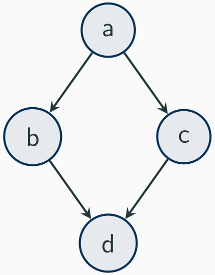
Towards Automating Code-Reuse Attacks Using Synthesized Gadget Chains

Moritz Schloegel, Tim Blazytko, Julius Basler, Fabian Hemmer, and Thorsten Holz

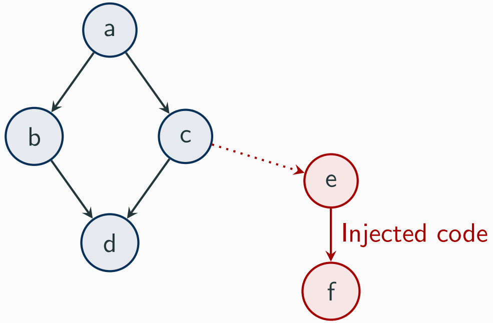
Ruhr-Universität Bochum

Why should you even care?

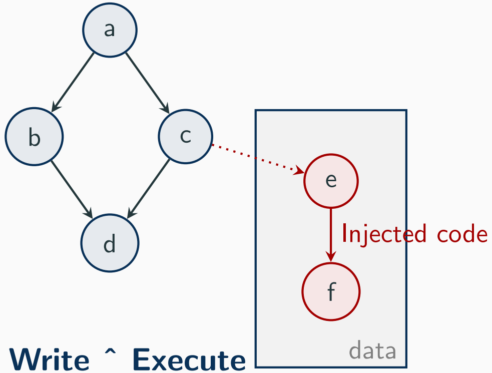
- Stitching gadgets manually is annoying
- Tools usually fail when you need them the most



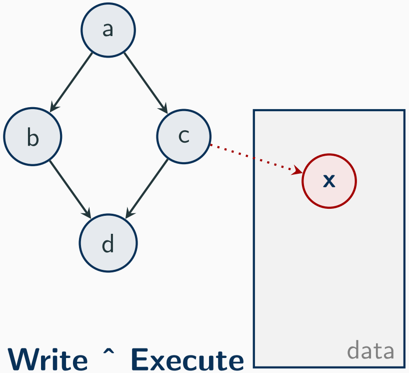
Control Flow Graph (CFG)



Code-Injection Attacks

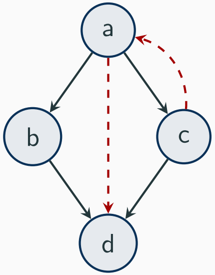


Write ^ Execute



Write ˆ Execute

$$\begin{tikzcd}
 & a \arrow[dl] \arrow[dr] & \\
b \arrow[dr] & & c \arrow[dl] \\
 & d &
\end{tikzcd}$$



Code-Reuse Attacks

Gadgets

```
n+nl0xc0deba5e:
    n+nfxor n+nbebxp, n+nbebx
    n+nfmov n+nbebpp, n+nbesp
    n+nfpop n+nbebp
    n+nfret

n+nl0xdeadbeef:
    n+nfmov n+nbecxp, l+m+mh0xFFFFFFFF
    n+nfinc n+nbecx
    n+nfcall n+nbedx

n+nl0xcafe:
    n+nfpop n+nbebx
    n+nfpop n+nbecx
    n+nfjmp n+nbecx
```

```
n+nl0xc0deba5e:
    n+nfxor n+nbebxp, n+nbebx
    n+nfmov n+nbebpp, n+nbesp
    n+nfpop n+nbebp
    n+nfret

n+nl0xdeadbeef:
    n+nfmov n+nbecxp, l+m+mh0xFFFFFFFF
    n+nfinc n+nbecx
    n+nfcall n+nbedx

n+nl0xcafe:
    n+nfpop n+nbebx
    n+nfpop n+nbecx
    n+nfjmp n+nbecx
```

