University of Chinese Academy of Sciences 中国科学院大学

中国科学院信息工程研究所 INSTITUTE OF INFORMATION ENGINEERING,CAS

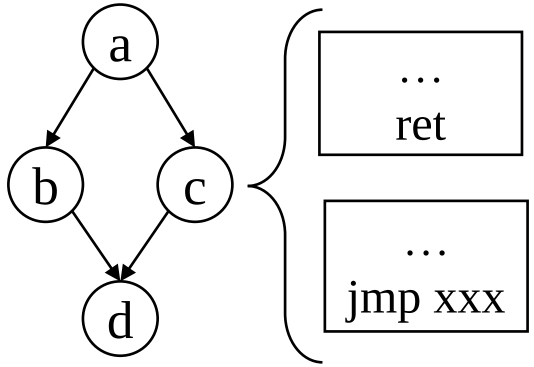
University of Colorado Boulder

Northwestern University

TGRop: Top Gun of Return-Oriented Programming Automation

Nanyu Zhong[1,2,3,4], Yueqi Chen[5], Yanyan Zou[1,2,3,4(✉)], Xinyu Xing[6], Jinwei Dong[1,2,3,4], Bingcheng Xian[1,2,3,4], Jiaxu Zhao[1,2,3,4], Menghao Li[1,2,3,4], Binghong Liu[1,2,3,4], and Wei Huo[1,2,3,4(✉)]

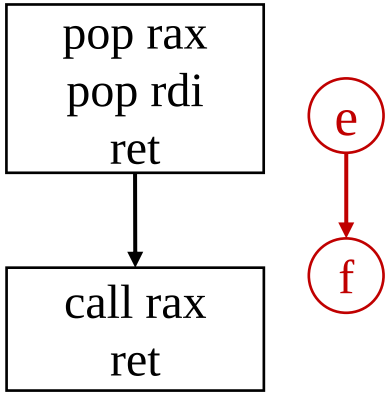
Return-Oriented Programming (ROP) —— What? When? How?



Control Flow Graph (CFG)

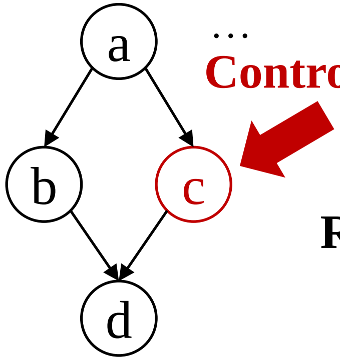
Gadgets

- From programs, libc,…
- A few instructions
- End with ret/jmp/call/…

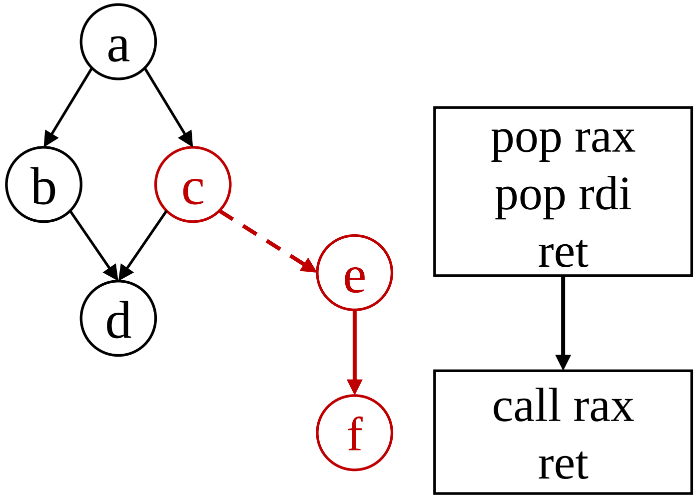


Chain Gadgets to call any function!

Buffer Overflow
Heap Overflow
Use After Free
…
Control Flow Hijack



ROP Time!



Magic!

Exploitation Base on ROP

An Interesting Metaphors Of ROP

Gadgets Classification

Some Gadget Chains

One ROP To Call Function

One LEGO Figures Match ROP Tasks: Memory-write, Memory-read or Call any Function!

The Past and Present of Return-Oriented Programming Automation

CCS 2007
Hovav Shacham's First Time Proposal of **ROP Attacks**!

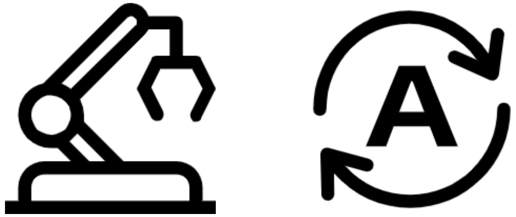
USENIX 2011
Edward J. Schwartz's Initial Proposal of **Automated ROP Exploitation!**

Open-Source Tool：
AngRop、Exrop、Ropium

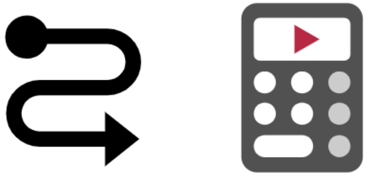
CCS 20，Esorics 21：
Limbo、SGC

?

Manual ROP

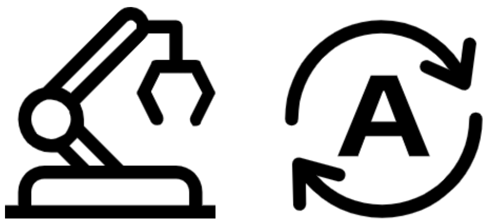


Heuristic-Based Automation



Exploratory-Based Automation

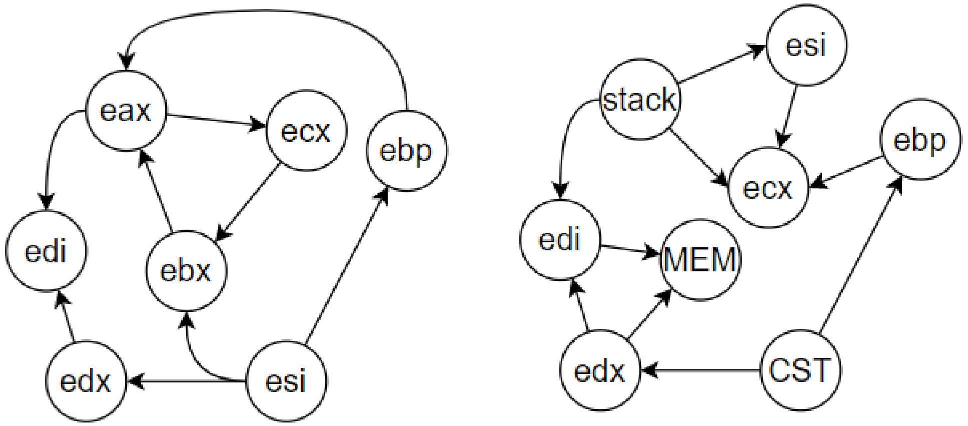
The Past and Present of Return-Oriented Programming Automation



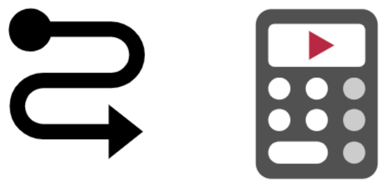
Heuristic-Based Automation

Gadgets Classification Based On semantic

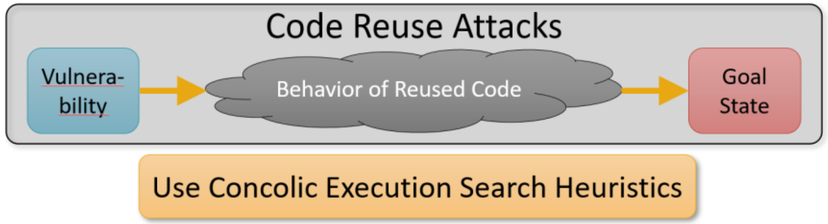
- Jump Gadget
- Call Gadget
- Move REG Gadget
- Load MEMORY Gadget
- Write MEMORY Gadget
- …

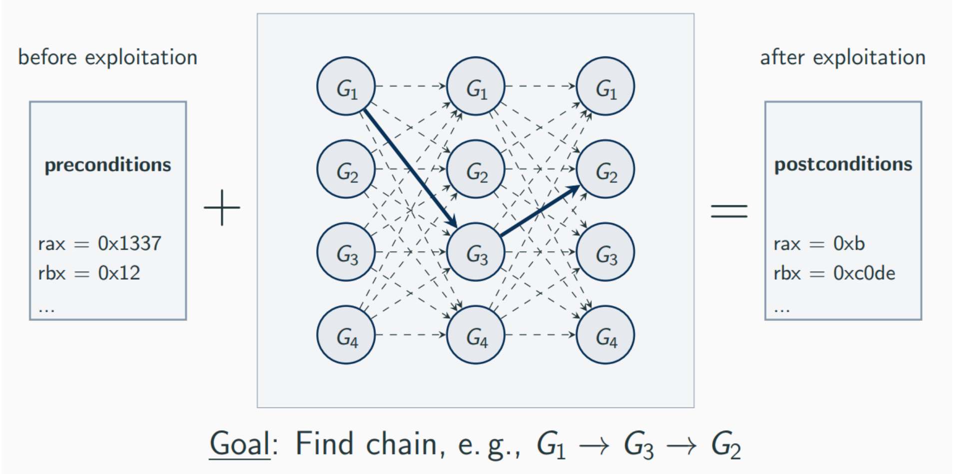


Chain through MOV-connection Graph or Rules



Exploratory-Based Automation





Exploring Chains Based on Concolic Execution OR SAT/SMT Solver

After 15 Years: Why Are We Even Talking About This?

