting	Run-time Efficiency
DROP [1]	Ret-based	√	X	X
ROPDefender [2]	Ret-based	√	X	X
ROPGuard [3]	Ret-based	√	X	√
Return-less Kernel [4]	Ret-based	X	√	√
CFLocking [5]	All	X	√	√
G-Free [6]	All	X	√	√
ILR [7]	All	√	X	√
Binary Stirring [8]	All	√	X	√
IPR [9]	All	√	X	√
CCFIR [10]	All	√	X	√
KBouncer [11]	All	√	X	√
ROPecker	All	√	√	√

an ideal ROP detector:

1. operates w/binaries (no source needed)
2. w/o debug symbols
3. does not insert code into calls
4. low overhead (time, memory)

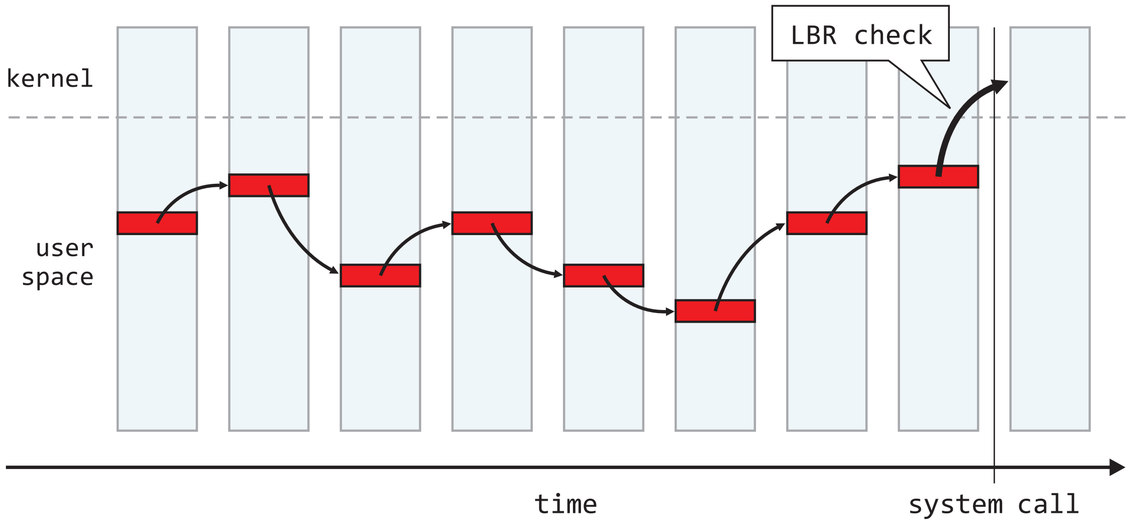
	ROP Types	No Source Code	No Binary Rewriting	Run-time Efficiency
DROP [1]	Ret-based	√	X	X
ROPDefender [2]	Ret-based	√	X	X
ROPGuard [3]	Ret-based	√	X	√
Return-less Kernel [4]	Ret-based	X	√	√
CFLocking [5]	All	X	√	√
G-Free [6]	All	X	√	√
ILR [7]	All	√	X	√
Binary Stirring [8]	All	√	X	√
IPR [9]	All	√	X	√
CCFIR [10]	All	√	X	√
KBouncer [11]	All	√	X	√
ROPecker	All	√	√	√

kBouncer

Pappas, V., Polychronakis, M. and Keromytis, A.D., 2013, August. Transparent ROP Exploit Mitigation Using Indirect Branch Tracing. In USENIX Security (Vol. 30, p. 38).

kBouncer

Pappas, V., Polychronakis, M. and Keromytis, A.D., 2013, August. Transparent ROP Exploit Mitigation Using Indirect Branch Tracing. In USENIX Security (Vol. 30, p. 38).

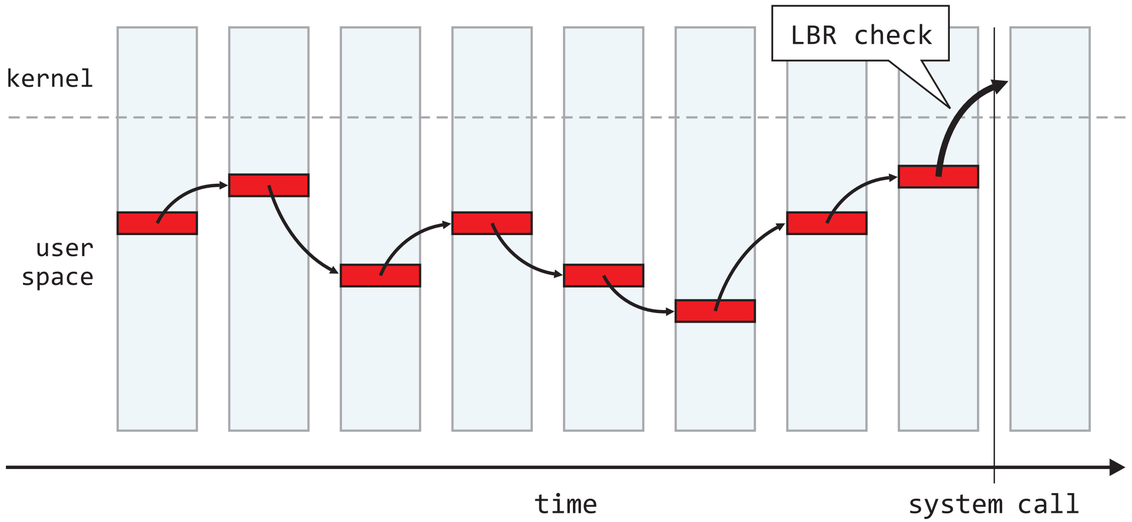


system call anatomy (UNIX)

1. indirect branches (red)
2. wrapper function for call (exposed directly)

kBouncer

Pappas, V., Polychronakis, M. and Keromytis, A.D., 2013, August. Transparent ROP Exploit Mitigation Using Indirect Branch Tracing. In USENIX Security (Vol. 30, p. 38).

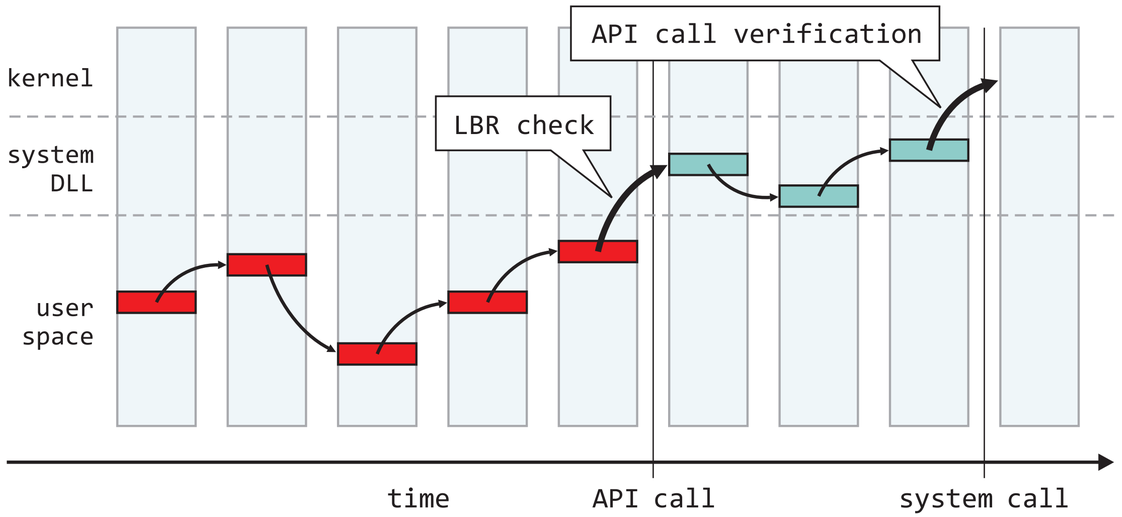


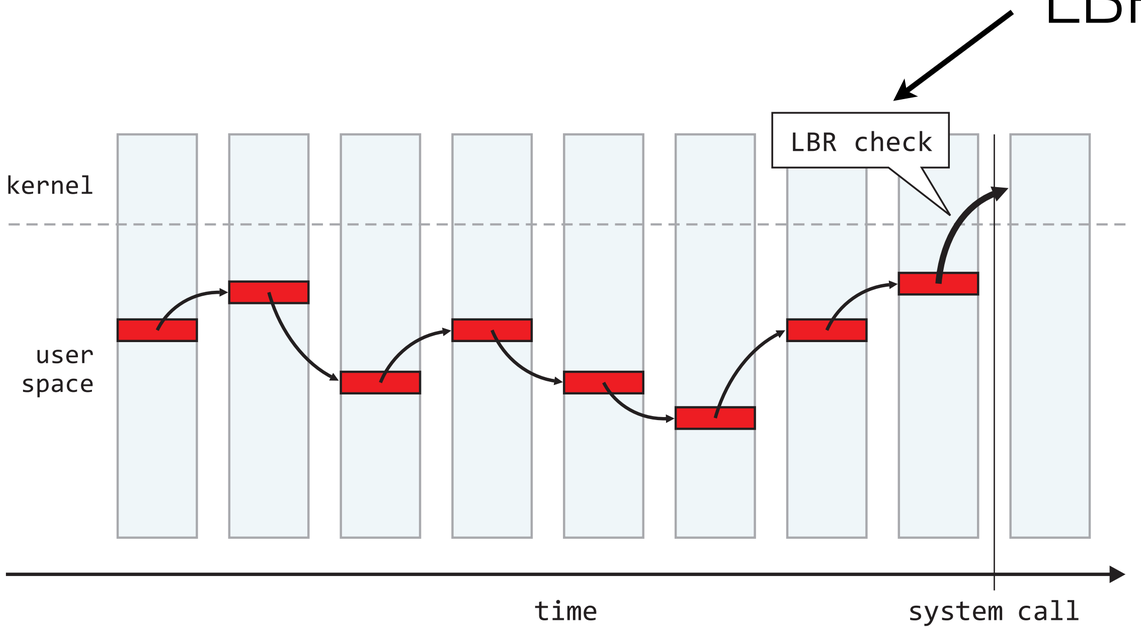
system call anatomy (UNIX)

1. indirect branches (red)
2. wrapper function for call (exposed directly)

system call anatomy (Windows)

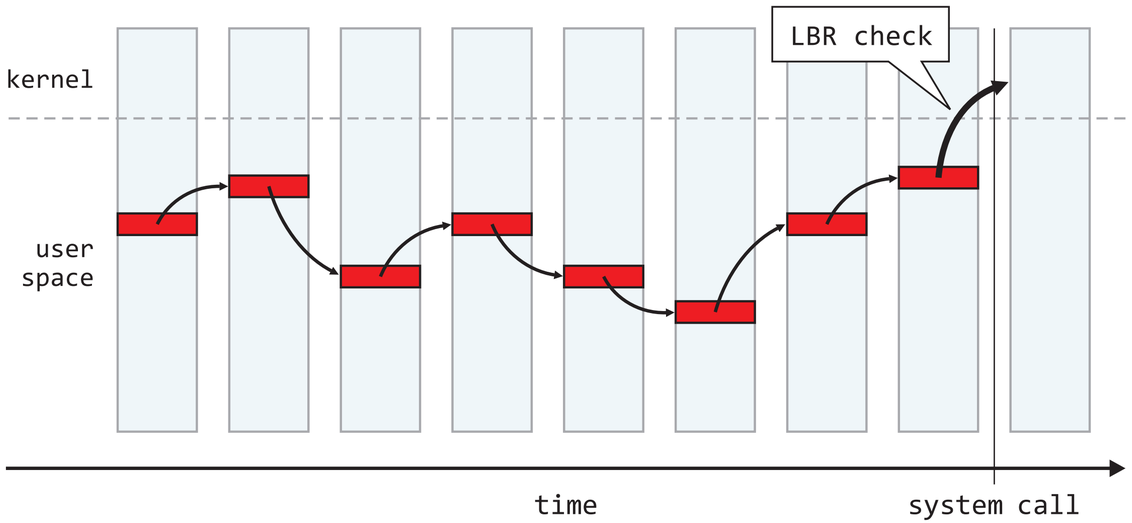
1. indirect branches
2. Windows API
3. Native API (not exposed)





LBR:

1. last branch record
2. hardware debug
3. last 16 branches
   a. address of
   b. target of
4. kernel-mode only