Gadgets

- Typically a few instructions

```
n+nl0xc0deba5e:
    n+nfxor n+nbebxp, n+nbebx
    n+nfmov n+nbebpp, n+nbesp
    n+nfpop n+nbebp
    n+nfret

n+nl0xdeadbeef:
    n+nfmov n+nbecxp, l+m+mh0xFFFFFFFF
    n+nfinc n+nbecx
    n+nfcall n+nbedx

n+nl0xcafe:
    n+nfpop n+nbebx
    n+nfpop n+nbecx
    n+nfjmp n+nbecx
```

Gadgets

- Typically a few instructions
- Followed by an *indirect control flow transfer*

```
n+nl0xc0deba5e:
    n+nfxor n+nbebxp, n+nbebx
    n+nfmov n+nbebpp, n+nbesp
    n+nfpop n+nbebp
    n+nfret

n+nl0xdeadbeef:
    n+nfmov n+nbecxp, l+m+mh0xFFFFFFFF
    n+nfinc n+nbecx
    n+nfcall n+nbedx

n+nl0xcafe:
    n+nfpop n+nbebx
    n+nfpop n+nbecx
    n+nfjmp n+nbecx
```

Gadgets

- Typically a few instructions
- Followed by an *indirect control flow transfer*

Many types

- Return-oriented programming
- Jump-oriented programming
- Call-oriented programming
- . . .

Problem: (Too) many potential gadgets and chains

Solution: Automation $\Rightarrow$ build tools!

Current tools are great..
.. but no panacea

	`P-SHAPE`	`angrop`	`ROPium`	`ROPgadget`	`Ropper`
supports chains without `ret`	✗	✗	✓	✓	✓
no hardcoded chaining rules	✓	✓	✓	✗	✗
no classification needed	✗	✗	✗	✗	✗
supports arbitrary postconditions	✗	✗	✗	✗	✗

Our approach: `SGC`

before exploitation

preconditions

rax = 0x1337
rbx = 0x12
...

before exploitation

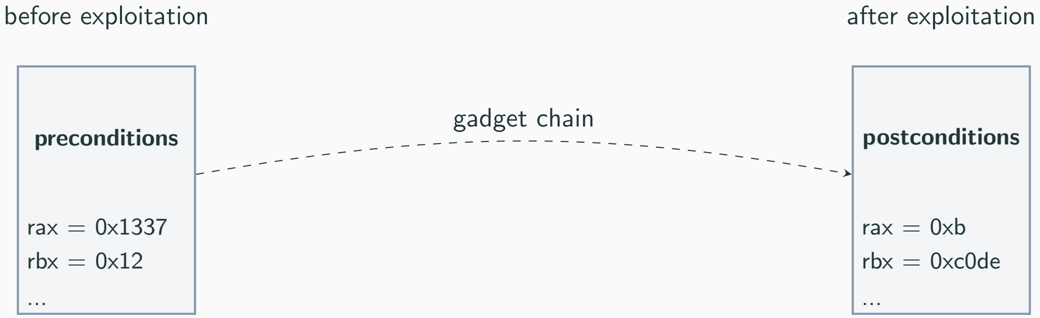
preconditions

rax = 0x1337
rbx = 0x12
...

after exploitation

postconditions

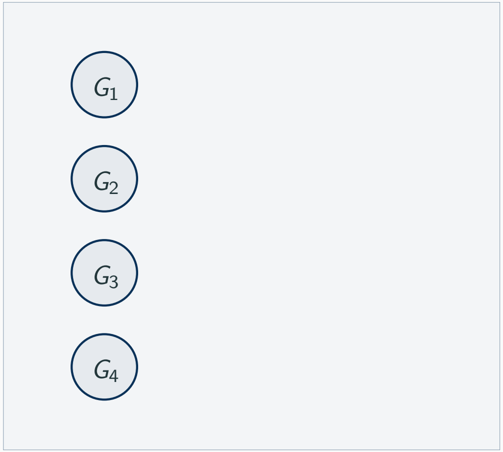
rax = 0xb
rbx = 0xc0de
...


