ropbot: Reimaging Code Reuse Attack Synthesis

Kyle Zeng[1], Moritz Schloegel[2], Christopher Salls[3],
Adam Doupé[1], Ruoyu Wang[1], Yan Shoshitaishvili[1], Tiffany Bao[1]

[1]Arizona State University
[2]CISPA Helmholtz Center for Information Security
[3]UC Santa Barbara

Automated Code Reuse Payload Generation

Code reuse attacks (e.g., ROP) are well-known

Automated Code Reuse Payload Generation

Code reuse attacks (e.g., ROP) are well-known

Expensive in human effort when it comes to the real-world

Automated Code Reuse Payload Generation

Code reuse attacks (e.g., ROP) are well-known

Expensive in human effort when it comes to the real-world

Multiple attempts to automate it since 2011

Existing Works

Existing Works

	Year	Algorithm	Arch
exrop	**2020**	GAT	x64
SGC	**2021**	BGAT	x64/x86
RiscyROP	**2022**	BGAT	AArch64/RISC-V
crackers	**2025**	BGAT	x64
arcanist	**2025**	BGAT	x64/x86/ARM

(B)GAT: (bounded) generate-and-test

Bounded Generate-and-Test

- Enumerate all gadget combinations (with a bound N)
  - Check whether the gadget list works

Bounded Generate-and-Test

- Enumerate all gadget combinations (with a bound N)
  - Check whether the gadget list works
- Dilemma of choosing N

Bounded Generate-and-Test

- Enumerate all gadget combinations (with a bound N)
  - Check whether the gadget list works
- Dilemma of choosing N
- Heuristics to list possible combinations first
- Time complexity: O(n^N)

Why BGAT then?

- Valid ROP chain: chainable gadgets + correct effect

Why BGAT then?

- Valid ROP chain: chainable gadgets + correct effect

G1: pop rax; ret

G2: pop rbx; ret

Why BGAT then?

- Valid ROP chain: chainable gadgets + correct effect

G1: pop rax; ret

G2: pop rbx; ret

G1: pop rax; jmp rcx

G2: pop rbx; jmp rdx

Why BGAT then?

- Valid ROP chain: chainable gadgets + correct effect

G1: pop rax; ret

G2: pop rbx; ret

G1: pop rax; jmp rcx

G2: pop rbx; jmp rdx

- No way to guarantee chainability between gadgets

What if:

We have a special way of grouping gadgets to guarantee chainability?

Gadget Finding Algorithm

- Gadget: a unit that maintains control flow

Gadget Finding Algorithm

- Gadget: a unit that maintains control flow
- Look for code sequence that can *potentially* move a symbolic value into PC in a fully symbolic state (i.e. symbolic registers, stack, memory)

Gadget Finding Algorithm

- Gadget: a unit that maintains control flow
- Look for code sequence that can *potentially* move a symbolic value into PC in a fully symbolic state (i.e. symbolic registers, stack, memory)
- ret ⇔ pop PC
- jmp rax ⇔ mov PC, rax

Gadget Finding Algorithm – cont.

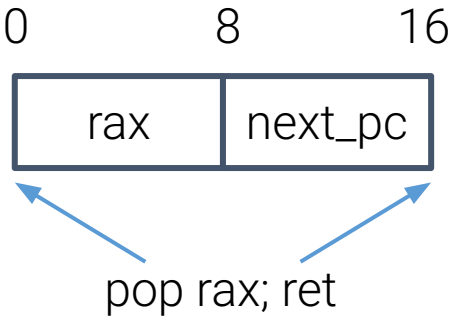
- Advantages
  a. Architecture agnostic
  b. Accounts for conditional branches
  c. More gadgets (gadgets can span multiple basic blocks)
  d. Flexible, can find new gadgets
- new gadget type: gadgets ending with “ldr pc” on ARM

Self-Contained Property

- Has positive SP change (before and after the gadget)
- Take next PC from within the SP change
- No conditional branches

Self-Contained Property

- Has positive SP change (before and after the gadget)
- Take next PC from within the SP change
- No conditional branches



ROPBlock

- A gadget or a sequence of gadgets that is self-contained
- By definition, self-contained gadgets are ROPBlocks (e.g. pop rax; ret)

ROPBlock

- A gadget or a sequence of gadgets that is self-contained
- By definition, self-contained gadgets are ROPBlocks (e.g. pop rax; ret)
- ROPBlock + ROPBlock = ROPBlock (guaranteed to be chainable)