CCS 2007
Hovav Shacham's First Time Proposal of **ROP Attacks**!

USENIX 2011
Edward J. Schwartz's Initial Proposal of **Automated ROP Exploitation!**

Open-Source Tool：
AngRop、Exrop、Ropium

CCS 20，Esorics 21：
Limbo、SGC

?

Why Are We Even Talking About This ?

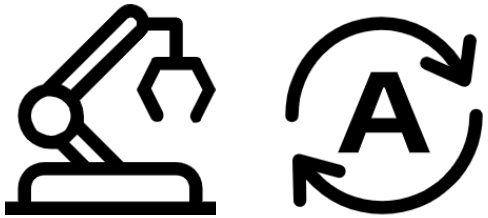
return-oriented programming

Manual ROP

Heuristic-Based Automation

Exploratory-Based Automation

After 15 Years: Why Are We Even Talking About This?



Heuristic-Based Automation

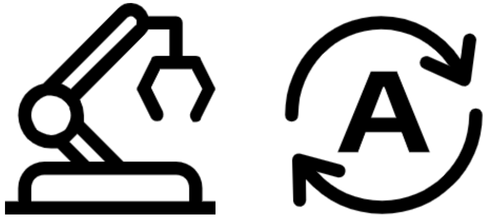
Simple Gadgets

Single ROP Exploit

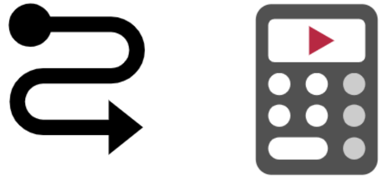
Complex Gadgets

Diverse ROP Exploits

After 15 Years: Why Are We Even Talking About This?



Heuristic-Based Automation



Exploratory-Based Automation

Discard Partials, Not Seeking All Gadget Combinations

Gadg

- W
- W
- W
- …

Discard Complex Gadgets → Can't Create Magic!

rsi;

;

sub rdx, r8; sar r8, 63; ret;

Exploration is SL.OW...

More Gadgets?

Concolic Execution And SMT Solver

Turn Slow…

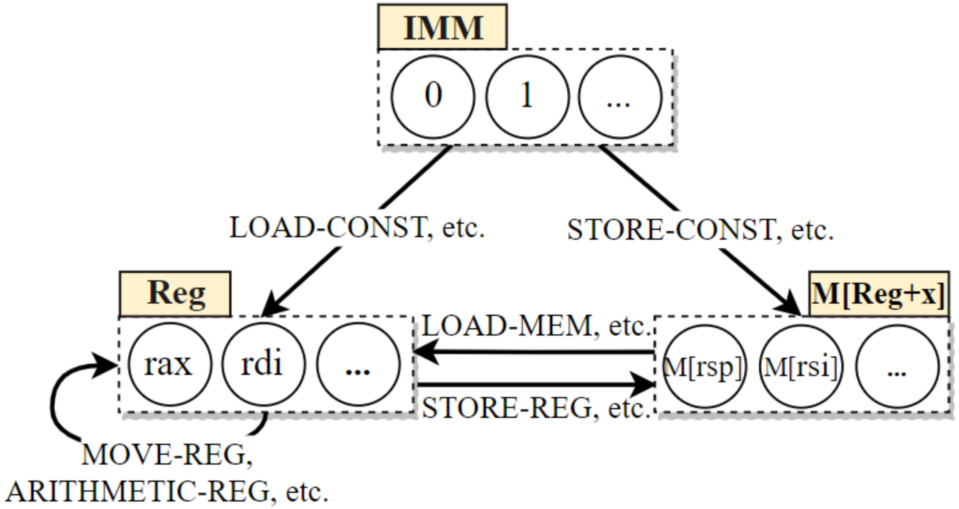
Our Goals --- Find All Gadget Combinations Using Heuristic-Based Method

TGRop – Try to put all gadgets to graph

Considering Complex Gadgets

Semantic Name	Src Operand	Dst Operand	Semantic Definition
MOVE-REG	InReg		OutReg ← InReg
LOAD-CONST	IMM		OutReg ← IMM
ARITHMETIC-REG	$\mathtt{InReg}_1$, $\mathtt{InReg}_2$	OutReg	OutReg ← $\mathtt{InReg}_1 \diamond_b \mathtt{InReg}_2$
LOAD-MEM	M[AddrReg+x]		OutReg ← M[AddrReg+x]
ARITHMETIC-LOAD-MEM	M[AddrReg+x], OutReg		$\mathtt{OutReg}_{\diamond b}$ ← M[AddrReg+x]
STORE-REG	InReg		M[AddrReg+x] ← InReg
ARITHMETIC-STORE-REG	InReg, M[AddrReg+x]	M[AddrReg+x]	$\mathtt{M[AddrReg+x]}_{\diamond b}$ ← InReg
STORE-CONST	IMM		M[AddrReg+x] ← IMM
ARITHMETIC-STORE-CONST	IMM, M[AddrReg+x]		$\mathtt{M[AddrReg+x]}_{\diamond b}$ ← IMM
MOVE-REG_{pc}	Reg		PC ← Reg
$\text{ARITHMETIC-REG}_{pc}$	$\mathtt{Reg}_1$, $\mathtt{Reg}_2$		PC ← $\mathtt{Reg}_1 \diamond_b \mathtt{Reg}_2$
LOAD-MEM_{pc}	M[AddrReg+x] (AddrReg!=sp)	PC	PC ← M[AddrReg+x] (AddrReg!=sp)
LOAD-STACK_{pc}	M[AddrReg+x] (AddrReg==sp)		PC ← M[AddrReg+x] (AddrReg==sp)
SYSCALL	Syscall#, SyscallParameters		SYSCALL(SyscallNumber, SyscallParameters)
IF	Condition, Gadget1, Gadget2		if(Condition):PC ← Gadget1 else: PC ← Gadget2

Gadget Computation Graph (GCG)



① Gadget Classification, Including Complex Gadgets.