before exploitation
after exploitation
preconditions
rax = 0x1337
rbx = 0x12
...
gadget chain
postconditions
rax = 0xb
rbx = 0xc0de
...

before exploitation

preconditions

rax = 0x1337
rbx = 0x12
...



after exploitation

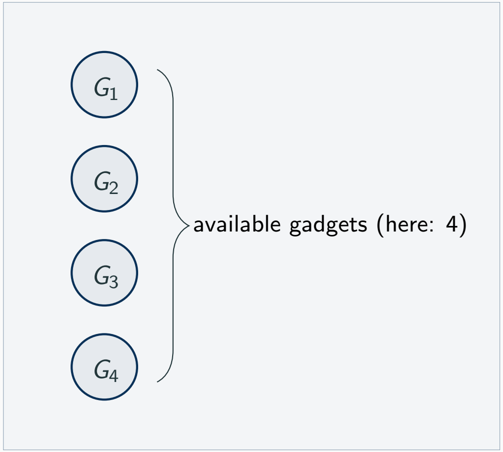
postconditions

rax = 0xb
rbx = 0xc0de
...

before exploitation

preconditions

rax = 0x1337
rbx = 0x12
...



after exploitation

postconditions

rax = 0xb
rbx = 0xc0de
...

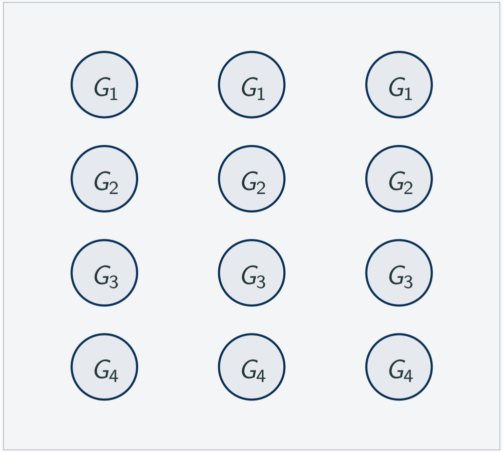
before exploitation

preconditions

rax $=$ 0x1337

rbx $=$ 0x12

...



after exploitation

postconditions

rax $=$ 0xb

rbx $=$ 0xc0de

...

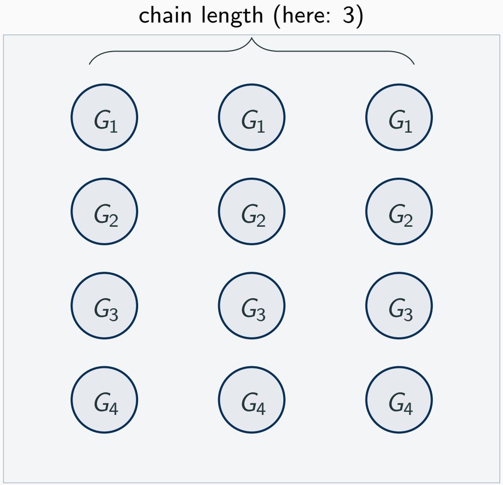
before exploitation

preconditions

rax = 0x1337

rbx = 0x12

...



after exploitation

postconditions

rax = 0xb

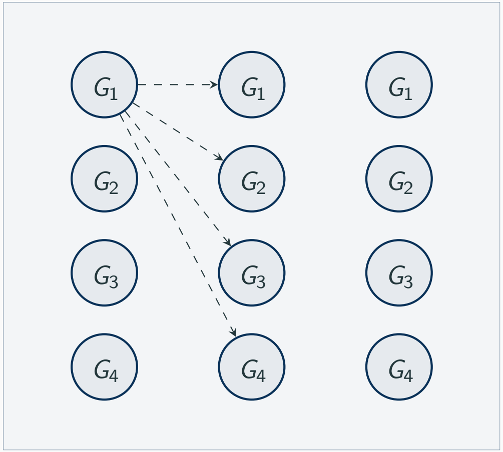
rbx = 0xc0de

...

before exploitation

preconditions

rax = 0x1337
rbx = 0x12
...