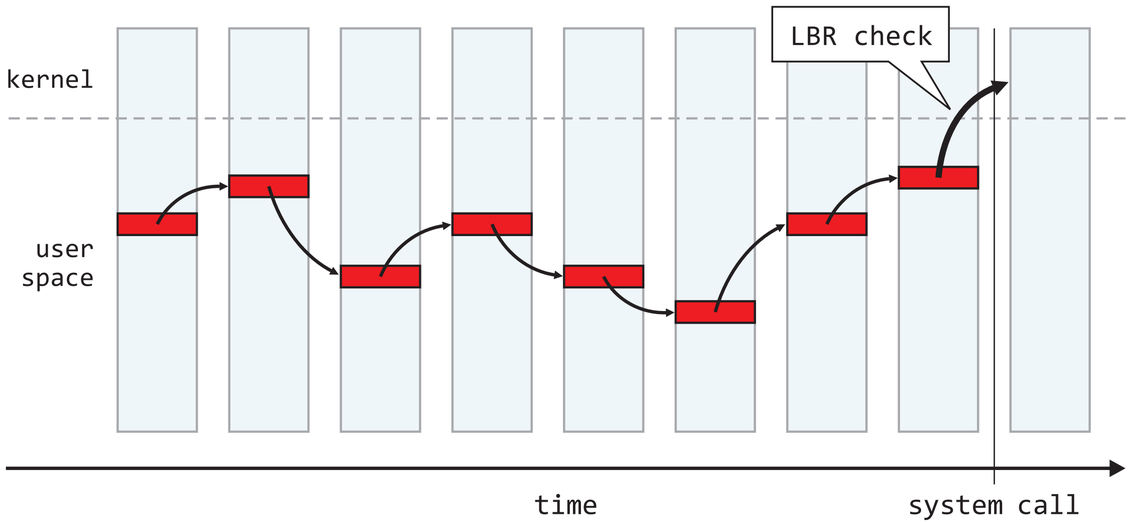


LBR:

1. last branch record
2. hardware debug
3. last 16 branches
   a. address of
   b. target of
4. kernel-mode only

ROP code:

1. many indirect calls
2. system call



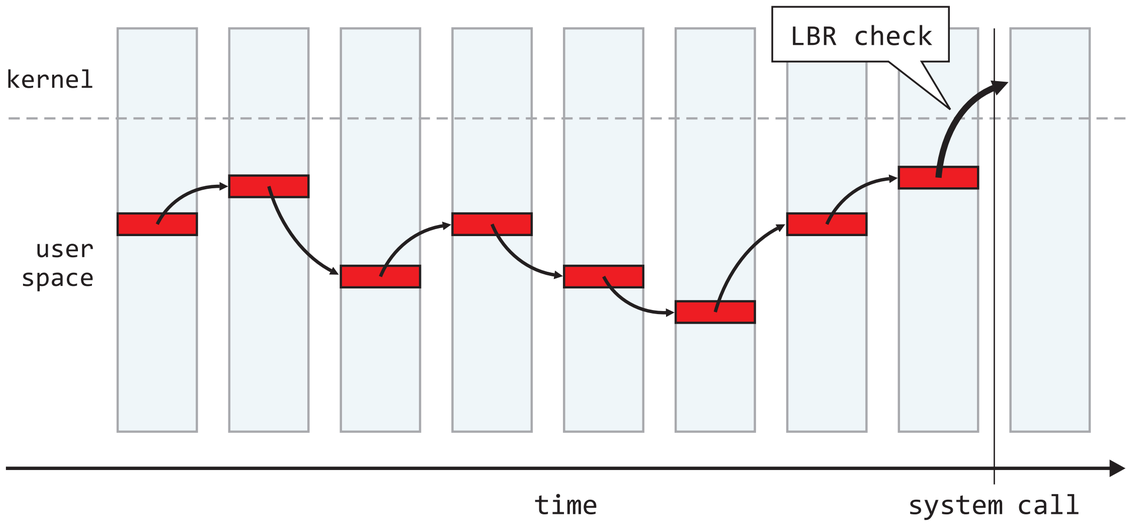
LBR:

1. last branch record
2. hardware debug
3. last 16 branches
   a. address of
   b. target of
4. kernel-mode only

ROP code:

1. many indirect calls
2. system call

→ however → non-malicious code does too



LBR:

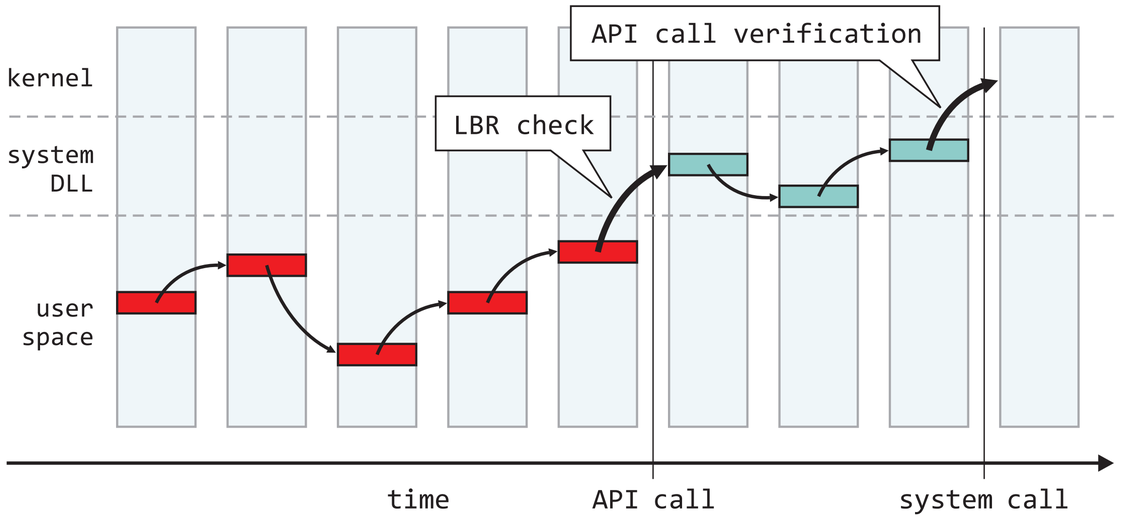
1. last branch record
2. hardware debug
3. last 16 branches
   a. address of
   b. target of
4. kernel-mode only

use LBR to distinguish

ROP code:

1. many indirect calls
2. system call

→ however → non-malicious code does too

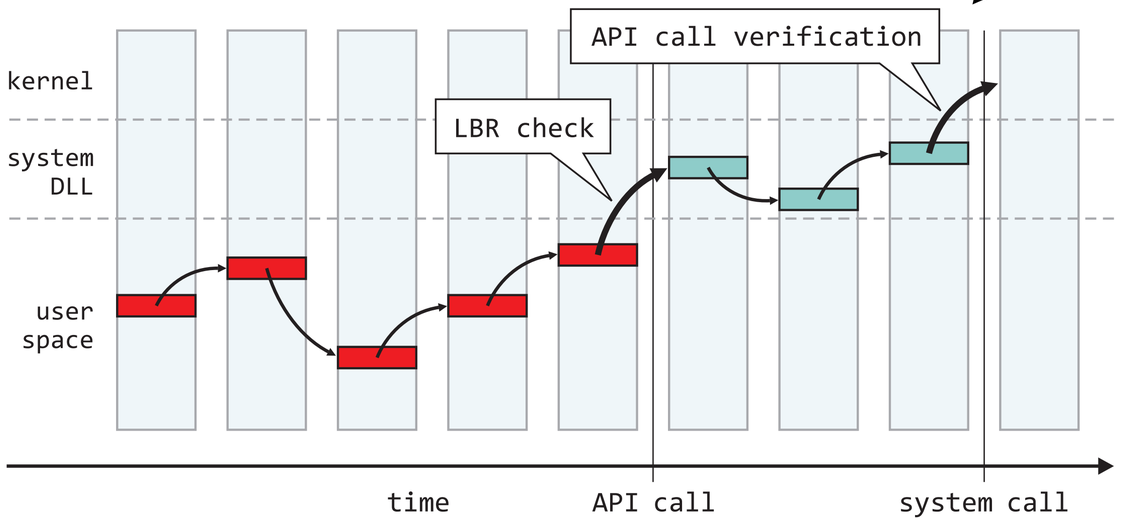


exploit code:

1. does not (usually) invoke system call directly
2. detection code runs at (select) API calls

if it does:

easy to verify that API hasn't made call



exploit code:

1. does not (usually) invoke system call directly
2. detection code runs at (select) API calls

ROP detection:

1. illegal returns
2. gadget chaining

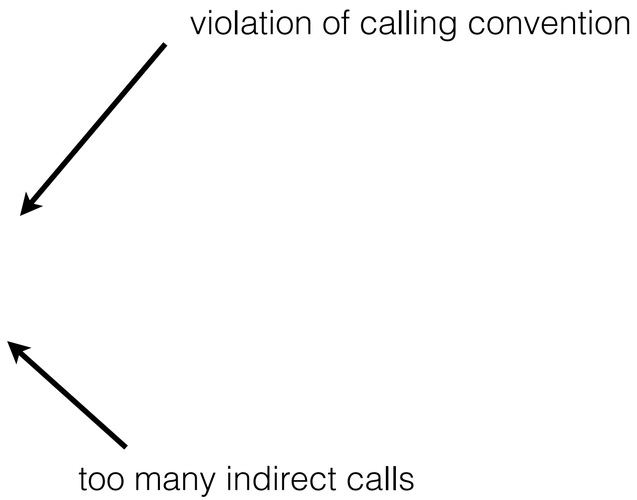
violation of calling convention

ROP detection:

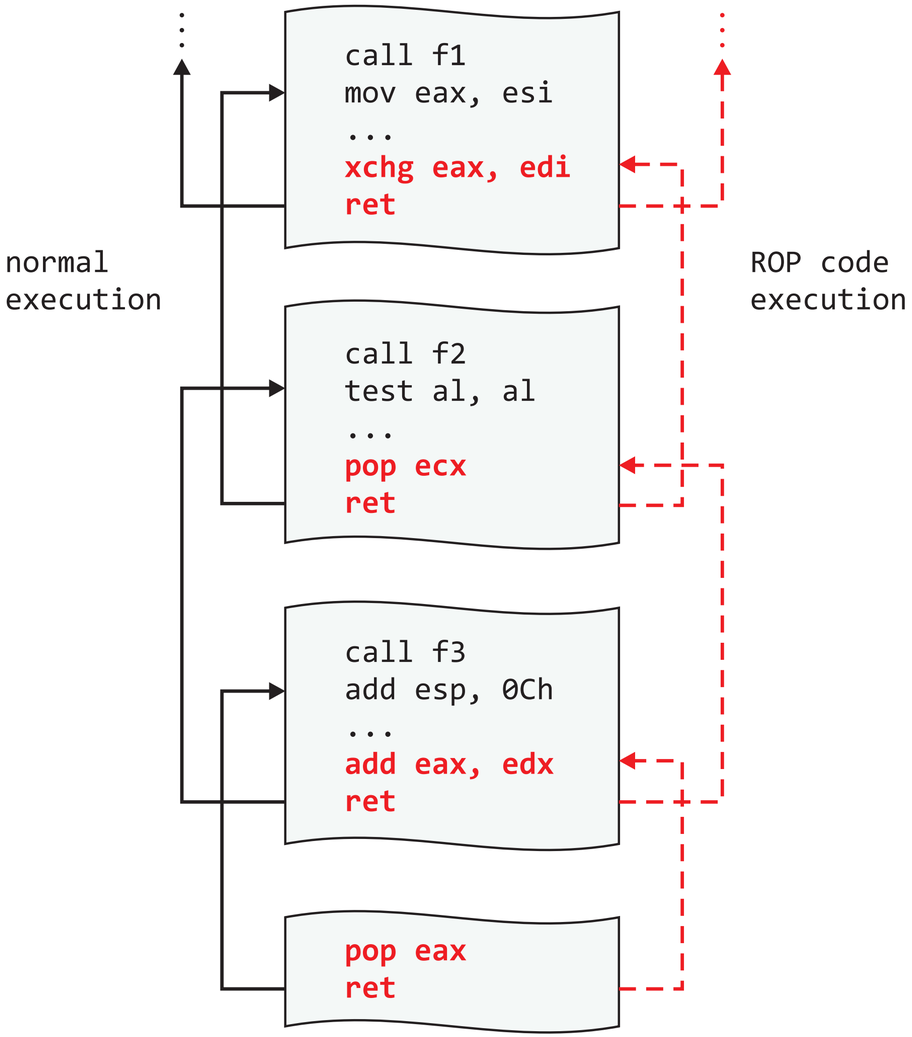
1. illegal returns
2. gadget chaining

ROP detection:

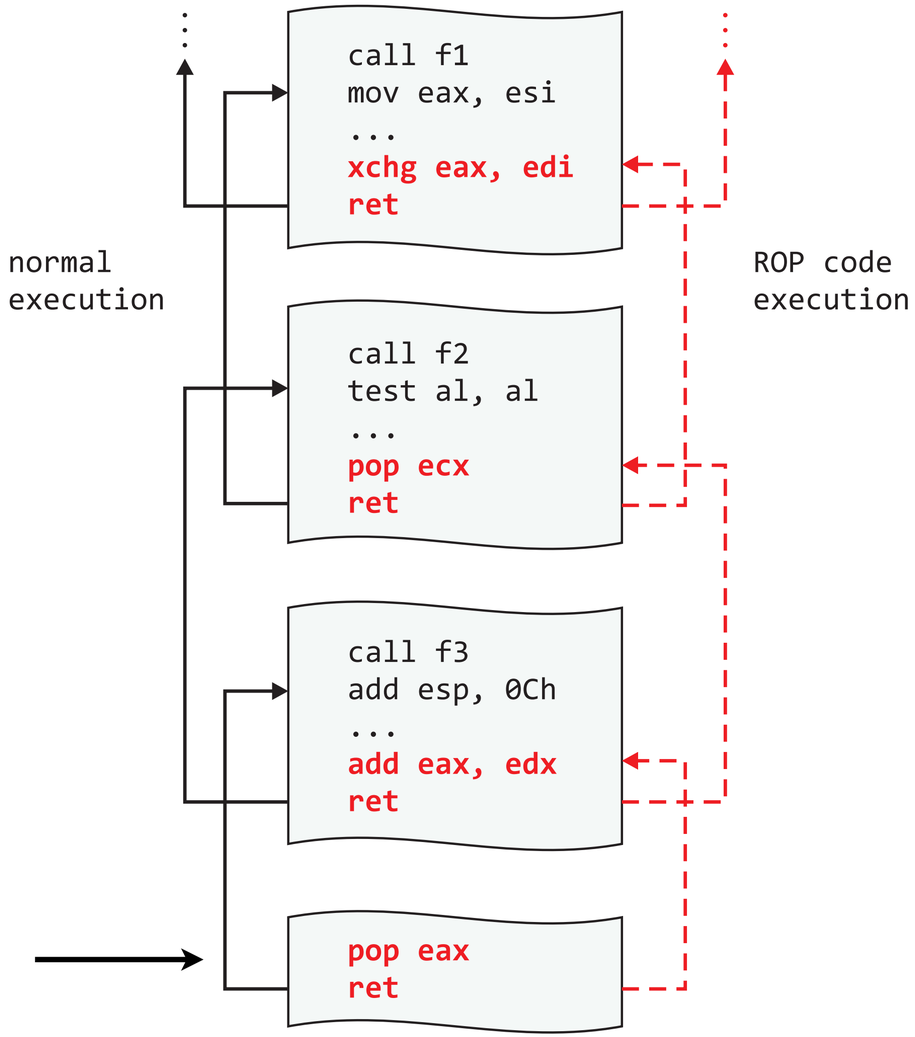
1. illegal returns
2. gadget chaining



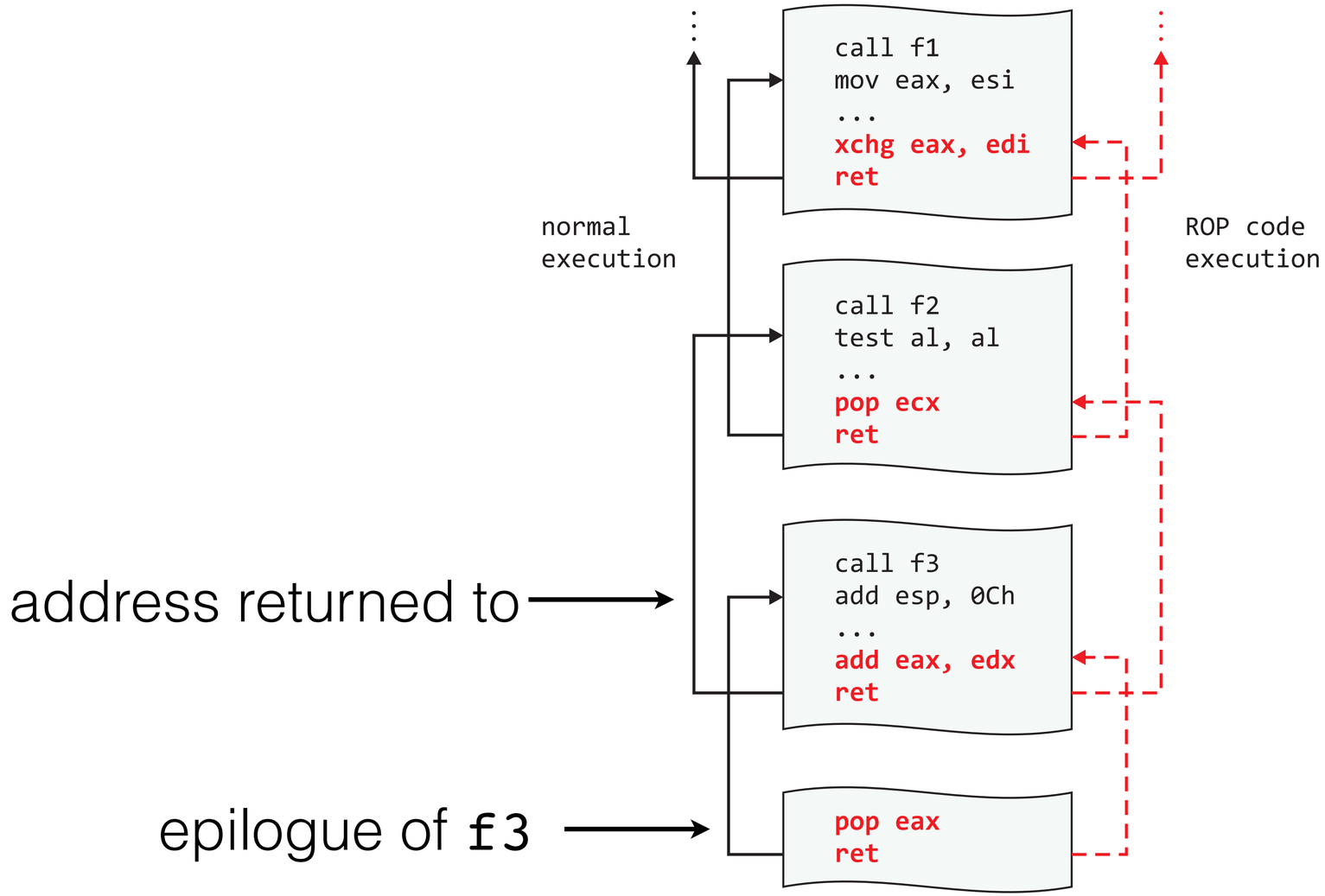
ROP detection: illegal returns



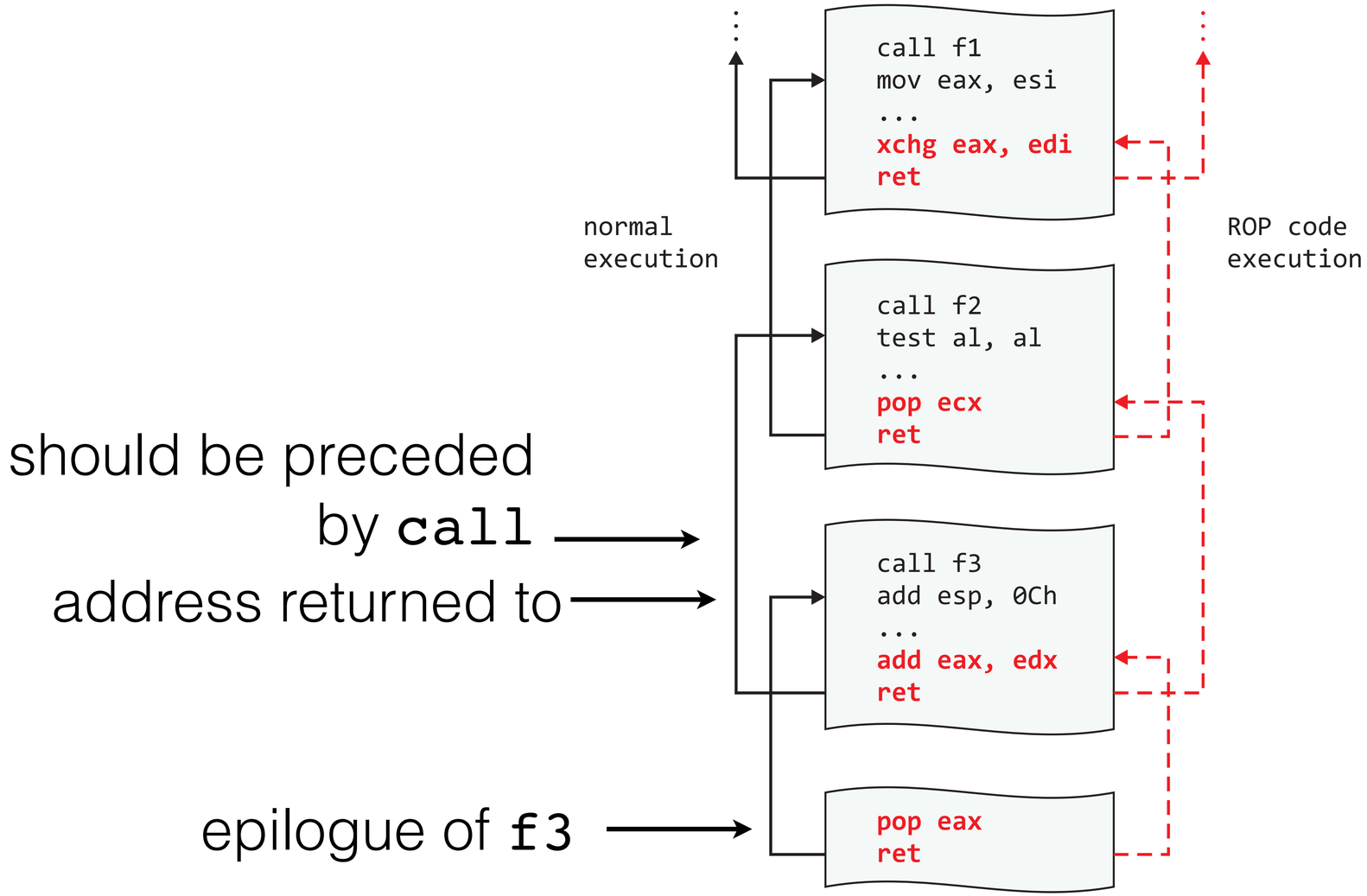
ROP detection: illegal returns



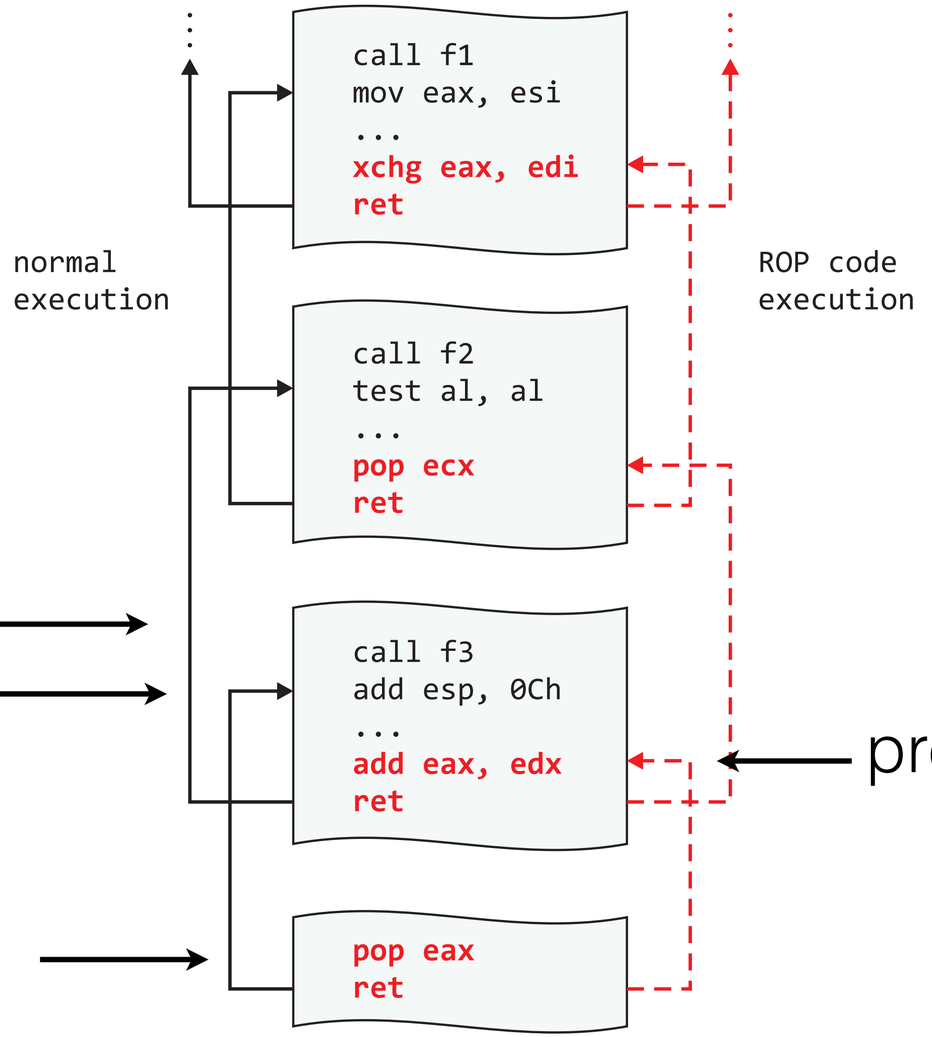
ROP detection: illegal returns



ROP detection: illegal returns



ROP detection: illegal returns

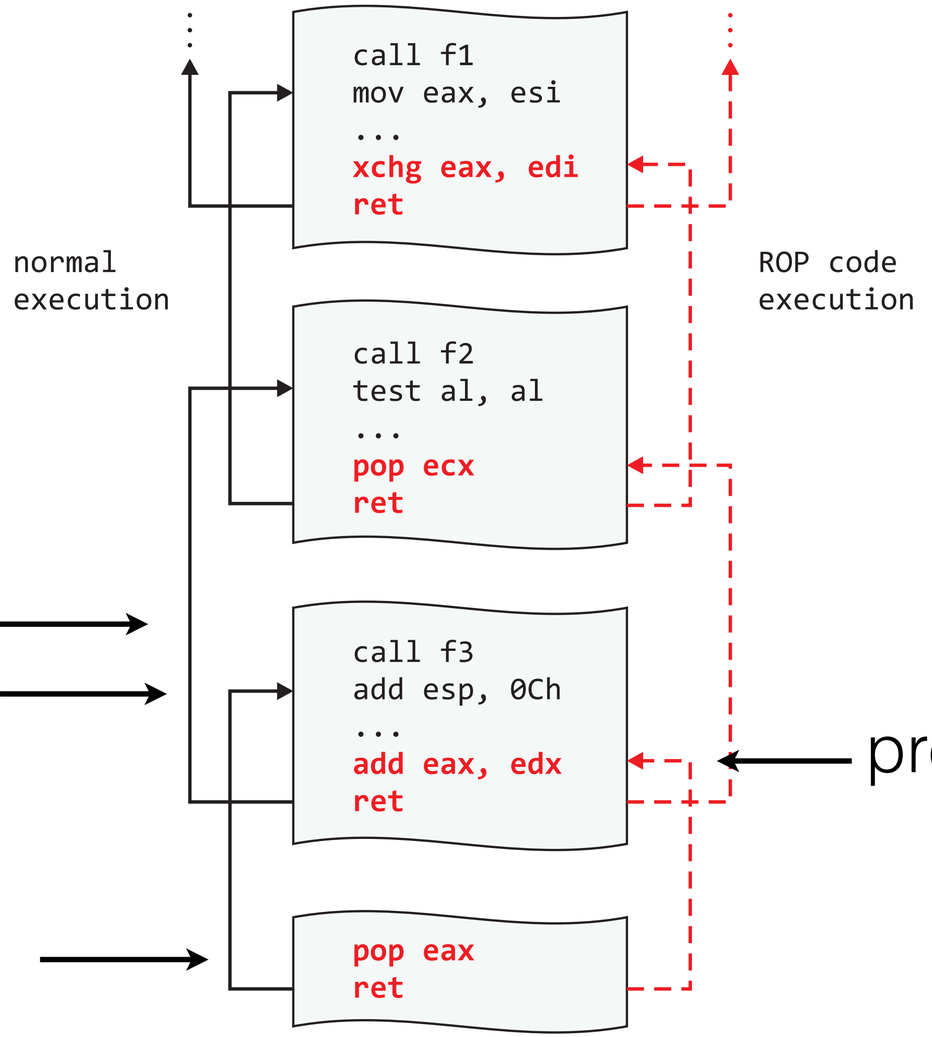


should be preceded by **`call`** → address returned to

epilogue of **`f3`** →

← ROP return to address, not preceded by **`call`**

ROP detection: illegal returns



should be preceded by **`call`**

address returned to

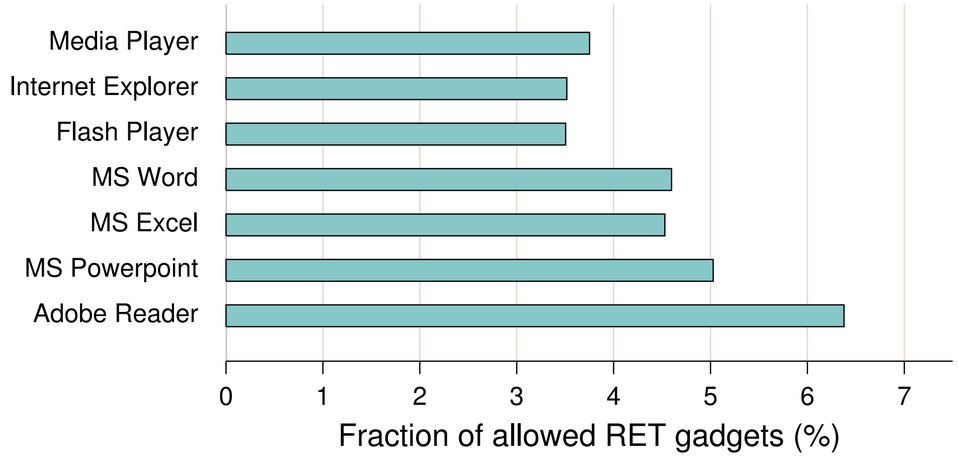
epilogue of **`f3`**

ROP return to address, not preceded by **`call`**

Q: what about gadgets that begin at **`call`**?

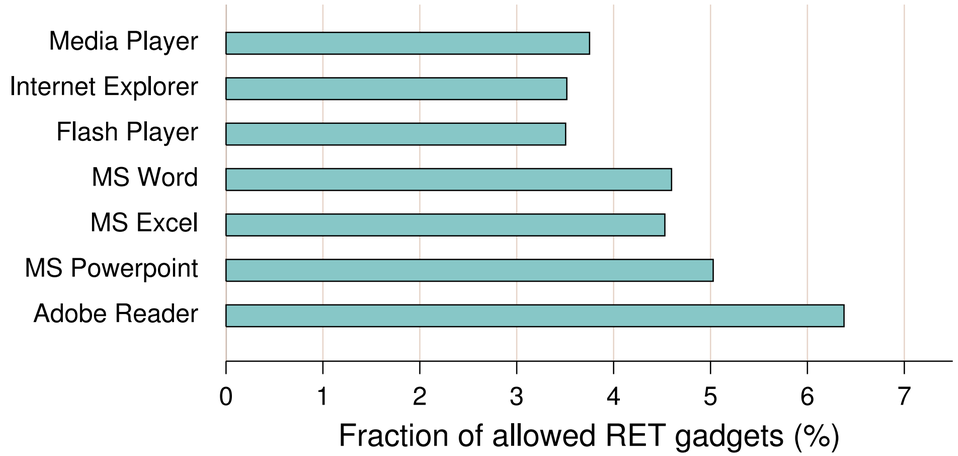
prevalence of `call`-preceded gadgets:

1. `call`