② Building GCG Based on All Gadgets Node: operands; Edge: Data-Flow direction

Our Goals --- Find All Gadget Combinations Using Heuristic-Based Method

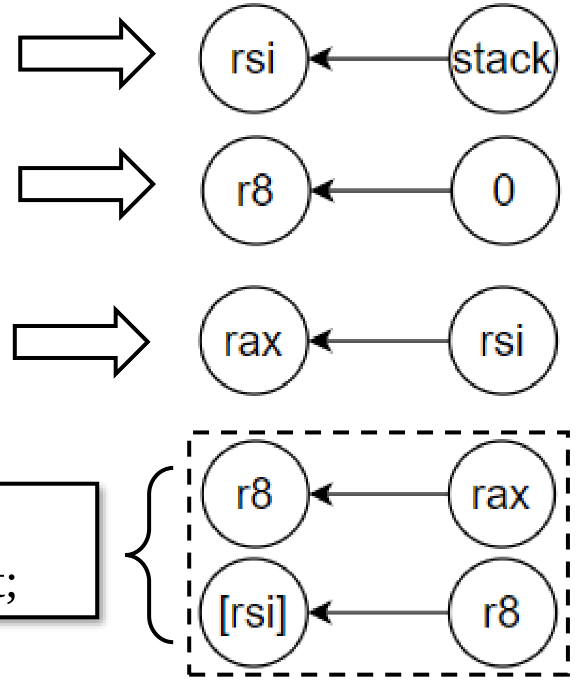
Motivation Example： Set R8 = 3

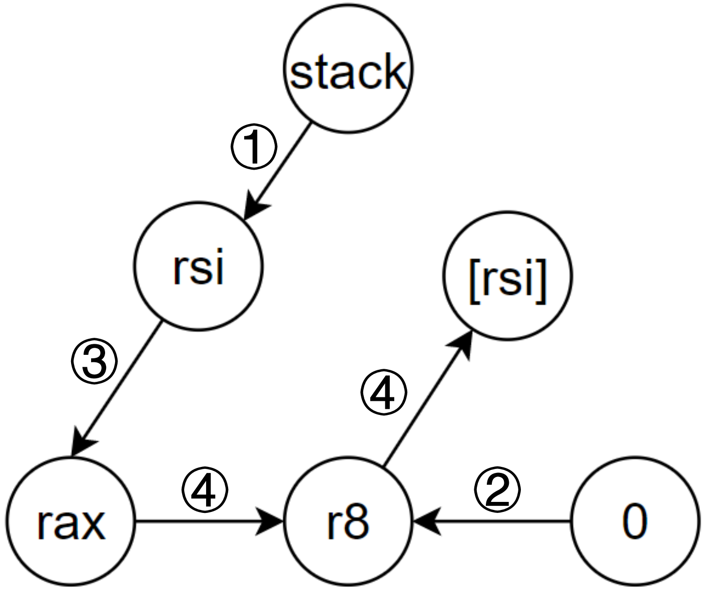
① pop rsi; ret;

② xor r8d, r8d; ret;

③ mov rax, rsi; ret;

④ add r8, rax;
mov qword ptr [rsi], r8; ret;





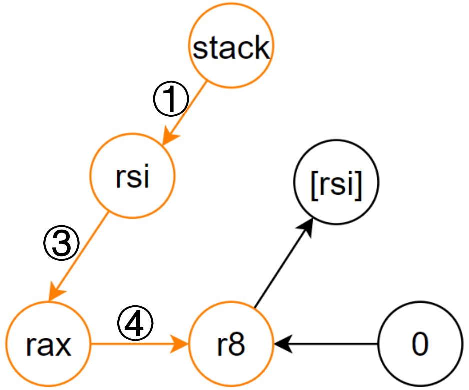
① Fully Defining Gadget Semantics, Including Complex Gadgets.

**② Building GCG Based on All Gadgets
Node: operands; Edge: Data-Flow**

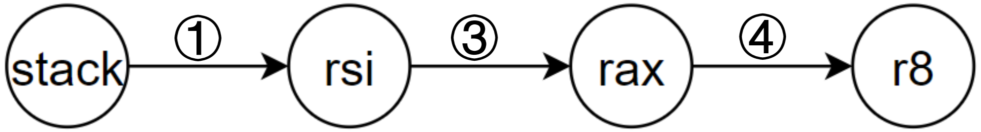
Traverse GCG To Get All Gadget Combination!

We will encounter some troubles…

- **Data dependency**: a register or memory cell must be set with the desired value.



Traverse and get only path to
set R8 = 3



① pop rsi;

③ mov rax, rsi;

④ add r8, rax;
mov qword ptr [rsi], r8;

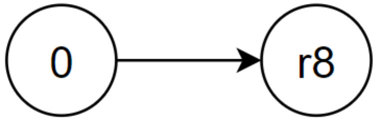
rsi = ?

rax = rsi

r8 = r8’ + rax

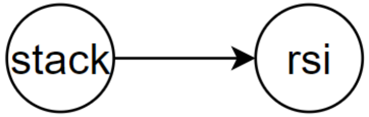
rsi = 3, r8 = 0

SMT Solver



② xor r8d, r8d;

Get one path to set **R8 = 0**



① pop rsi;

Get one path to set **RSI = One Addr**

① pop rsi;

③ mov rax, rsi;

④ add r8, rax;
mov qword ptr [rsi], r8;

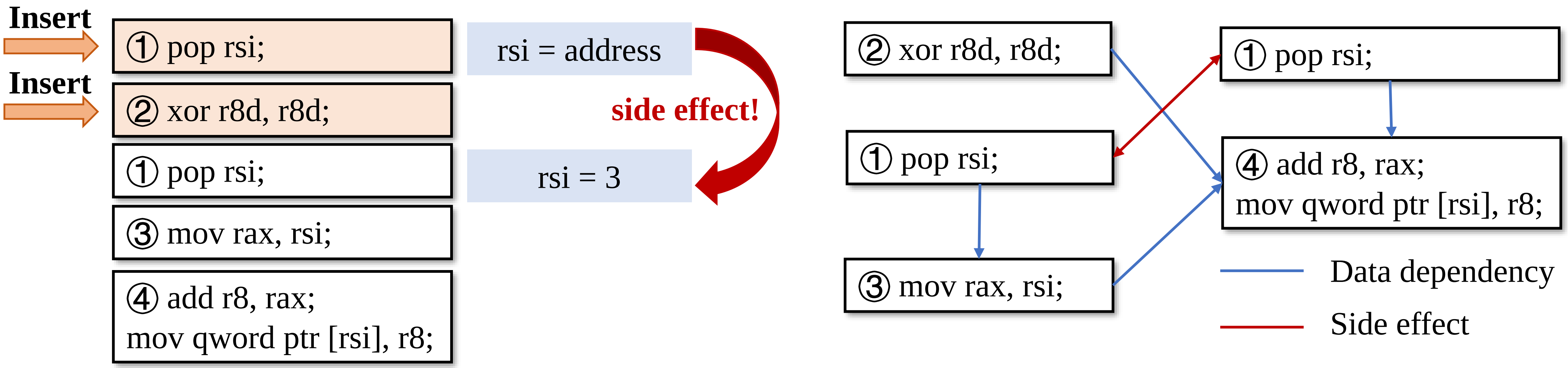
Data dependency: r8 = 0

Data dependency: rsi = One Address

- r8 = r8+3
- rsi require address

We will encounter some troubles…

- **Data dependency**: a register or memory cell must be set with the desired value.
- **Side effect**: a desired context built by an earlier gadget is tampered by a later gadget.



Identify And Solve Them.

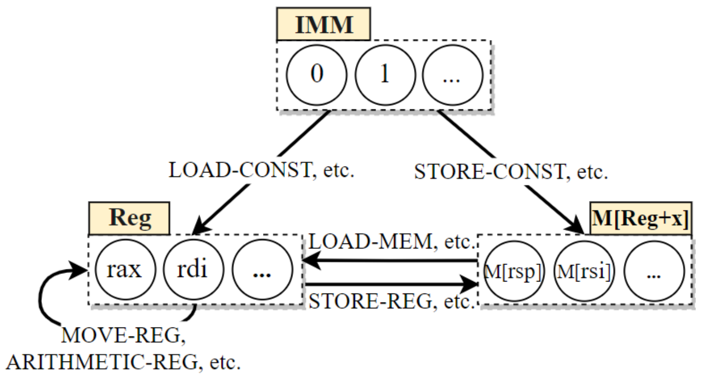
We will encounter some troubles…

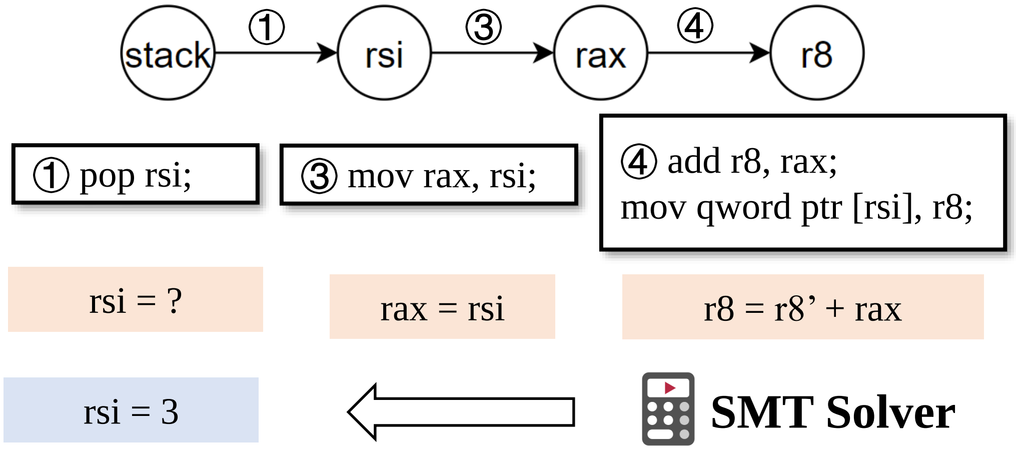
Challenges: Find All Gadget Combinations Based on the GCG

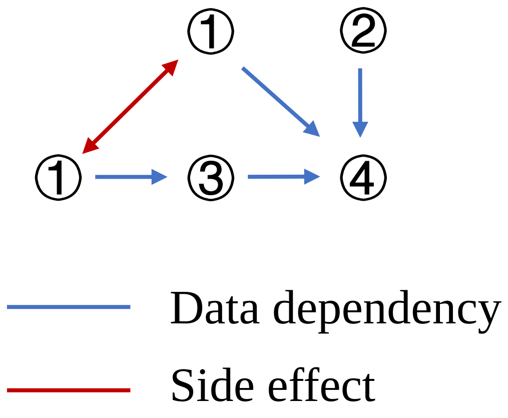
→ How to put all complex gadgets to GCG?