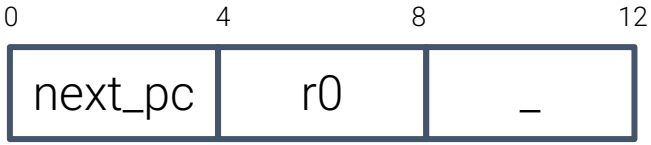
ROPBlock Chaining

ldr r0, [sp, #4]; ldr lr, [sp], #4; add sp, #8; bx lr

pop.w {r1, lr}; bx lr

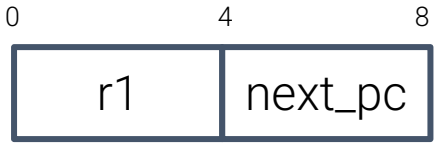
ROPBlock Chaining

ldr r0, [sp, #4]; ldr lr, [sp], #4; add sp, #8; bx lr



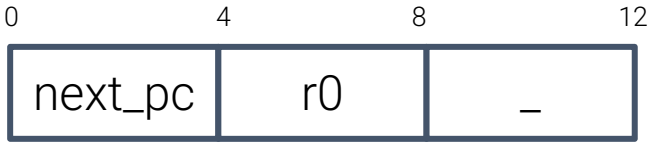
ldr r0, [sp, #4]; ldr lr, [sp], #4; add sp, #8; bx lr
{gadget1}