2. up-to 20 instructions
3. `ret`



prevalence of **`call`**-preceded gadgets:

1. **`call`**
2. up-to 20 instructions
3. **`ret`**



1. thought to be too few
2. non-sense instructions should affect ROP program state

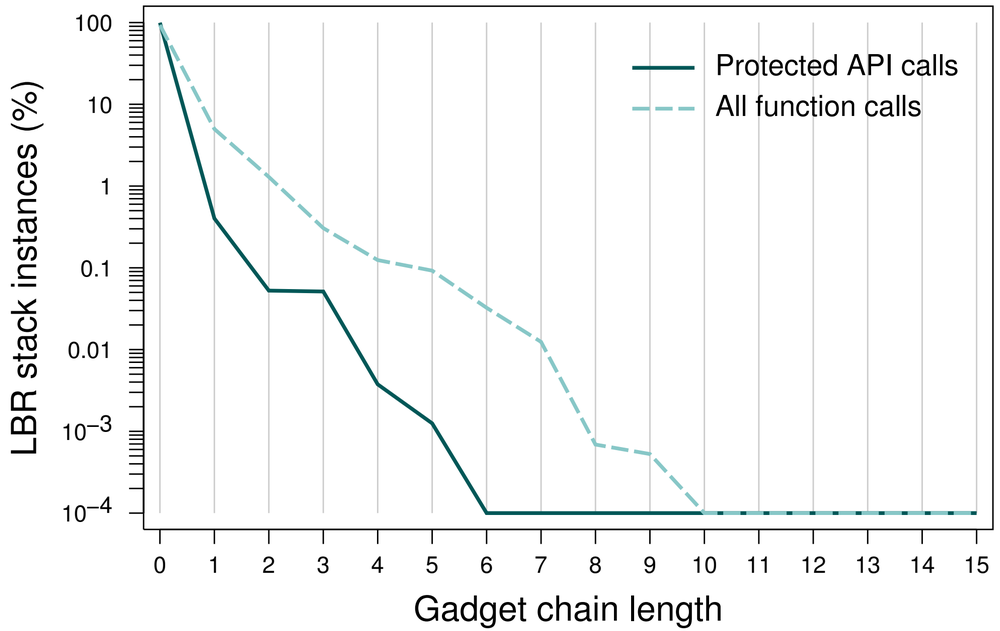
ROP detection: gadget chaining

ROP detection: gadget chaining

sequence of indirect calls, with fewer than 20 instructions between call and return

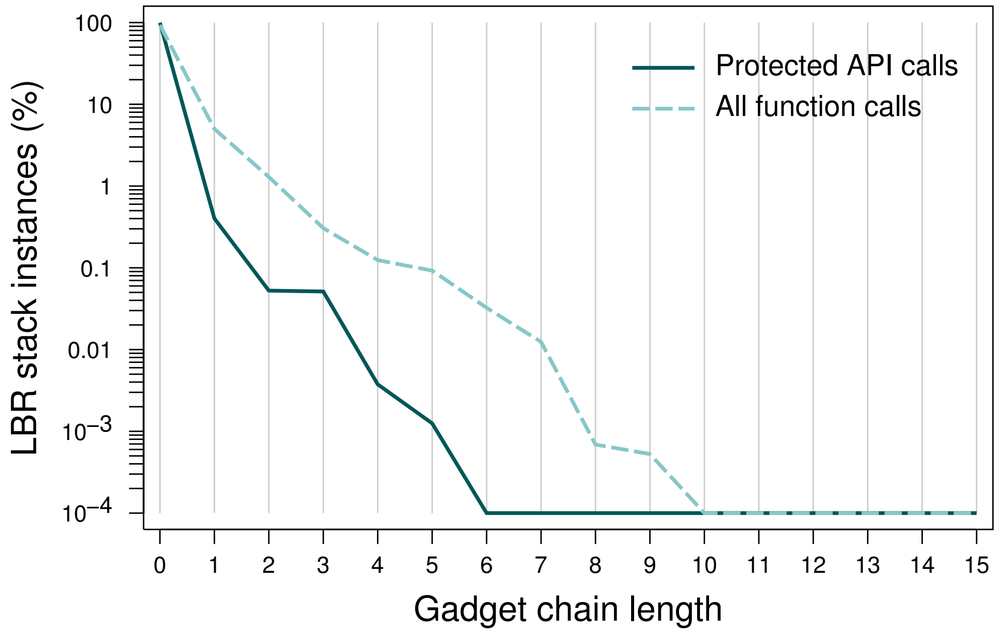
ROP detection: gadget chaining

sequence of indirect calls, with fewer than 20 instructions between call and return



ROP detection: gadget chaining

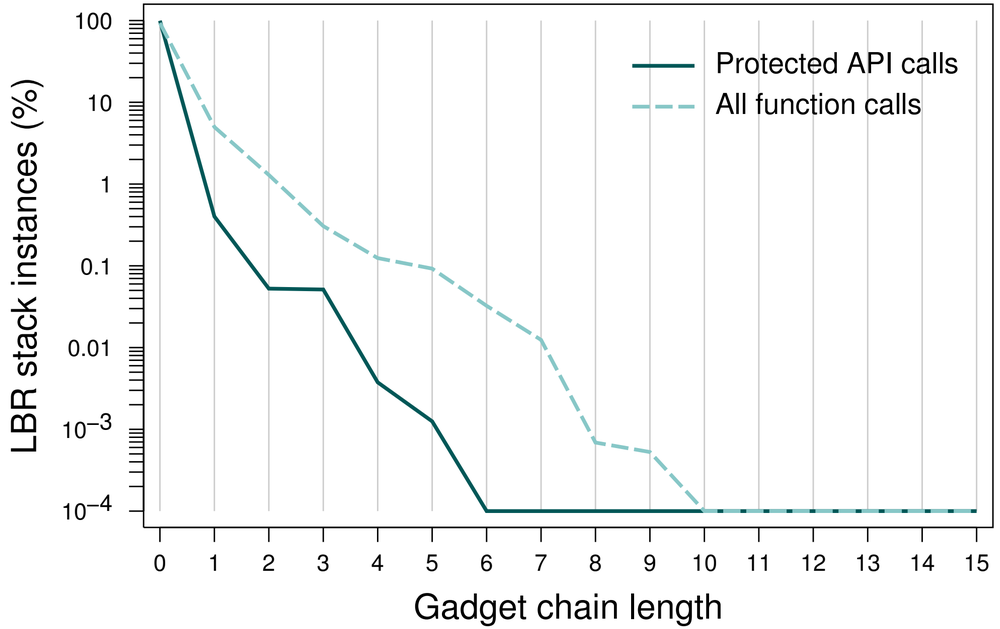
sequence of indirect calls, with fewer than 20 instructions between call and return



most APIs won't exhaust LBR buffer

ROP detection: gadget chaining

sequence of indirect calls, with fewer than 20 instructions between call and return