→ How to reduce the search space of SMT Solver?

→ How to resolve Data Dependencies and eliminate Side Effects in Gadgets?

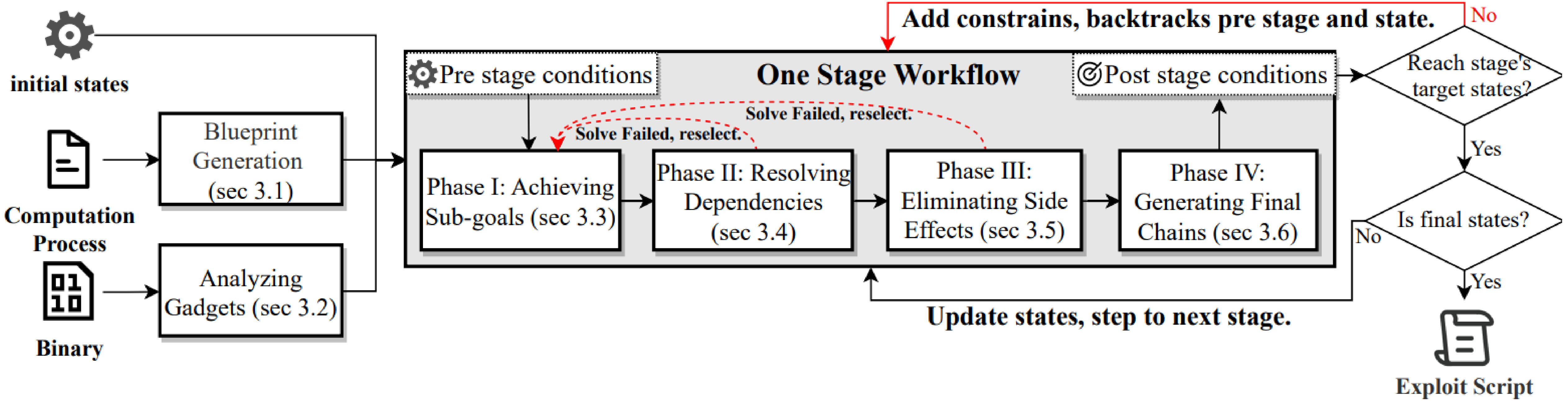


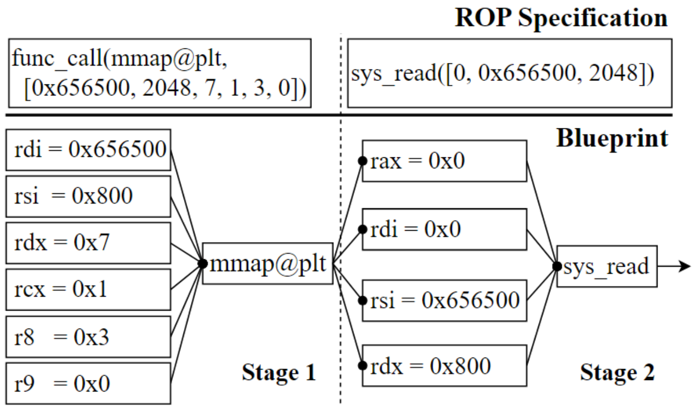




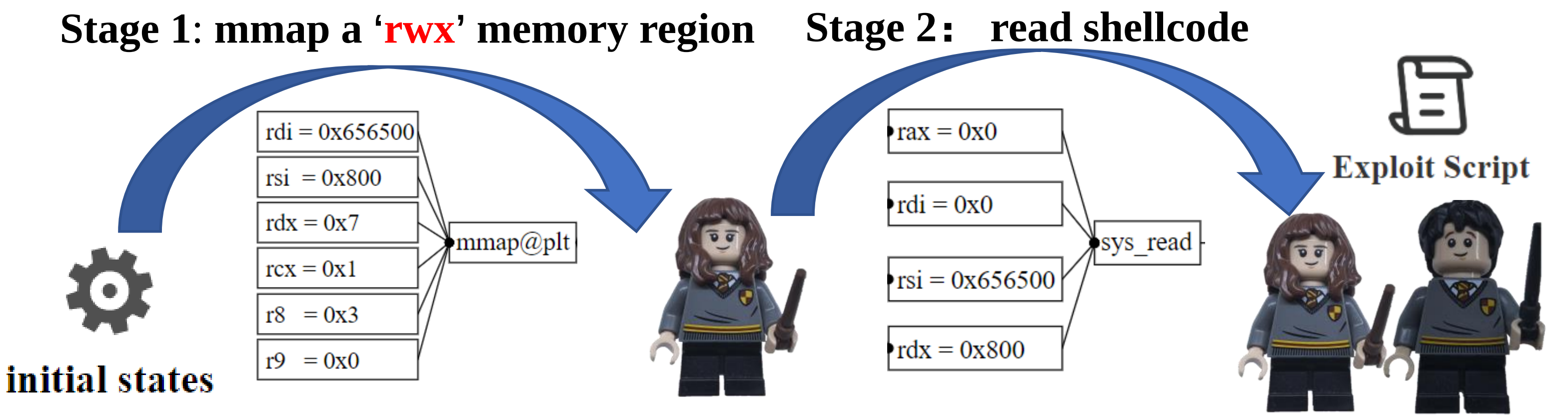
We Use a Systematic Approach to Address Them!

Our Method - Overview





Exploitation Buleprint Example



Guided by a blueprint, constructs ROP Stage-by-stage

Our Method - Preprocessing

Blueprint Generation

TGRop's support interfaces

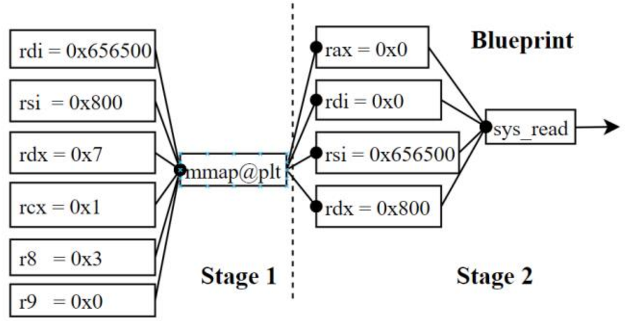
Sub-goal
Stage

1. Register write
2. Memory read
3. Memory write
4. Function call
5. System call

```
func_call(mmap@plt,
  [0x656500, 2048, 7, 1, 3, 0])
sys_read([0, 0x656500, 2048])
```