after exploitation

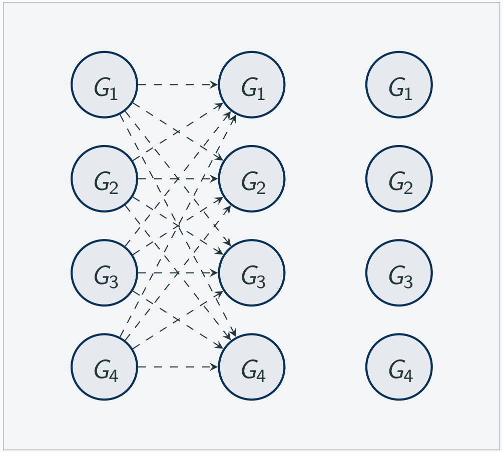
postconditions

rax = 0xb
rbx = 0xc0de
...

before exploitation

preconditions

rax = 0x1337
rbx = 0x12
...



after exploitation

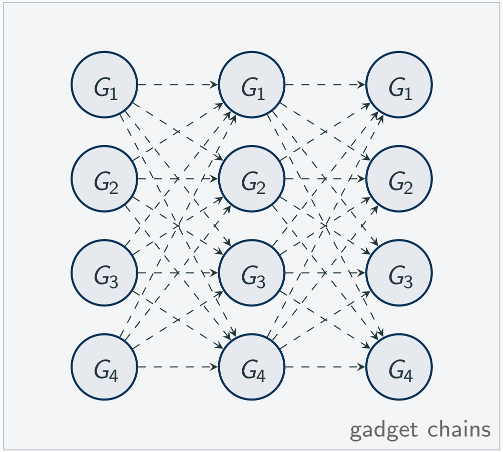
postconditions

rax = 0xb
rbx = 0xc0de
...

before exploitation

preconditions

rax = 0x1337
rbx = 0x12
...



after exploitation

postconditions

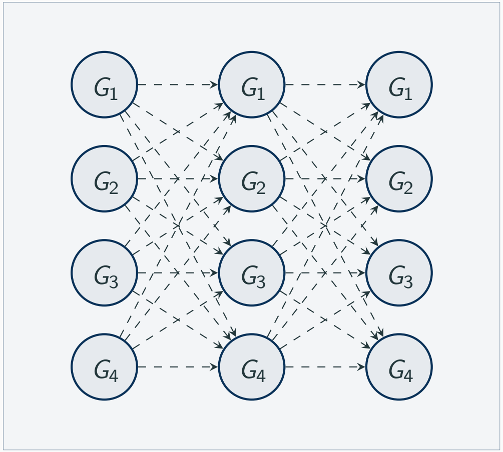
rax = 0xb
rbx = 0xc0de
...

before exploitation

preconditions

rax $=$ 0x1337
rbx $=$ 0x12
…

$+$



$=$

after exploitation

postconditions

rax $=$ 0xb
rbx $=$ 0xc0de
…

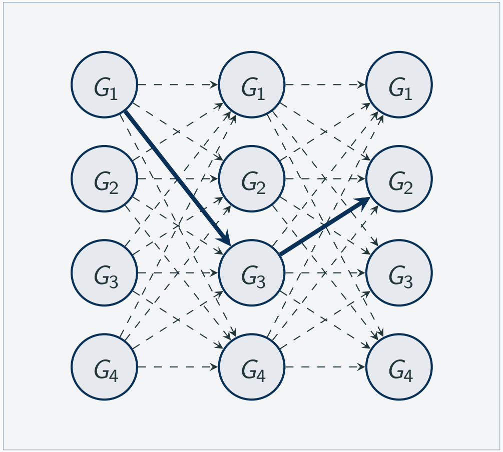
Goal: Find chain

before exploitation

preconditions

rax $=$ 0x1337
rbx $=$ 0x12
…

$+$



$=$

after exploitation

postconditions

rax $=$ 0xb
rbx $=$ 0xc0de
…

Goal: Find chain, e. g., $G_1 \rightarrow G_3 \rightarrow G_2$

How?