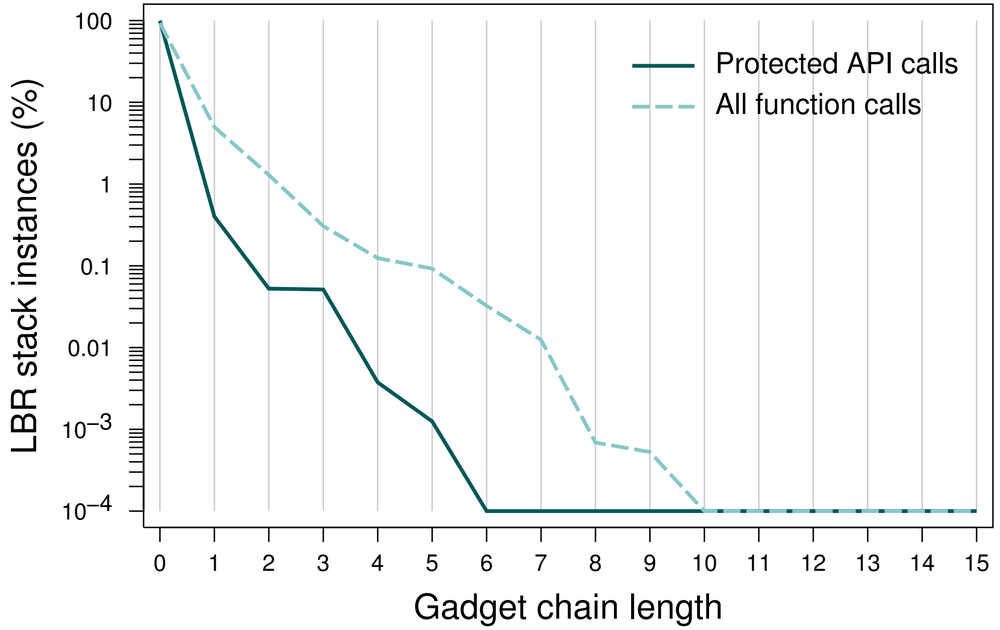


most APIs won't exhaust LBR buffer

but, gadgets → require many more indirects

ROP detection: gadget chaining

sequence of indirect calls, with fewer than 20 instructions between call and return



LBR Stack

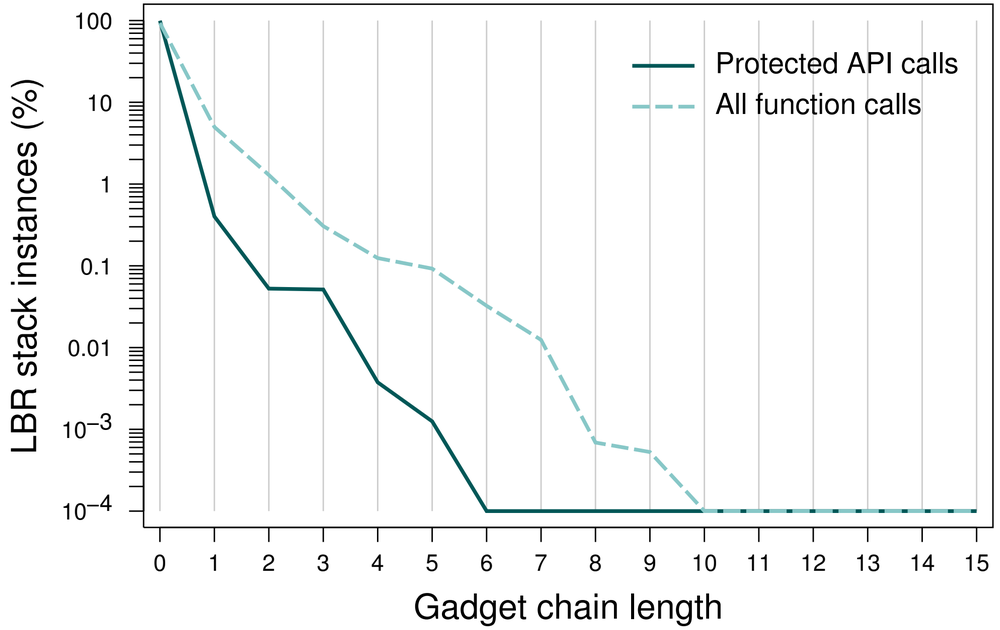
	Branch	Target	
00	73802745	738028D7	
01	05F015CA	05F00E17	
02	7C348B06	7C34A028	G01
03	7C34A02A	7C34252C	G02
04	7C34252D	7C36C55A	G03
05	7C36C55B	7C345249	G04
06	7C34524A	7C3411C0	G05
07	7C3411C1	7C34B8D7	G06
08	7C34B8D8	7C366FA6	G07
09	7C366FA7	7C3762FB	G08
10	7C3762FC	7C378C81	G09
11	7C378C84	7C346C0B	G10
12	7C346C0B	7C3415A2	G11
13	7C3415A2	74F64347	
14	74F64908	752AD0A1	
15	752D6FC8	752AD0AD	

most APIs won't exhaust LBR buffer

but, gadgets → require many more indirects

ROP detection: gadget chaining

sequence of indirect calls, with fewer than 20 instructions between call and return



end of gadget

LBR Stack

	Branch	Target	
00	73802745	738028D7	
01	05F015CA	05F00E17	
02	7C348B06	7C34A028	G01
03	7C34A02A	7C34252C	G02
04	7C34252D	7C36C55A	G03
05	7C36C55B	7C345249	G04
06	7C34524A	7C3411C0	G05
07	7C3411C1	7C34B8D7	G06
08	7C34B8D8	7C366FA6	G07
09	7C366FA7	7C3762FB	G08
10	7C3762FC	7C378C81	G09
11	7C378C84	7C346C0B	G10
12	7C346C0B	7C3415A2	G11
13	7C3415A2	74F64347	
14	74F64908	752AD0A1	
15	752D6FC8	752AD0AD	

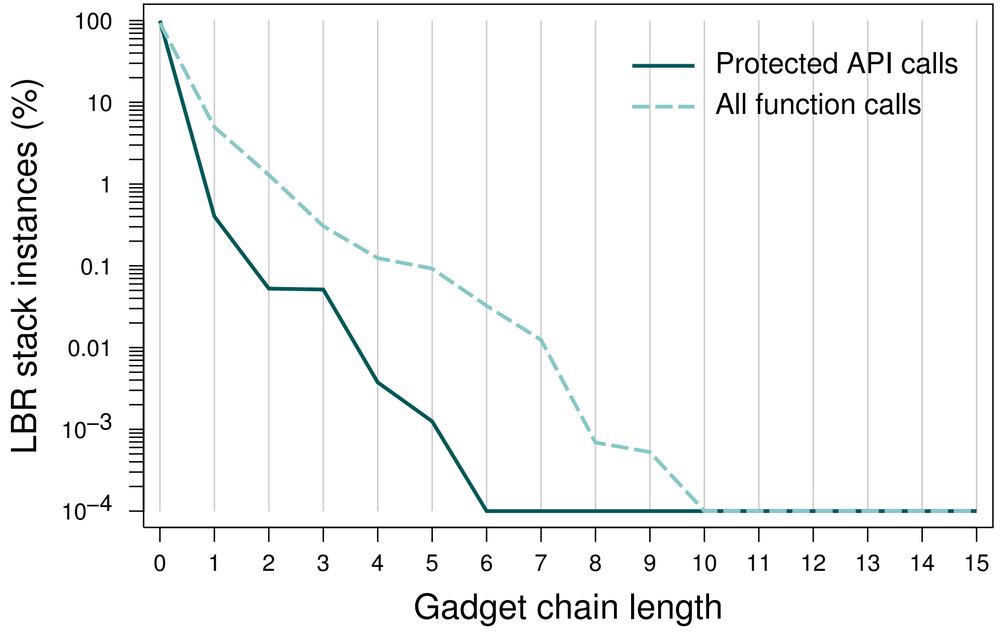
most APIs won't exhaust LBR buffer

but, gadgets

require many more indirects

ROP detection: gadget chaining

sequence of indirect calls, with fewer than 20 instructions between call and return



most APIs won't exhaust LBR buffer

but, gadgets require many more indirects

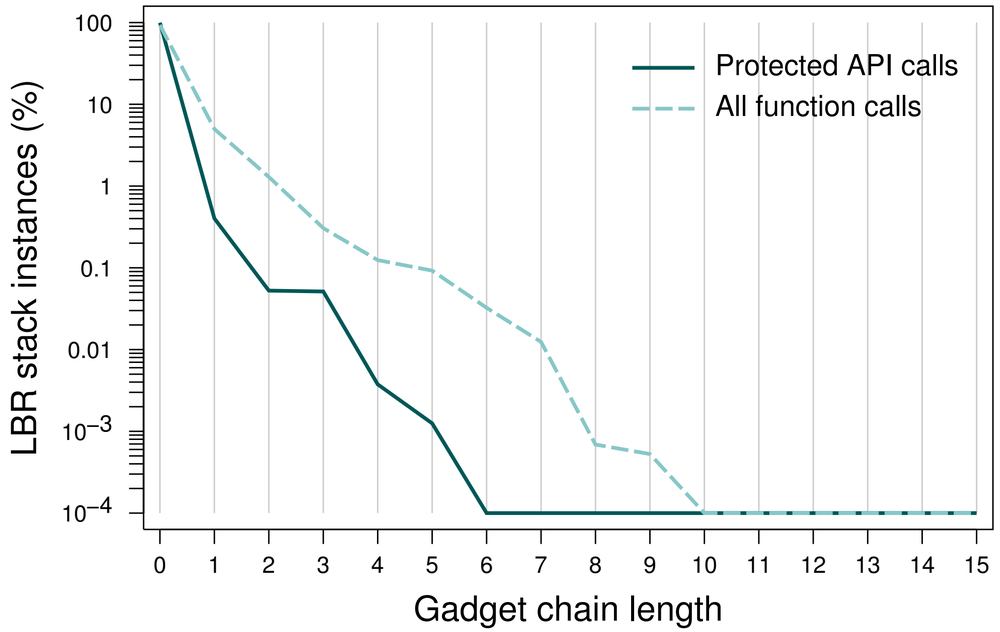
LBR Stack

end of gadget → Branch | Target ← beginning of gadget

	Branch	Target	
00	73802745	738028D7	
01	05F015CA	05F00E17	
02	7C348B06	7C34A028	G01
03	7C34A02A	7C34252C	G02
04	7C34252D	7C36C55A	G03
05	7C36C55B	7C345249	G04
06	7C34524A	7C3411C0	G05
07	7C3411C1	7C34B8D7	G06
08	7C34B8D8	7C366FA6	G07
09	7C366FA7	7C3762FB	G08
10	7C3762FC	7C378C81	G09
11	7C378C84	7C346C0B	G10
12	7C346C0B	7C3415A2	G11
13	7C3415A2	74F64347	
14	74F64908	752AD0A1	
15	752D6FC8	752AD0AD	

ROP detection: gadget chaining

sequence of indirect calls, with fewer than 20 instructions between call and return



most APIs won't exhaust LBR buffer

but, gadgets require many more indirects

LBR Stack

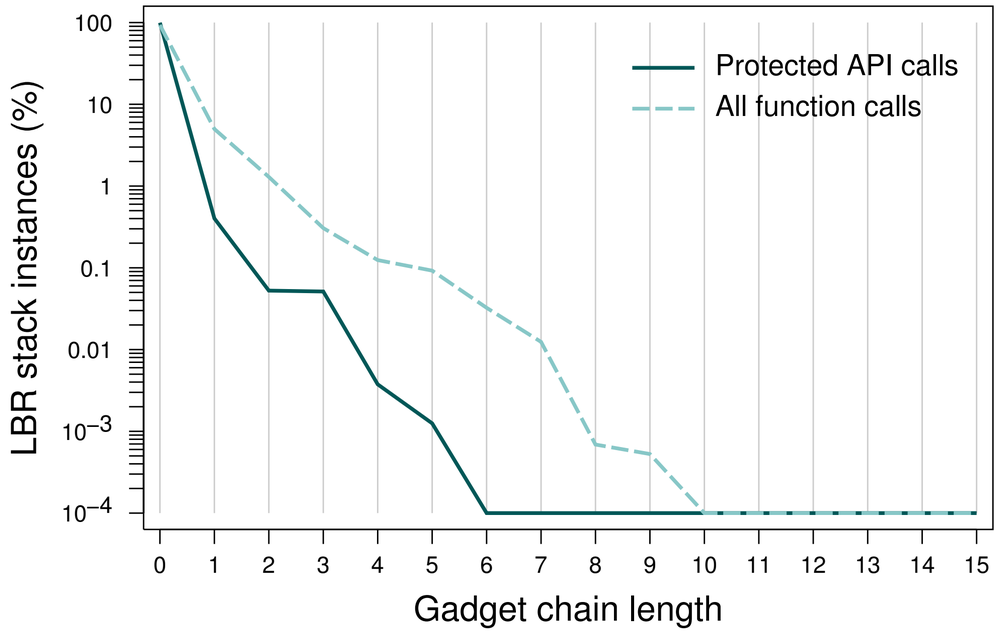
end of gadget → Branch | Target ← beginning of gadget