ROP Specification

Multiple Stages
Multiple Sub-goals



Blueprint Generation

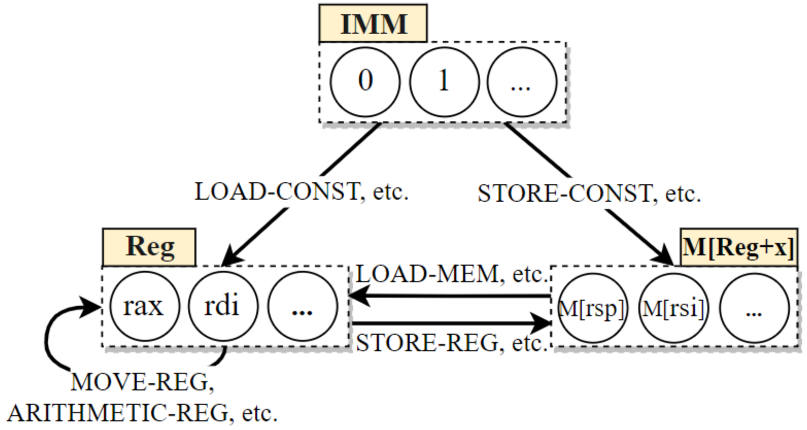
Analyzing Gadgets

Semantic Name	Src Operand	Dst Operand
MOVE-REG	InReg	OutReg
LOAD-CONST	IMM	
ARITHMETIC-REG	$InReg_1$, $InReg_2$	
LOAD-MEM	M[AddrReg+x]	
ARITHMETIC-LOAD-MEM	M[AddrReg+x], OutReg	
STORE-REG	InReg	M[AddrReg+x]
ARITHMETIC-STORE-REG	InReg, M[AddrReg+x]	
STORE-CONST	IMM	
ARITHMETIC-STORE-CONST	IMM, M[AddrReg+x]	
MOVE-REG_{pc}	Reg	PC
$\text{ARITHMETIC-REG}_{pc}$	Reg_1, Reg_2	
LOAD-MEM_{pc}	M[AddrReg+x] (AddrReg!=sp)	
LOAD-STACK_{pc}	M[AddrReg+x] (AddrReg==sp)	
SYSCALL	Syscall#, SyscallParameters	
IF	Condition, Gadget1, Gadget2	

Symbolic execution identify the semantic

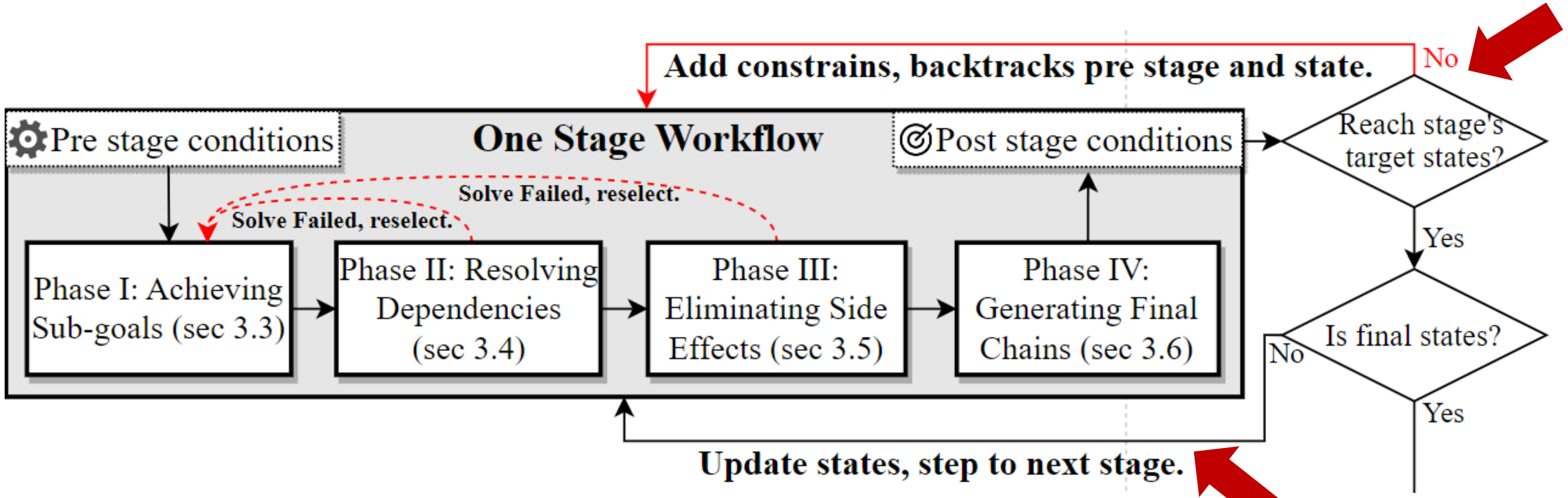
- Classification based one semantic
- Each semantic has src and dst operand
- Src operand can be multiple
- Dst is a single operand
- Symbolic Representation of Operators

Node: operand **Edge: Data-Flow**



Gadget Computation Graph Building

Our Method



Guided by a blueprint, TGRop constructs ROP chains stage-by-stage

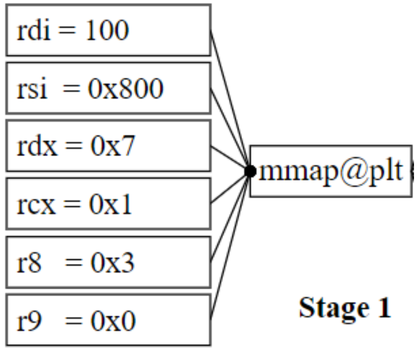
- Phase I: Achieving Sub-goals
- Phase II: Resolving Dependencies
- Phase III: Eliminating Side Effects
- Phase IV: Generating Final Chains

Each stage completes a function/system call, with independent ROP generation.

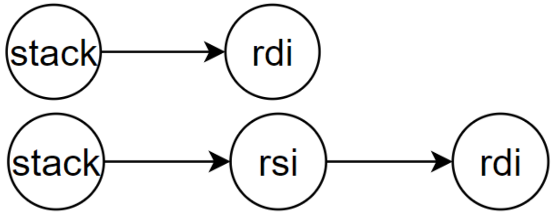
- No dependencies or side effects between stages
- If a stage wins, update states, step to the next stage
- If a stage fails, roll back and switch states

Our Method - Phase I: Achieving Sub-goals

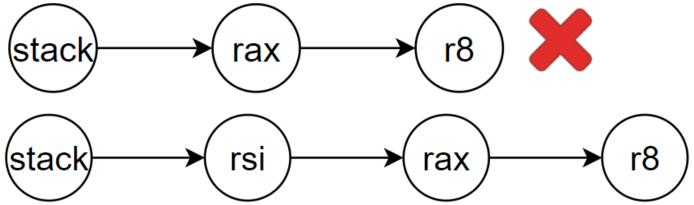
Pre & Post condition	GCG Traverse → Paths	Tentative Chaining



Pre	rdi = 0x414141
Post	rdi = 100



Pre	r8 = 0x7fffffffdbc8
Post	r8 = 3



stack → rdi:
- pop rdi; ret;

stack → rsi:
- Pop rsi; ret;

rsi → rdi:
- Mov rdi, rsi; ret;