pop.w {r1, lr}; bx lr

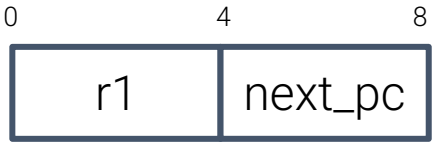


pop.w {r1, lr}; bx lr
{gadget2}

ROPBlock Chaining

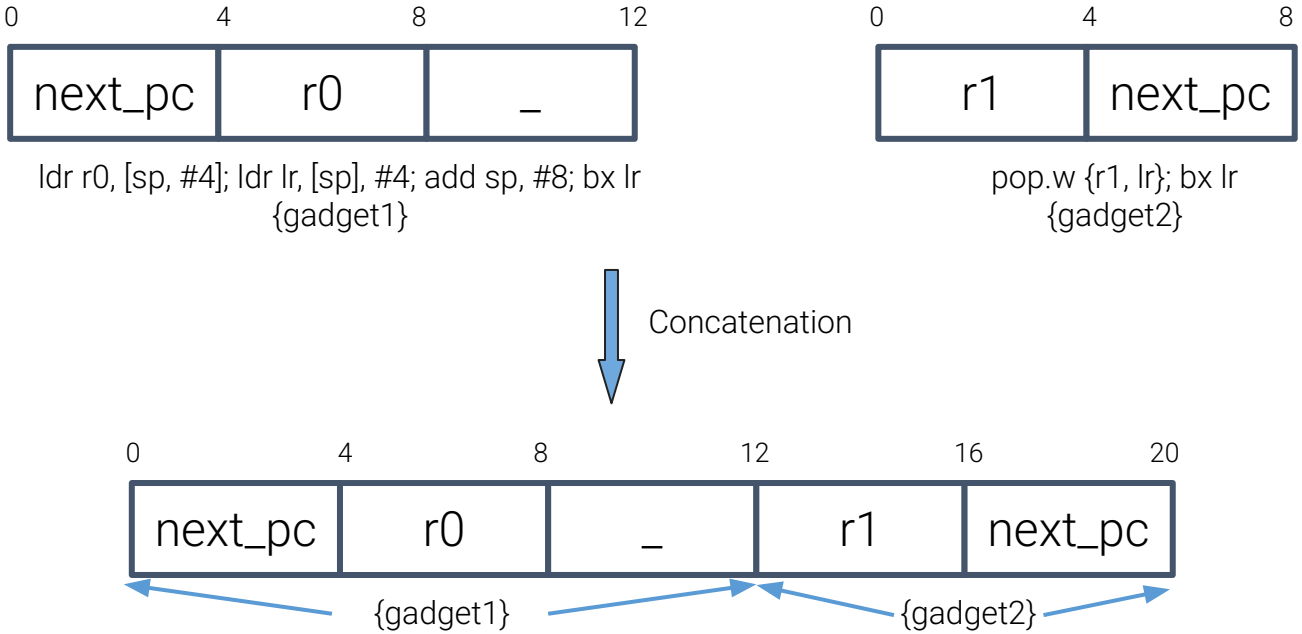


ldr r0, [sp, #4]; ldr lr, [sp], #4; add sp, #8; bx lr
{gadget1}

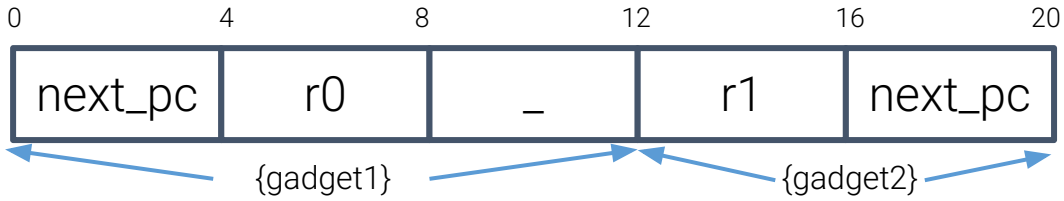


pop.w {r1, lr}; bx lr
{gadget2}

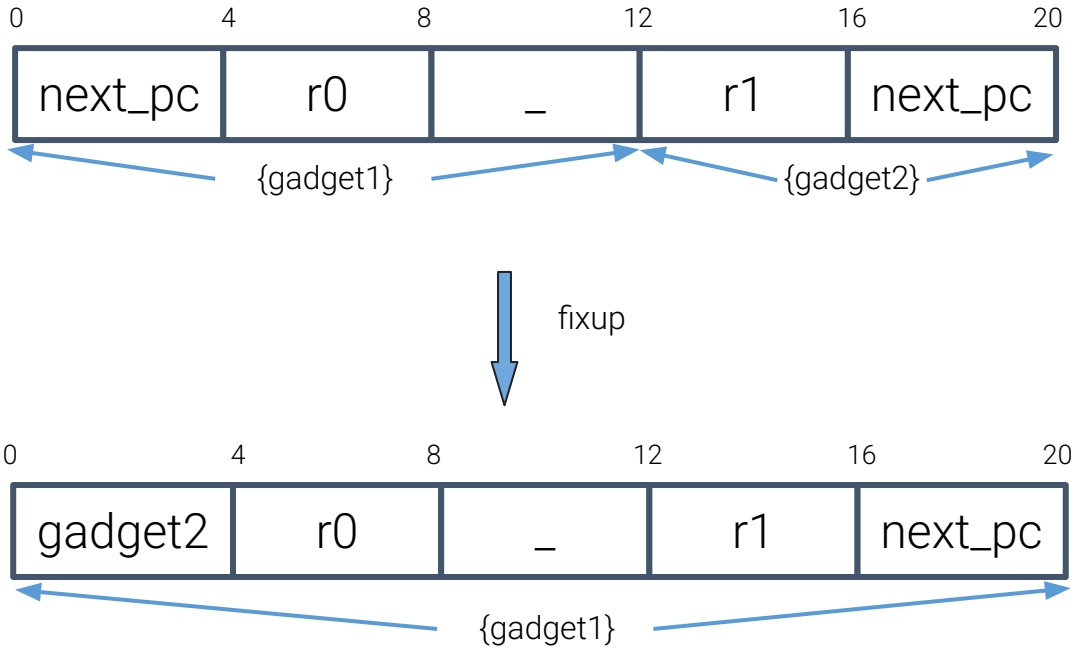
ROPBlock Chaining



ROPBlock Chaining



ROPBlock Chaining

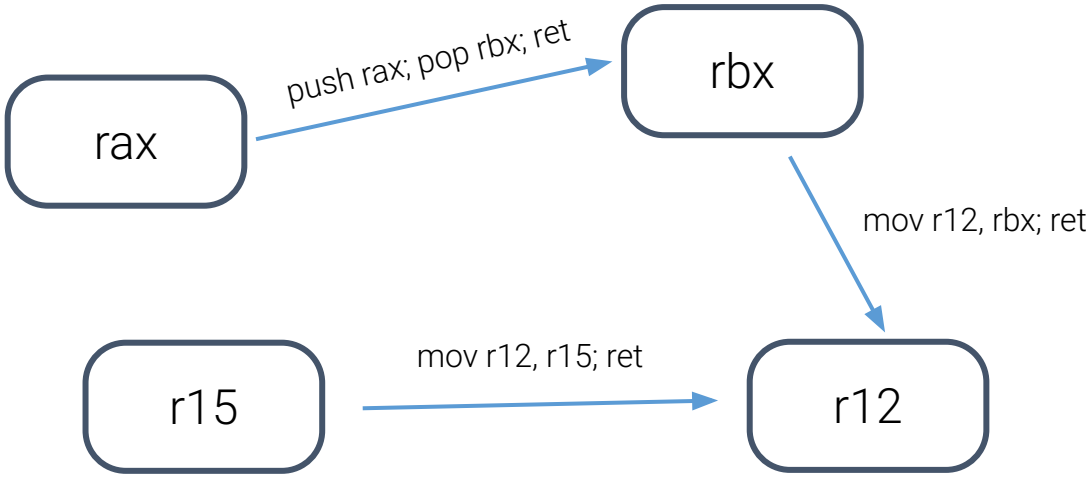


Normalization

- Goal: turn non self-contained gadgets into ROPBlock
  a. Set branch guard for gadgets with conditional branches
  b. Do memory write for `call [mem]` gadgets
  c. Set registers for `jmp/call <reg>` gadgets