	Branch	Target	
00	73802745	738028D7	
01	05F015CA	05F00E17	
02	7C348B06	7C34A028	G01
03	7C34A02A	7C34252C	G02
04	7C34252D	7C36C55A	G03
05	7C36C55B	7C345249	G04
06	7C34524A	7C3411C0	G05
07	7C3411C1	7C34B8D7	G06
08	7C34B8D8	7C366FA6	G07
09	7C366FA7	7C3762FB	G08
10	7C3762FC	7C378C81	G09
11	7C378C84	7C346C0B	G10
12	7C346C0B	7C3415A2	G11
13	7C3415A2	74F64347	
14	74F64908	752AD0A1	
15	752D6FC8	752AD0AD	

fewer than 20 instructions, so gadget

ROP detection: gadget chaining

sequence of indirect calls, with fewer than 20 instructions between call and return



most APIs won't exhaust LBR buffer

but, gadgets require many more indirects

LBR Stack

end of gadget → Branch | Target ← beginning of gadget

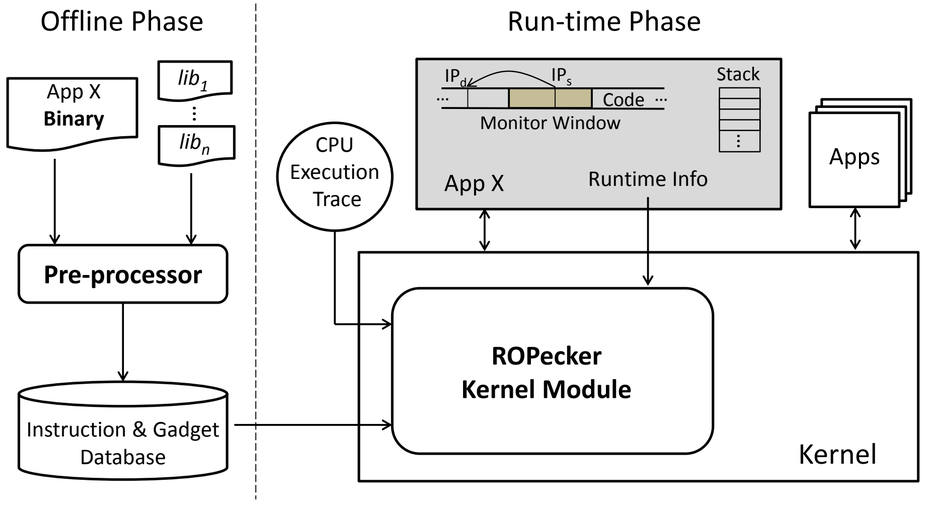
	Branch	Target	
00	73802745	738028D7	
01	05F015CA	05F00E17	
02	7C348B06	7C34A028	G01
03	7C34A02A	7C34252C	G02
04	7C34252D	7C36C55A	G03
05	7C36C55B	7C345249	G04
06	7C34524A	7C3411C0	G05
07	7C3411C1	7C34B8D7	G06
08	7C34B8D8	7C366FA6	G07
09	7C366FA7	7C3762FB	G08
10	7C3762FC	7C378C81	G09
11	7C378C84	7C346C0B	G10
12	7C346C0B	7C3415A2	G11
13	7C3415A2	74F64347	
14	74F64908	752AD0A1	
15	752D6FC8	752AD0AD	

fewer than 20 instructions, so gadget

ROP activity, if > 8

ROPecker

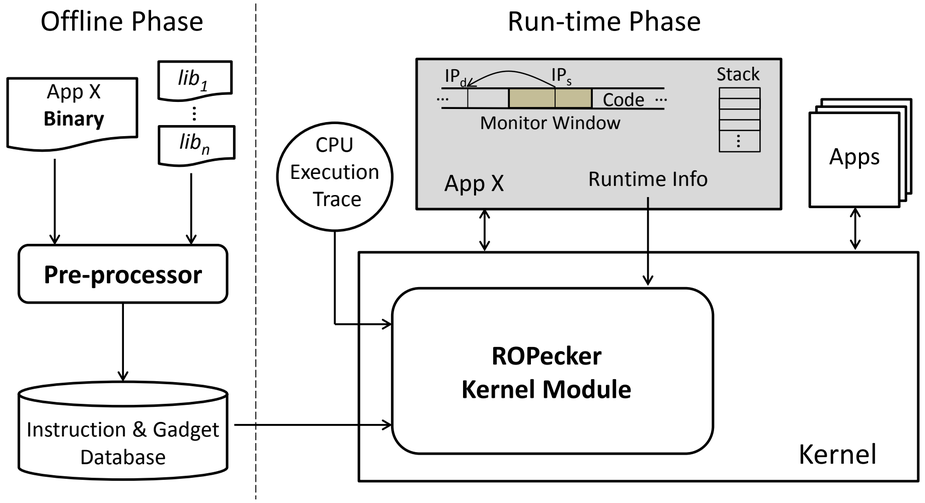
Cheng, Y., Zhou, Z., Miao, Y., Ding, X. and DENG, H., Robert. ROPecker: A Generic and Practical Approach For Defending Against ROP Attack.(2014). In Proceedings of the 21th Annual Network and Distributed System Security Symposium (NDSS'14), February 23-26, 2014, San Diago, CA.



ROPecker

Cheng, Y., Zhou, Z., Miao, Y., Ding, X. and DENG, H., Robert. ROPecker: A Generic and Practical Approach For Defending Against ROP Attack.(2014). In Proceedings of the 21th Annual Network and Distributed System Security Symposium (NDSS'14), February 23-26, 2014, San Diago, CA.

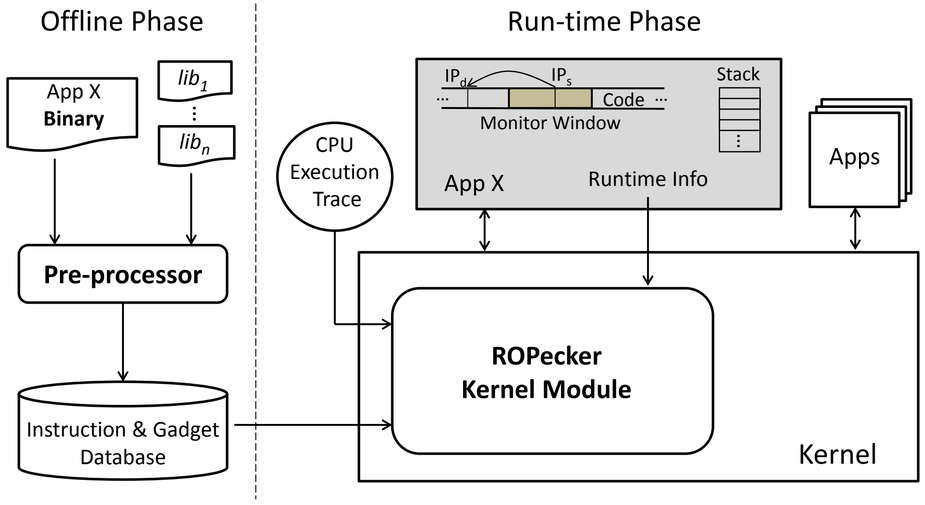
1. uses LBR, as kBouncer; past ROP activity



ROPecker

Cheng, Y., Zhou, Z., Miao, Y., Ding, X. and DENG, H., Robert. ROPecker: A Generic and Practical Approach For Defending Against ROP Attack.(2014). In Proceedings of the 21th Annual Network and Distributed System Security Symposium (NDSS'14), February 23-26, 2014, San Diago, CA.

1. uses LBR, as kBouncer; past ROP activity
2. emulates instructions; future ROP activity

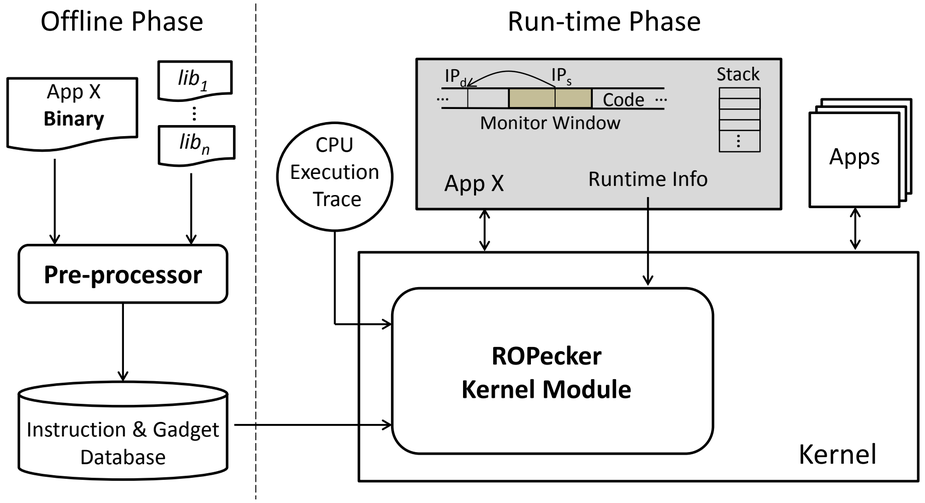


VirginiaTech
Invent the Future

ROPecker

Cheng, Y., Zhou, Z., Miao, Y., Ding, X. and DENG, H., Robert. ROPecker: A Generic and Practical Approach For Defending Against ROP Attack.(2014). In Proceedings of the 21th Annual Network and Distributed System Security Symposium (NDSS'14), February 23-26, 2014, San Diago, CA.

1. uses LBR, as kBouncer; past ROP activity
2. emulates instructions; future ROP activity
3. decreases availability of gadgets

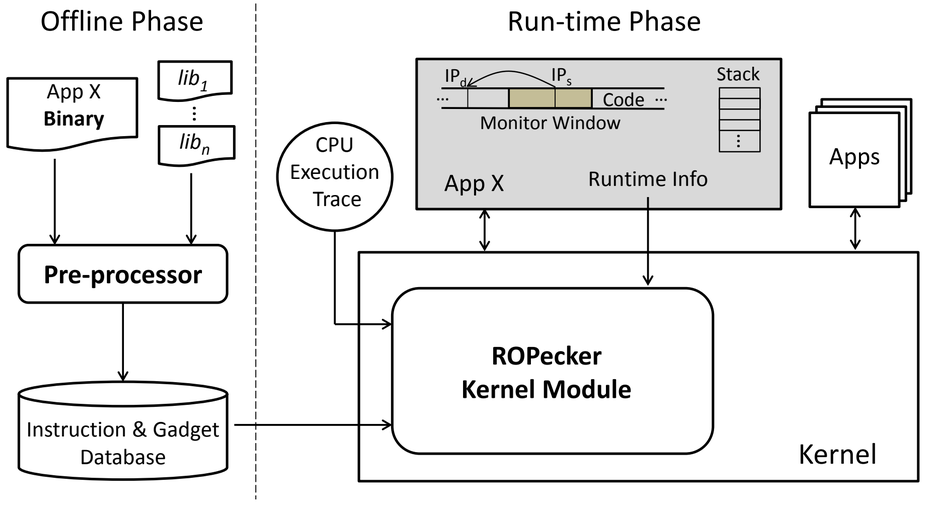


VirginiaTech
Invent the Future

ROPecker

Cheng, Y., Zhou, Z., Miao, Y., Ding, X. and DENG, H., Robert. ROPecker: A Generic and Practical Approach For Defending Against ROP Attack.(2014). In Proceedings of the 21th Annual Network and Distributed System Security Symposium (NDSS'14), February 23-26, 2014, San Diago, CA.