Tentiative chain by SMT Solver

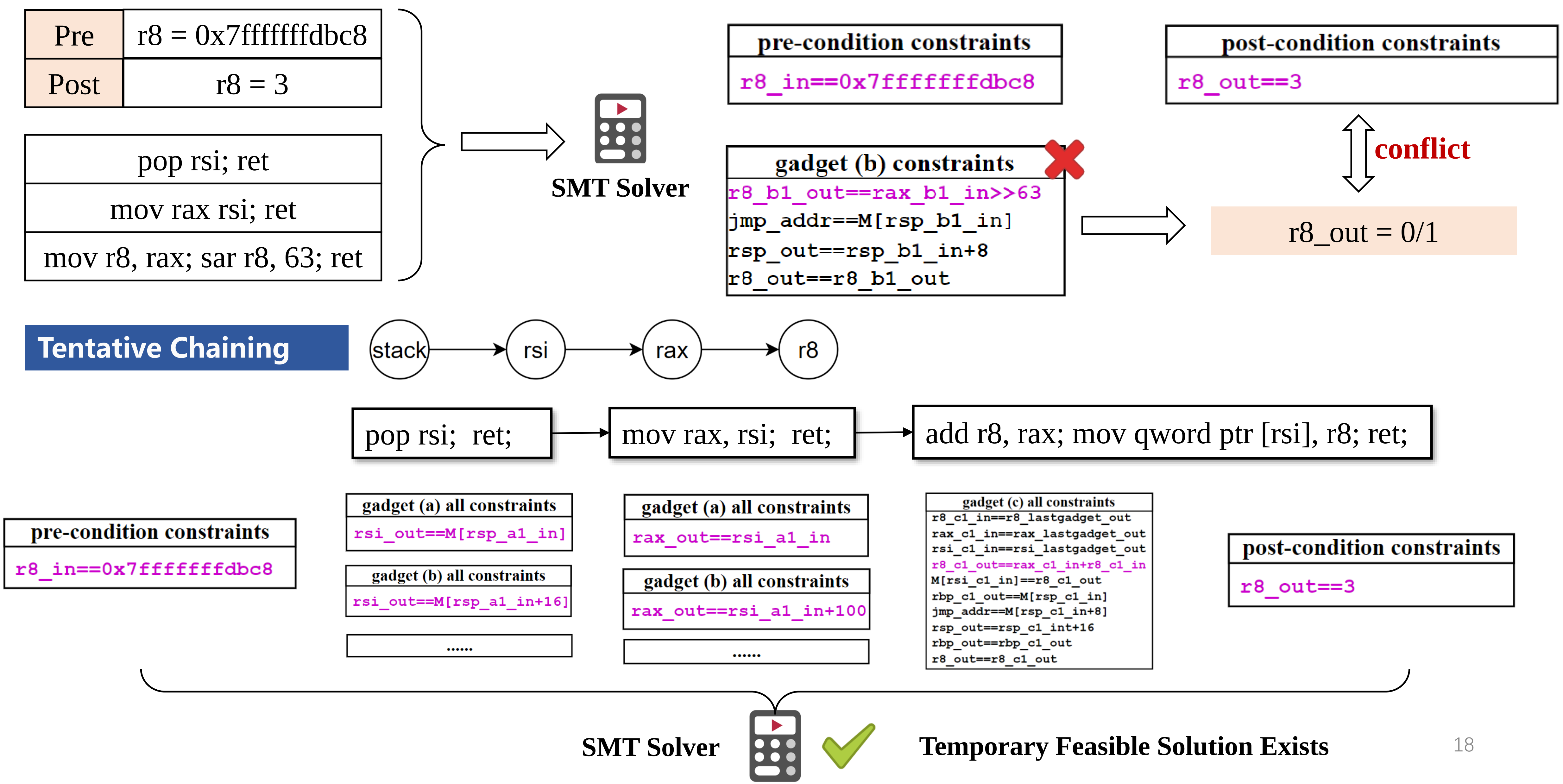
1 pop rdi; ret

1 pop rsi; ret; 2 mov rdi, rsi; ret;

stack -> rax:
- pop rax;
  or byte ptr [rax - 0x77], cl;
  ret;

rax → r8:
- mov r8, rax; jmp r8
- mov r8, rax; sar r8, 63; ret
- add r8, rax; mov [rsi], r8;
  pop rbp; ret

Our Method - Phase I: Achieving Sub-goals



Our Method – Phase II: Resolving Dependencies

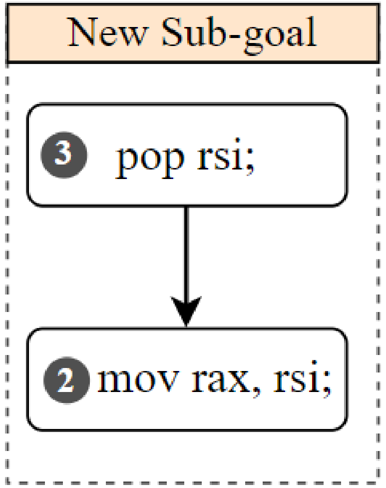
Data dependency: a register or memory cell must be set with the desired value.

- **Valid Address**: Requires the valid Address stored in a register `mov [rdi], r8`
- **Correct Target**: The next instruction to execute is the start of the desired gadget `jmp rax`
- **Suitable Value**: The value of a source operand is suitable so that after computation, the value of a destination operand is correct. `add r8, rax`

jmp rax

New Sub-goal:
rax = next gadget addr

Phase I + Phase II

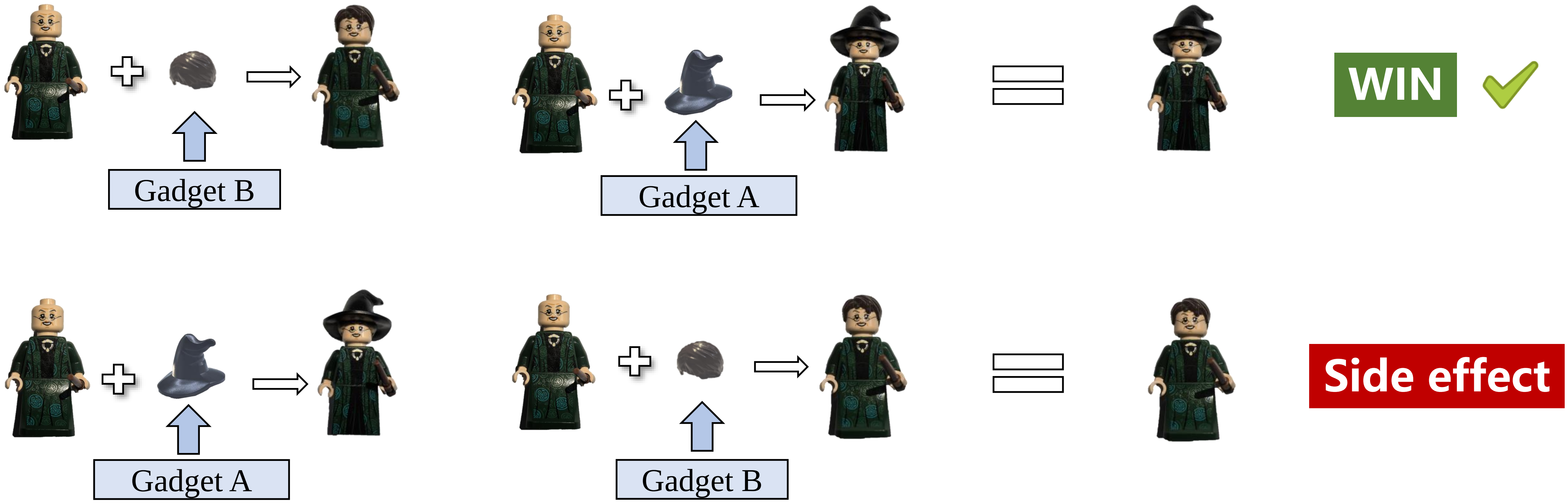


Data dependency

Iteratively Generate Each Sub-goal to Solve Data Dependency

Our Method – Phase III: Eliminating Side-Effects

Side effect: a desired context built by an earlier gadget is tampered by a later gadget.

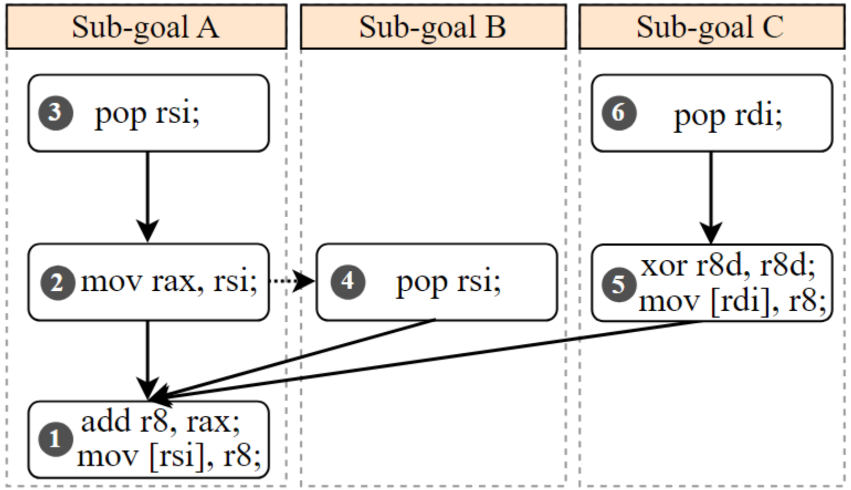


Focusing on Determining the Execution Order

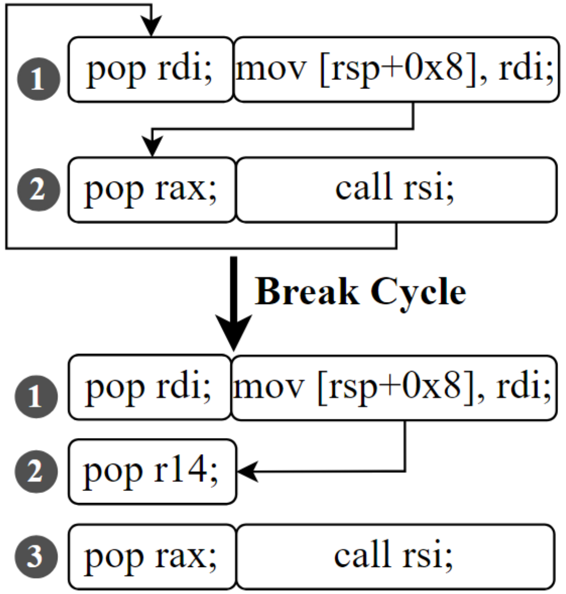
Our Method – Phase III: Eliminating Side-Effects

Side effect: a desired context built by an earlier gadget is tampered by a later gadget.