$\Rightarrow$ **SMT solver!**

Formula

$$preconditions \wedge gadget\_chain \wedge postconditions$$

Encoding of gadgets and chains

$\Rightarrow$ details in the paper

What do we get?

SAT ✓ | UNSAT ✗ | Timeout ⌛

SAT ✓ | UNSAT ✗ | Timeout ⌛

$\Rightarrow$ chain found!

SAT ✓ | UNSAT ✗ | Timeout ⌛

$\Rightarrow$ no chain can exist!

SAT ✓ | UNSAT ✗ | Timeout ⌛

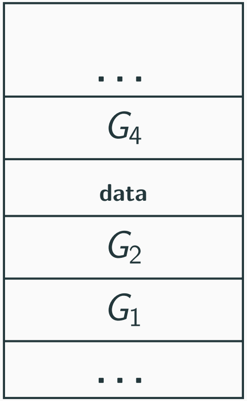
$\Rightarrow$ retry and sample subset?

Results

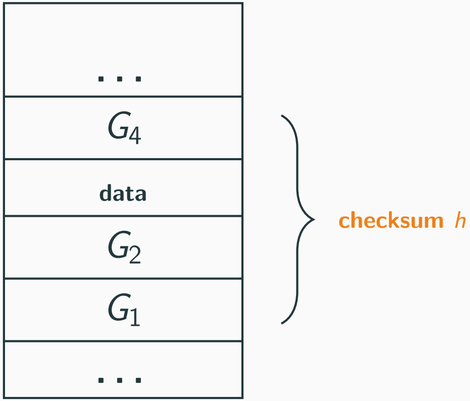
Comparison to other tools

		SGC	P-SHAPE	angrop	ROPium	ROPgadget	Ropper
mprotect	chromium	✓	✗	✗	✓	-	✗
	apache2	✓	(✓)	✓	✓	-	(✓)
	nginx	✓	(✓)	✓	✓	-	✗
	OpenSSL	✓	(✓)	✗	✗	-	✗
	libc	✓	(✓)	✓	✓	-	✓
mmap	chromium	✓[1]	✗	✗	✓	-	-
	apache2	✓	✗	✗	✓	-	-
	nginx	✓	(✓)	✗	✗	-	-
	OpenSSL	✗[2]	✗	✗	✗	-	-
	libc	✓	(✓)	✗	✓	-	-
execve	chromium	✓	-	✗	✓	✓	✗
	apache2	✓	-	(✓)	✓	✗	(✓)
	nginx	✓	-	(✓)	✓	✗	✗
	OpenSSL	✓	-	✗	✗	✗	✗
	libc	✓	-	✓	✓	✓	✓

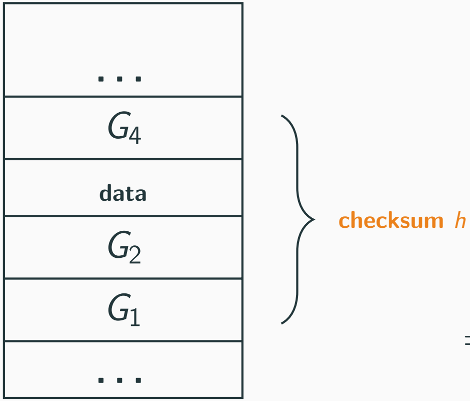
Target-specific constraints



Stack



Stack



$$\Rightarrow h(stack) == 0x1337$$

Stack

Takeaways

- Finding gadget chains is tedious

Takeaways

- Finding gadget chains is tedious
- But: SMT solver excels at doing so

Takeaways

- Finding gadget chains is tedious
- But: SMT solver excels at doing so
- Even for complex constraints

Takeaways

- Finding gadget chains is tedious
- But: SMT solver excels at doing so
- Even for complex constraints

Thank you!

Want to know more? Contact

Moritz Schloegel

@m_u00d8

Tim Blazytko

@mr_phrazer

Code: `https://github.com/RUB-SysSec/gadget_synthesis`