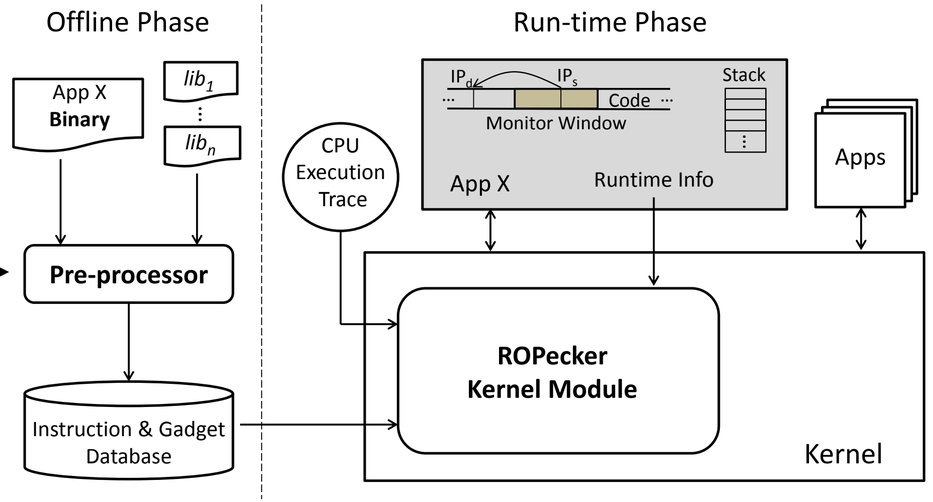
1. uses LBR, as kBouncer; past ROP activity
2. emulates instructions; future ROP activity
3. decreases availability of gadgets
4. identifies potential gadgets



VirginiaTech
Invent the Future

identify potential gadgets (six instructions) via disassembly:

1. application
2. shared library

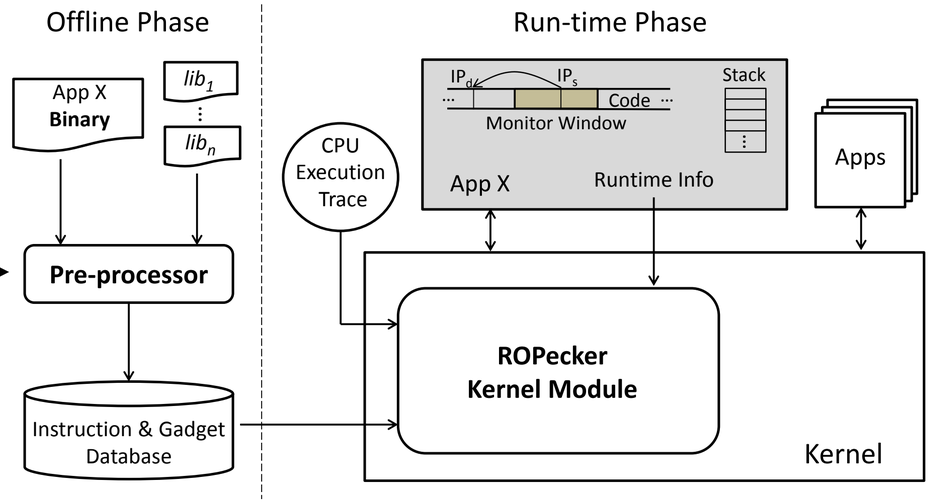


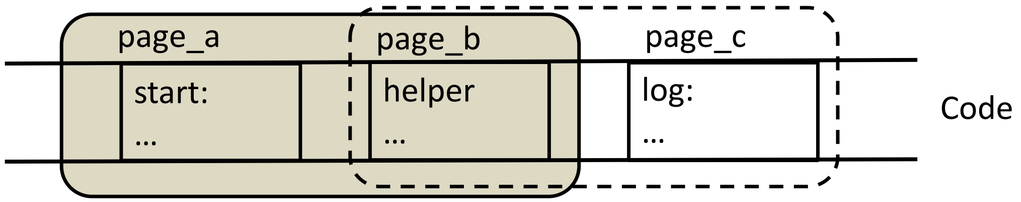
identify potential gadgets (six instructions) via disassembly:

1. application
2. shared library

create db

if addresses of gadget used, we assume ROP activity






```
1:  int helper(int cmd,char* in){
2:  log(cmd, in);
3:  switch(cmd){
4:      case CMD_START:
5:       start(inputs);
6:       break;
        . . .
    }
```

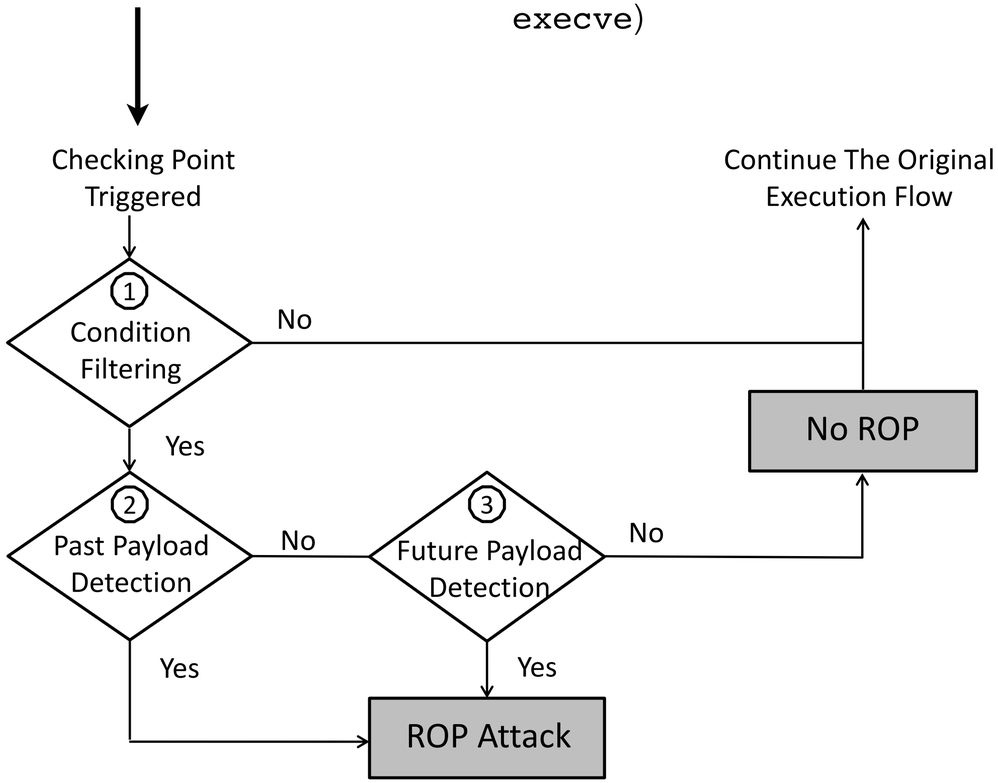
Old Sliding Window

New Sliding Window

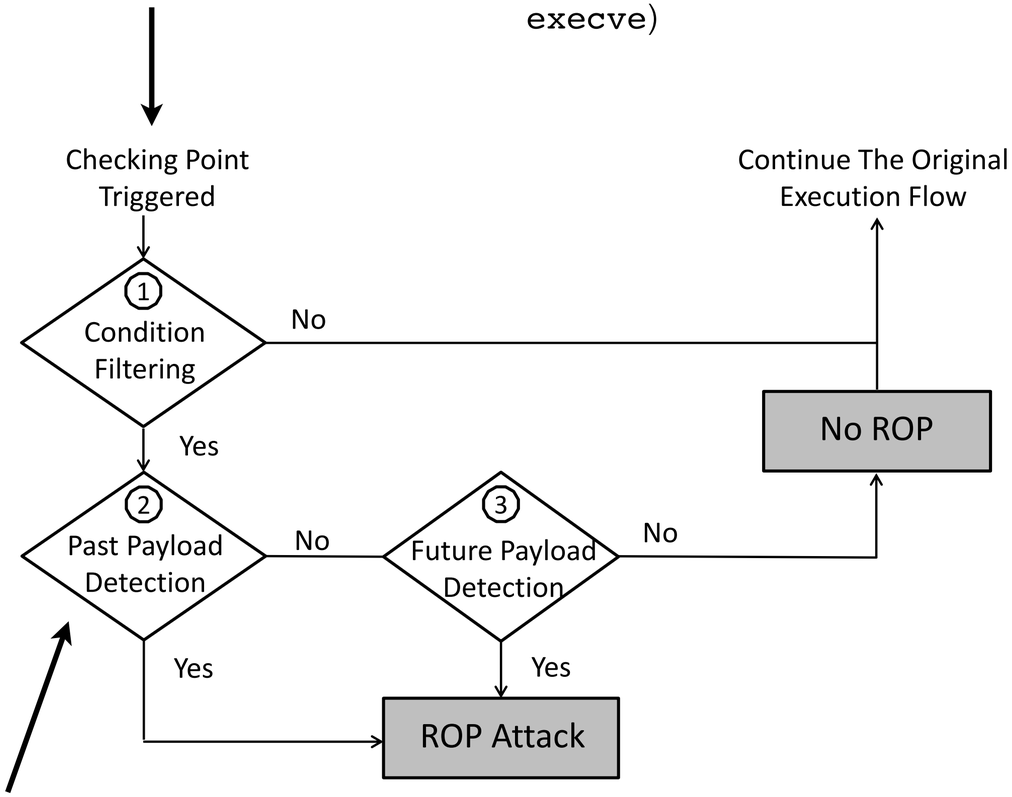
sliding window:

1. all pages set NX
2. for each call:
   a. enable X for next N bytes/pages
   b. disable X for previous N bytes/pages
3. 8 KB or 16 KB

1. page fault from execution in NX region
2. call to protect system call (`mprotect, mmap2, execve`)

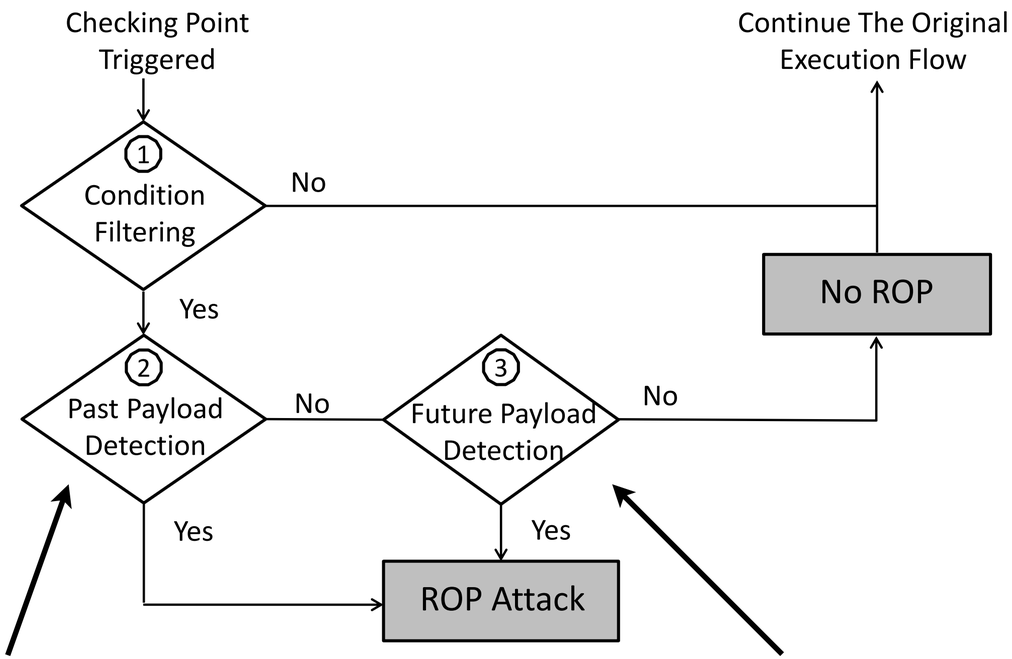


1. page fault from execution in NX region
2. call to protect system call (`mprotect, mmap2, execve`)



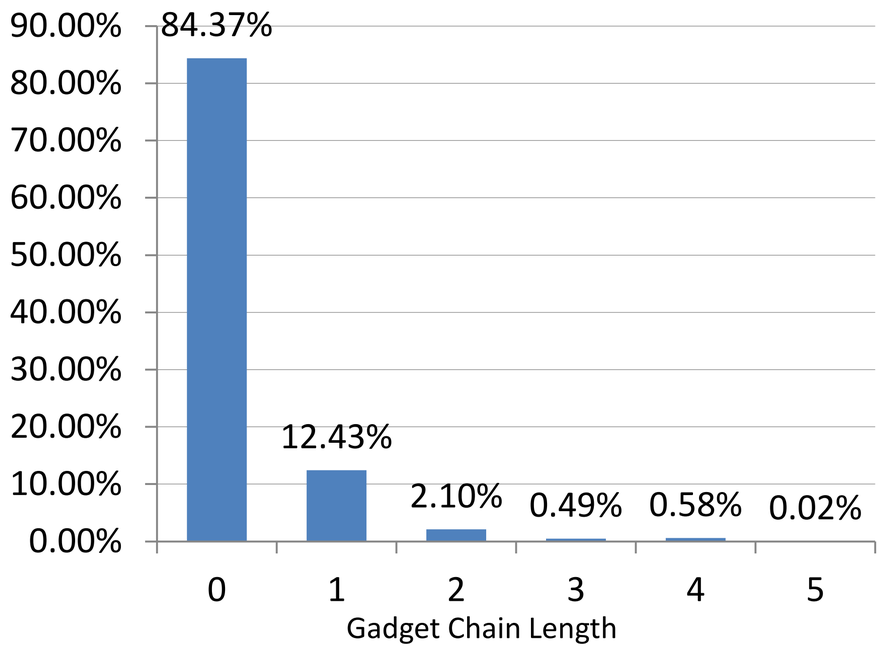
1. for indirect calls after context switch
2. count branches to potential gadgets

1. page fault from execution in NX region
2. call to protect system call (`mprotect, mmap2, execve`)