- Build topology to eliminate side effects
- Special case: Topology containing cycles.



Topology's direction:
The desired execution order or write-read/ read-write pair



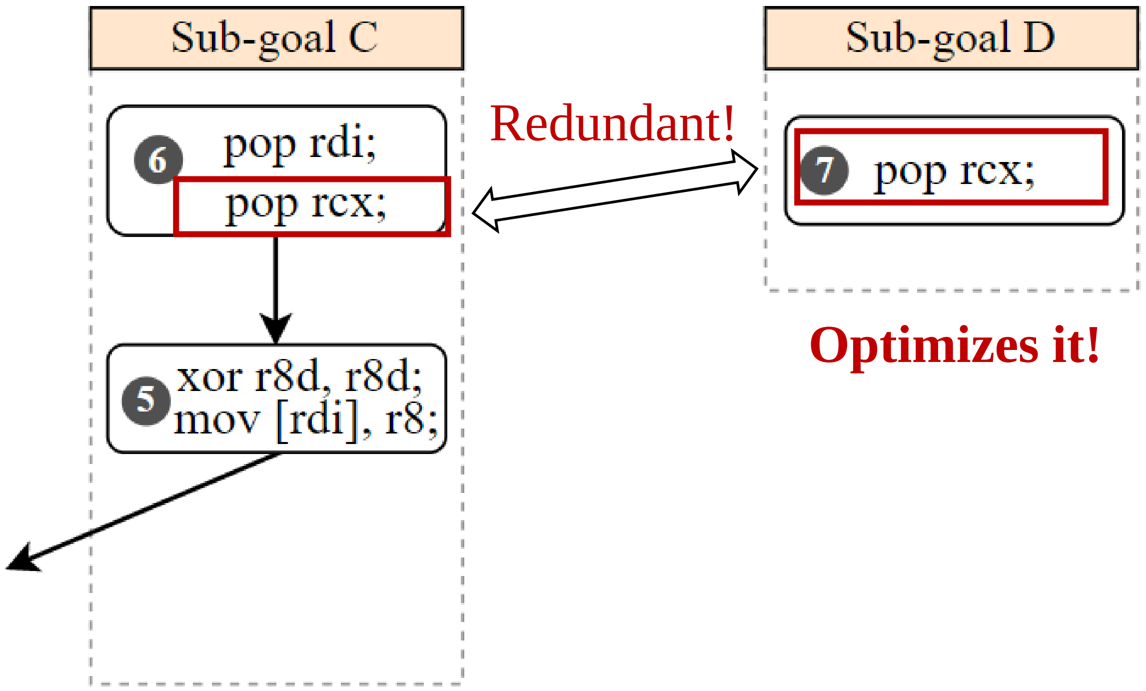
Special case: Insert a gadget to adjust the stack.

Then, topological sorting can eliminate side effects

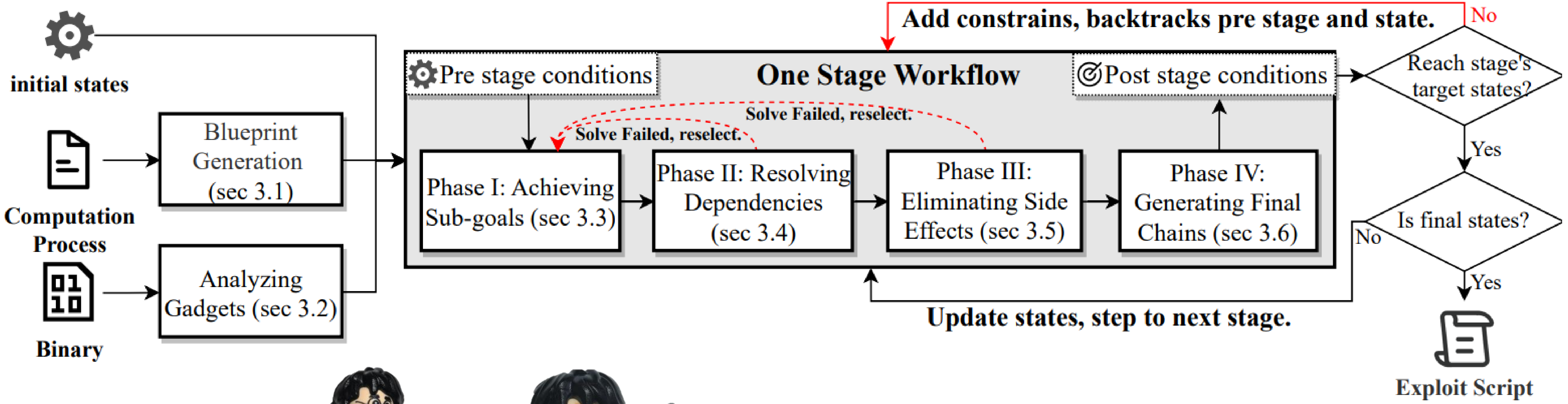
Our Method – Phase IV Generating Final Chains

Choose a better gadget chaining solution

- Optimizes the chain by pruning redundant gadgets!
- Perform topological sorting and SMT solver to generate gadget chain !
- Steps into the next stage, repeating Phases I, II, and III to produce another chain!

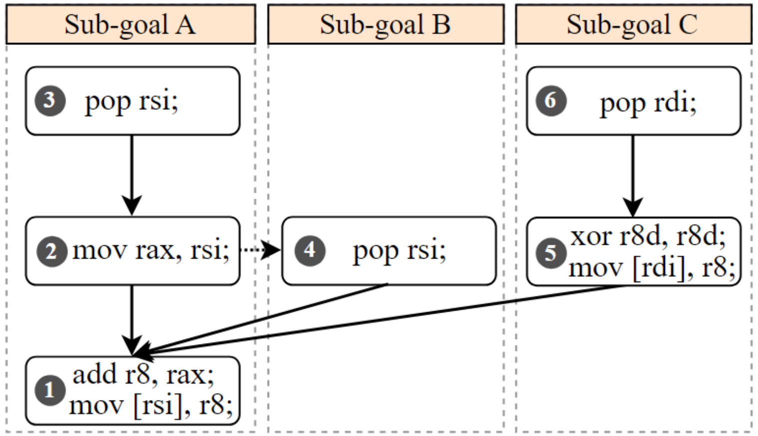


Our Method – Core Implementation



sub-goals | **stage**

Data dependency



Topology → Side effect

Evaluation --- Experiment Setup

Baselines: Six open-source SOTA Tools

- Hardcode-based approach: Ropper, ROPgadget;
- Heuristic-based approach: Angrop, Ropium, Exrop;
- Exploratory approach: SGC;

Test suites:

- All programs shipped in Debian 10, CentOS 7, and OpenBSD 6.2, from ROP-Benchmark.
- Different architectures programs in Firmwares: MIPS, ARM, and PowerPC.
- All programs in OpenBSD 6.5 and 50 programs with ROP Mitigations.

ROP Goals:

- Goal #1: Get a Shell
- Goal #2: Arbitrary Write
- Goal #3: Set Three Registers
- Goal #4: Set Four Registers
- Goal #5: Set Five Registers
- Goal #6: Set Six Registers

Others:

- Ubuntu 20.04 machine with an Intel i9-10900X 3.70GHz20-core and 128GB RAM.
- Tools have one hour for each goal per program.

Evaluation - TGRop is outperform SOTA approaches

	Debian 10	CentOS 7	OpenBSD 6.2	TOTAL	Debian 10	CentOS 7	OpenBSD 6.2	TOTAL	Debian 10	CentOS 7	OpenBSD 6.2	TOTAL
Program #	139	121	87	347	139	121	87	347	139	121	87	347
	Goal #1: Get a Shell				Goal #2: Arbitrary Write				Goal #3: Set Three Registers			
ROPgadget	5/0/0	4/0/0	4/0/0	3.75%	NA	NA	NA	/	NA	NA	NA	/
Ropper	53/NA/0	31/NA/0	17/NA/1	29.11%	NA	NA	NA	/	NA	NA	NA	/
Exrop	55/3/6	48/6/9	6/7/12	31.41%	67/4/5	72/7/6	7/20/9	42.07%	85/0/1	71/1/2	38/0/6	55.91%
Angrop	98/11/2	70/5/1	33/4/5	57.93%	85/4/29	90/3/5	48/1/12	64.27%	58/3/22	54/5/3	19/3/9	37.75%
ROPium	100/12/1	66/9/0	43/6/1	60.23%	122/5/0	100/1/0	57/3/2	80.40%	104/5/0	73/3/0	43/4/1	63.40%
SGC	17/NA/38	20/NA/30	9/NA/33	13.26%	25/NA/54	18/NA/83	15/NA/45	16.71%	58/NA/41	63/NA/38	60/NA/24	52.16%
TGRop	138/0/1	117/0/0	84/0/1	97.69%	138/0/1	121/0/0	86/0/1	99.42%	138/0/1	121/0/0	86/0/1	99.42%

TGRop is 1.62 to 27 times better.