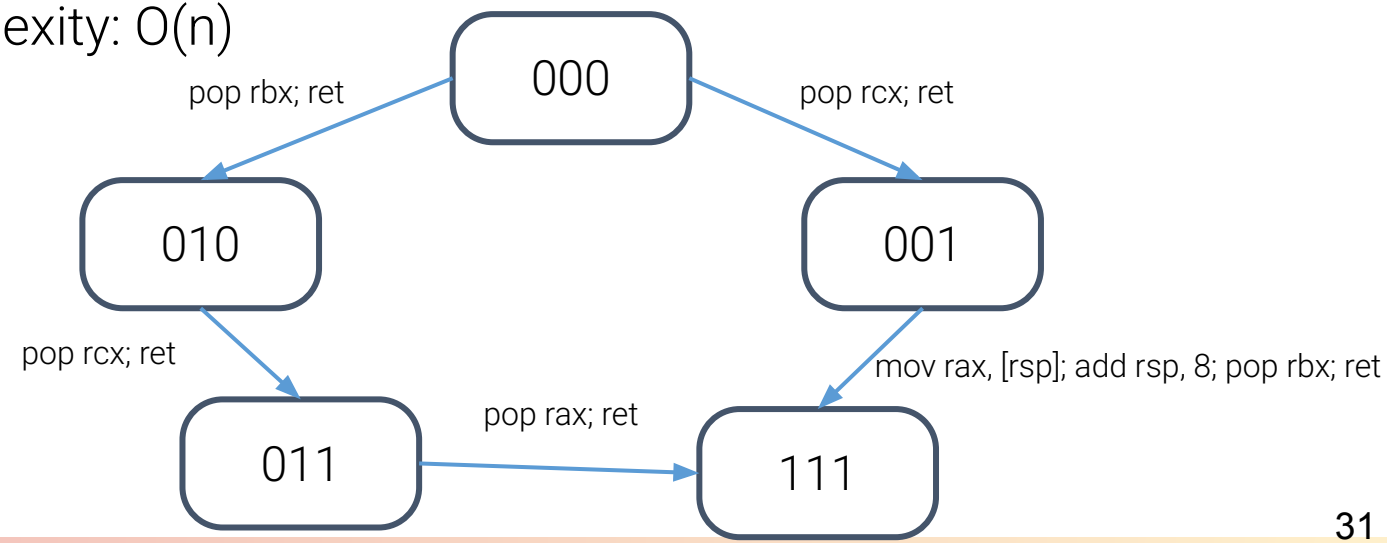
Register Moving Graph

- Capture the register moving relationship.
- New reg setting capability if 1) there is path and 2) we can set the source



Graph Search

- Given a request {rax=..., rbx=..., rcx=...}, goal: a path from 000 to 111
- 001 means, can't control {rax, rbx} and can control {rcx}
- Time Complexity: O(n)



Security-Sensitive Dataset (10 binaries*)

Func call: 0xfacefeed(0xdeadbeef, 0x40, 0x7b)

	ropbot	exrop	SGC	Crackers	Arcanist
Success	10	6	1	5	7

* tftpd, jsc, openssl, openssh, dnsmasq, apache2, nginx, linux kernel, chromium, firefox

execve(/bin/sh, 0, 0) (272 binaries)

	Success	False Positive	Time
ropbot	**244**	**0%**	**3.2h**
SGC	27	1.6%	123.h
exrop	90	23.1%	20.9h
crackers	0	N/A	55.2h

* Arcanist is left out from this evaluation task because it does not support memory write, which is needed for this task

Full Chain (37 binaries)

full chain: dup+dup+execve

	Success	False Positive	Time
ropbot	**37**	**0%**	**1.0h**
exrop	21	4.5%	2.5h

ropbot's Cross Arch Eval

Func call: 0xfacefeed(0xdeadbeef, 0x40, 0x7b)

	Success	Total	Rate	FP	Time
x64	635	1022	62.1%	0%	5.3h
mips	285	1000	28.5%	0%	10.1h
aarch64	58	172	33.7%	0%	3.9h
arm	660	1000	66.0%	0%	4.1h
risc-v	358	1000	35.8%	0%	24.1h

Industry Adoption

- Google uses ropbot as part of its kernelXDK framework

- Required for all submissions to KernelCTF VRP

Conclusion

- Propose the ROPBlock abstraction that guarantees chainability
- Devise new algorithm that reduces time complexity from O(n^N) to O(n)
- Outperform all state-of-the-art works in all benchmarks
- ropbot is adopted by the industry

ropbot: Reimaging Code Reuse Attack Synthesis

Thank you!

Q & A

https://github.com/sefcom/ropbot


Kyle Zeng


zengyhkyle@asu.edu


@ky1ebot

@Kyle-Kyle