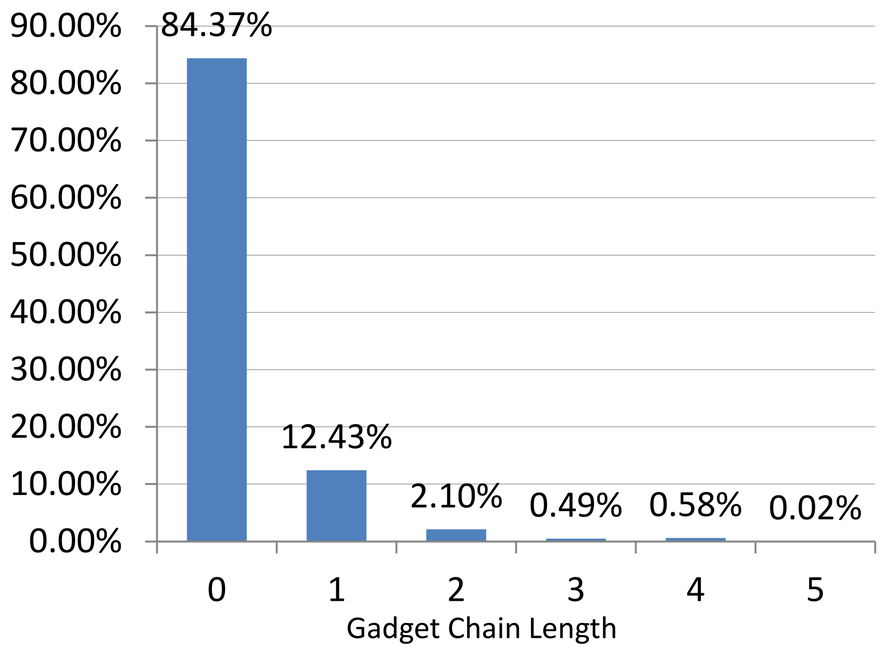
1. for indirect calls after context switch
2. count branches to potential gadgets

1. gadget chain too small?
2. emulate call stack, do we keep returning to suspected gadgets?



length of (potential) gadget chains in non-malicious code

17 popular Linux tools, 12 benchmark tools of SPEC INT2006, and 3 large binaries



if count from (2,3) > 11—16, have ROP activity

length of (potential) gadget chains in non-malicious code

17 popular Linux tools, 12 benchmark tools of SPEC INT2006, and 3 large binaries

ROPGuard

Fratrić, Ivan. "ROPGuard: Runtime prevention of return-oriented programming attacks." (2012).

runtime checks when critical functions are called:

1. ensure stack pointer stays within area
(prevent stack pivot attack)
2. address of function would be on the top of the stack if called using return otherwise not
3. instructions preceded by call; return address executable; call must be related to function
4. valid stack: return address just below frame pointer
5. stack pointer should always be resolvable from machine code
6. function should not be allowed to make code sections executable; don’t load dlls from some places (network)

Evaluating the Effectiveness of Current Anti-ROP Defenses

Schuster, F., Tendyck, T., Pewny, J., Maaß, A., Steegmanns, M., Contag, M. and Holz, T., 2014, September. Evaluating the effectiveness of current anti-ROP defenses. In International Workshop on Recent Advances in Intrusion Detection (pp. 88-108). Springer, Cham.

Evaluating the Effectiveness of Current Anti-ROP Defenses

Schuster, F., Tendyck, T., Pewny, J., Maaß, A., Steegmanns, M., Contag, M. and Holz, T., 2014, September. Evaluating the effectiveness of current anti-ROP defenses. In International Workshop on Recent Advances in Intrusion Detection (pp. 88-108). Springer, Cham.

kBouncer assumptions:

Evaluating the Effectiveness of Current Anti-ROP Defenses

Schuster, F., Tendyck, T., Pewny, J., Maaß, A., Steegmanns, M., Contag, M. and Holz, T., 2014, September. Evaluating the effectiveness of current anti-ROP defenses. In International Workshop on Recent Advances in Intrusion Detection (pp. 88-108). Springer, Cham.

kBouncer assumptions:

1. long gadgets unlikely

Evaluating the Effectiveness of Current Anti-ROP Defenses

Schuster, F., Tendyck, T., Pewny, J., Maaß, A., Steegmanns, M., Contag, M. and Holz, T., 2014, September. Evaluating the effectiveness of current anti-ROP defenses. In International Workshop on Recent Advances in Intrusion Detection (pp. 88-108). Springer, Cham.

kBouncer assumptions:

1. long gadgets unlikely
2. can't string together non-malicious indirects that exhaust LBR

strategy: last 14 indirect branches should be benign

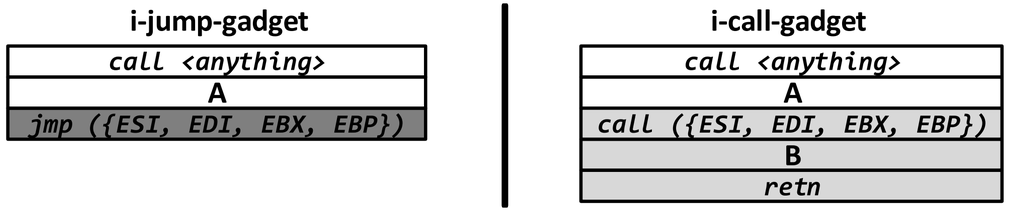
1. identify: LBR-flushing function
    a. legitimate function
    b. many legitimate indirects (call-preceded)

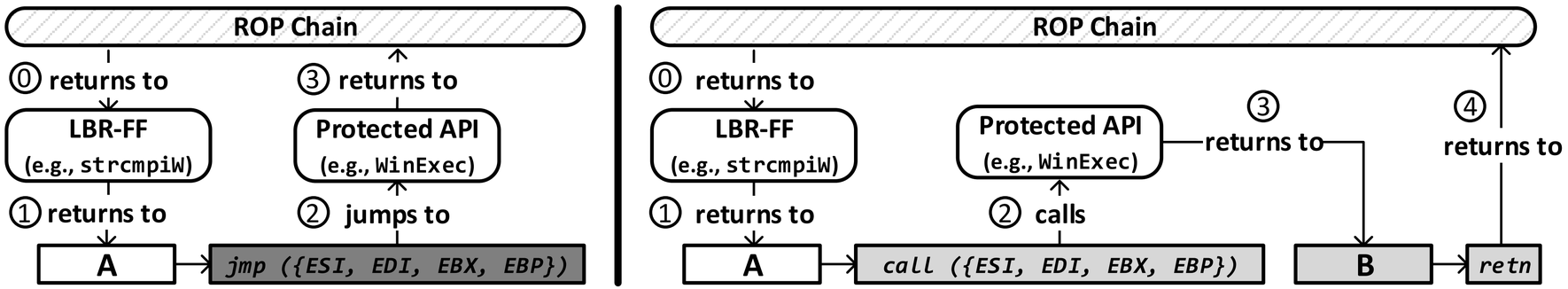
strategy: last 14 indirect branches should be benign

1. identify: LBR-flushing function
   a. legitimate function
   b. many legitimate indirects (call-preceded)

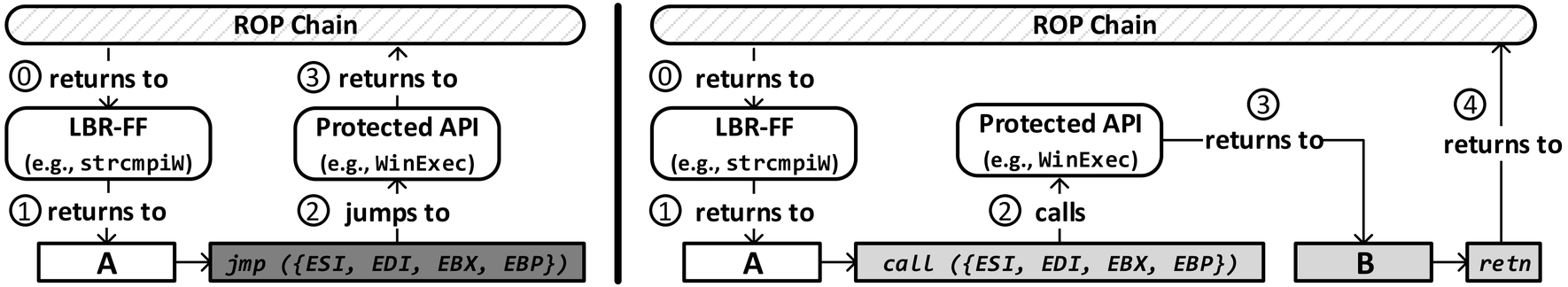
strategy: last 14 indirect branches should be benign

2. identify: call-preceded gadget that doesn't affect ROP state

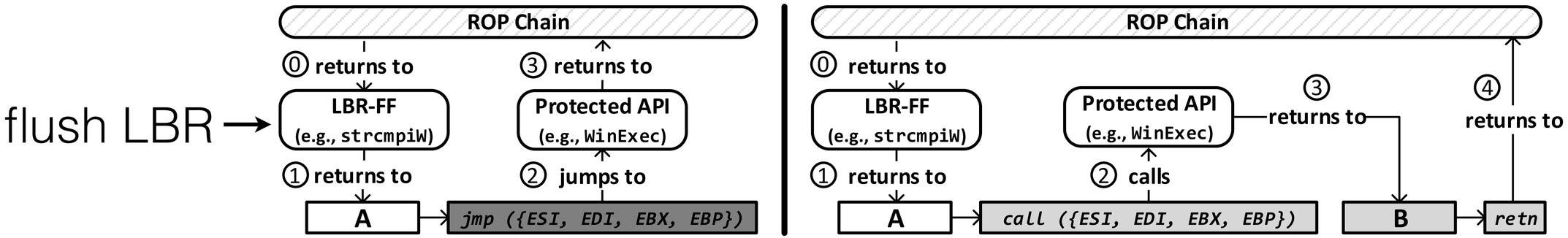




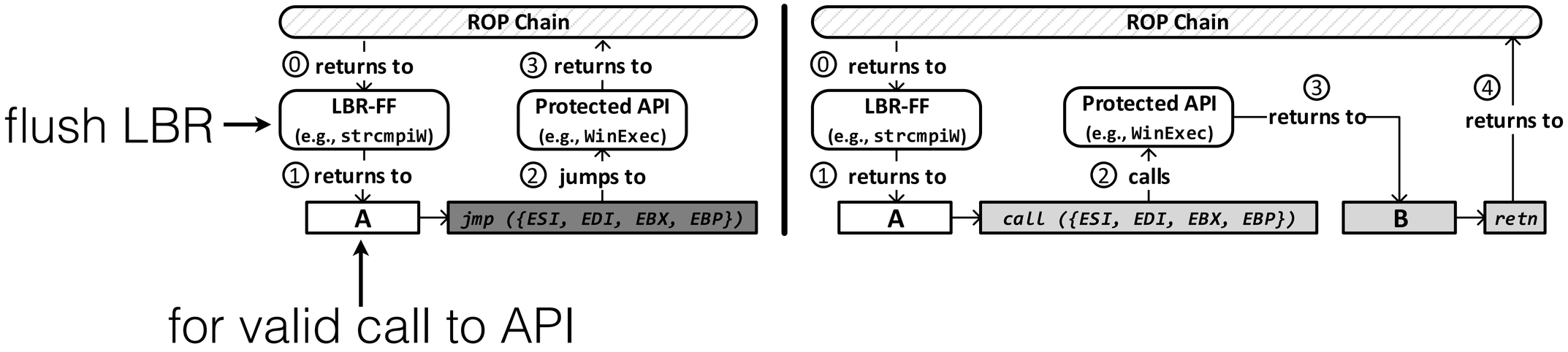
setup vars/args



setup vars/args



setup vars/args



for valid call to API

ROPecker vulnerabilities:

1. long gadgets

ROPecker vulnerabilities:

1. long gadgets
2. gadget that contains direct branch

ROPecker vulnerabilities:

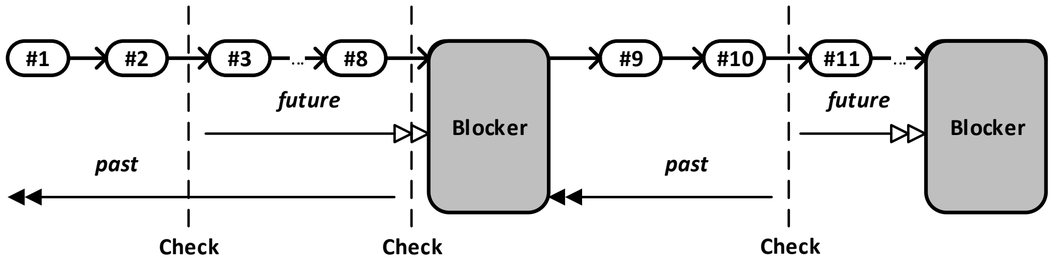
1. long gadgets
2. gadget that contains direct branch to another gadget

ROPecker vulnerabilities:

1. long gadgets
2. gadget that contains direct branch to another gadget
3. gadgets within windows

ROPecker vulnerabilities:

1. long gadgets
2. gadget that contains direct branch to another gadget
3. gadgets within windows



blocker: end of detection

1. smallest window size: one page
2. direct jumps to gadgets
3. long gadgets