- For goal #1, TGRop Successfully Generates ROP for 97.69% of Programs
- At Least a 1.62x Improvement in Success Rate Over Existing SOTA Tools
- No False Positives, Only 2 Programs Timed Out (File Size Exceeds 26M)

Evaluation - TGRop is outperform SOTA approaches

Arch	Target	Program #	Program # with GCG paths	Results
MIPS	Archer A54v1	78	45	36/0/0
	blink X12	74	30	25/0/0
	Cisco RV110W	218	110	75/0/4
	TL-WR841Nv14	59	31	24/0/0
	TL-WR902ACv3	82	42	33/0/0
	debian mipsel	624	272	223/0/14
ARM	NETGEAR R8500	7	7	6/0/0
	Tenda G3	7	7	5/0/0
	Tenda W20EV4	6	6	3/0/0
PowerPC	libc-2.31.so	1	1	1/0/0
Total		1156	551	431/0/18

	Human A	Human B	Human C	Human D	Human E	Human F	Human G	Human H	Human I	TGRop
smbclient	✔/2917	✔/3308	✔/2191	✔/2821	✔/1952	✔/3276	✔/2886	✔/3290	✔/3425	✔/984
wuftpd	✔/2166	✗/NA	✗/NA	✔/3320	✔/3398	✔/3198	✔/2857	✗/NA	✗/NA	✔/168
vmd	✔/3054	✗/NA	✗/NA	✔/3249	✔/2720	✔/3502	✗*/3814	✗/NA	✗/NA	✔/177
sudo	✔/2796	✗/NA	✗/NA	✗/NA	✗/NA	✗*/3808	✗*/4161	✔/2730	✗/NA	✔/143
install-info	✗/NA	✗/NA	✗/NA	✗/NA	✗/NA	✗/NA	✗*/5737	✗/NA	✗/NA	✔/100
libnetsnmp.so.40.2.1	✔/3410	✔/3521	✗/NA	✗/NA	✔/2647	✗*/3712	✔/2899	✗/NA	✗/NA	✔/561
mpathpersist	✗/NA	✗*/3812	✗/NA	✗/NA	✔/2863	✔/2788	✔/3199	✗/NA	✗/NA	✔/82
mount_nfs	✗/NA	✗/NA	✗/NA	✗/NA	✗/NA	✔/3094	✔/2546	✗/NA	✗/NA	✔/183
ftp	✔/3313	✗/NA	✗/NA	✗/NA	✔/2091	✗*/6061	✗*/3710	✗/NA	✗/NA	✔/126
libwget.so.2.0.0	✔/2381	✔/3498	✗/NA	✗/NA	✔/2960	✔/2418	✗*/3918	✔/1437	✗/NA	✔/584
Total/Avg.	7/2862	3/3535	1/2191	3/3130	7/2662	6/3540	5/3573	3/2486	1/3425	10/310.8

TGRop also works on multi-arch programs.

Faster and More Accurate Than Humans

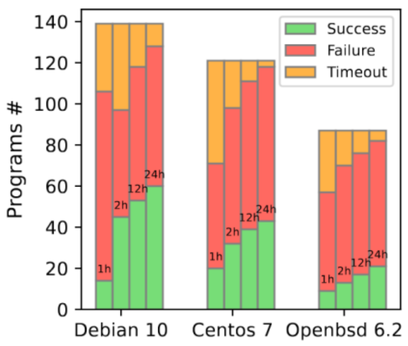
Evaluation - Ablation Analysis of TGRop

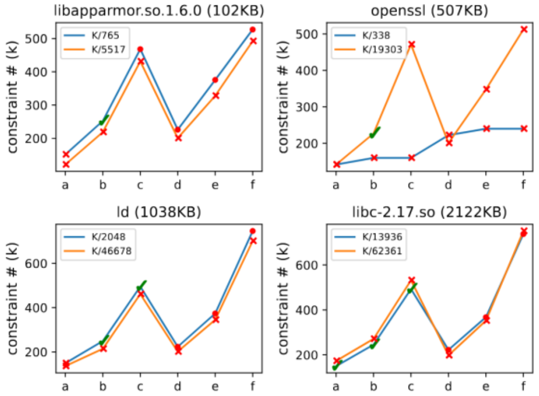
- **GCG Helps**: TGRop leverages all program gadgets, utilizing **81.66% more gadgets** then other tools.
- **Resolve Dependencies Matters**: **64.7%** of Programs Have Data Dependency Issues

	Program #	Valid Address	Correct Target	Suitable Value
OpenBSD 6.2	32	6	9	25
OpenBSD 6.4	52	9	27	46
OpenBSD 6.5	118	4	53	112
Total	202	19	89	183

Only TGRop Succeeds: 202 of 312 Programs

C: Chain Length K: Candidate Gadgets T: Given Time





Exploratory Tools: SGC in different threshold settings

- **Compared with Exploratory Tools: Heuristic Approaches are More Lightweight**
  - Time is Not the Key; Even with Increased Time, SGC Still Fails in Most Programs.
  - Longer Chains or More Gadgets Increase Overhead, Easily Leading to SGC Failures or Timeouts.

Evaluation – Has TGRop discovered weaknesses in newest mitigations?

ROP Mitigations

- ‘RETGUARD’ by OpenBSD: **Cut Gadgets** by About **76%**.
- ‘-fzero-call-used-regs’ option in GCC: **Reduced Gadgets** by About **60%**.

TGRop Successfully Bypasses These Mechanisms

- TGRop uses complex gadgets, like ending in call/jmp/jnz…
- For ‘RETGUARD’, TGRop performed well in **185, 231** and **231** programs of each goal.
- For ‘GCC fzero’, TGRop succeeds in **92%** of cases.

	OpenBSD 6.4	OpenBSD 6.5	OpenBSD 7.3	GCC fzero
Program #	98	240	264	50
Goal #1: Get a Shell				
ROPgadget	2/0/0	0/0/0	0/0/0	0/0/0
Ropper	3/NA/0	4/NA/0	12/NA/1	4/NA/0
Exrop	1/7/23	0/0/13	0/0/21	3/2/12
Angrop	18/1/1	24/0/10	9/0/3	18/1/0
ROPium	20/2/0	17/0/0	12/1/0	9/0/0
SGC	7/NA/11	1/NA/17	1/NA/34	2/NA/28
TGRop	81/0/1	185/0/9	124/0/10	45/0/1
Goal #2: Arbitrary Write				
Exrop	1/25/16	0/0/13	0/0/18	3/3/11
Angrop	15/0/5	31/0/14	35/2/5	20/1/0
ROPium	24/1/2	8/1/9	21/0/6	12/0/0
SGC	18/NA/17	0/NA/29	0/NA/35	17/NA/9
TGRop	93/0/1	231/0/9	180/0/10	46/0/1
Goal #3: Set Three Registers				
Exrop	38/1/13	140/0/17	38/0/27	8/0/14
Angrop	23/3/3	43/2/13	15/0/1	17/2/0
ROPium	46/0/0	48/0/0	40/1/0	23/2/0
SGC	81/NA/10	206/NA/22	0/NA/27	12/NA/4
TGRop	97/0/1	231/0/9	126/0/10	47/0/1

Conclusion

- TGRop is a systematic approach to automating ROP chain construction, overcoming the fundamental limitations of existing works.

- TGRop outperforms all open-sourced state-of-the-art tools and demonstrates that its design principles are both rational and efficient.

- TGRop has disclosed design weaknesses in the latest ROP mitigations, which have been reported to vendors.

Others Details:

https://sites.google.com/view/tgropsdetails

中国科学院大学
University of Chinese Academy of Sciences
中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS
University of Colorado Boulder
Northwestern University

Q&A

TGRop: Top Gun of Return-Oriented Programming Automation

Contact: zhongnanyu@iie.ac.cn
Repo: https://github.com/ZoEplA/TGRop