Block Oriented Programming: Automating Data-Only Attacks

Kyriakos Ispoglou
Purdue University

Bader AlBassam
Purdue University

Trent Jaeger
Penn State University

Mathias Payer
EPFL, Purdue University

Automatic
CFI-aware
Exploitation

Introduction

- Memory corruption results in arbitrary code execution
- Mitigations: DEP, ASLR, Canaries, CFI, Shadow Stacks, ...
- Advanced mitigations call for advanced attacks
- Data-Only attacks are still possible

Approach in a nutshell

- **Perform *Code Reuse* using *Data-Only attacks***
- **Leverage a memory corruption vulnerability**
- **Build Turing-complete payloads as execution traces**
- **Express execution traces as memory writes**

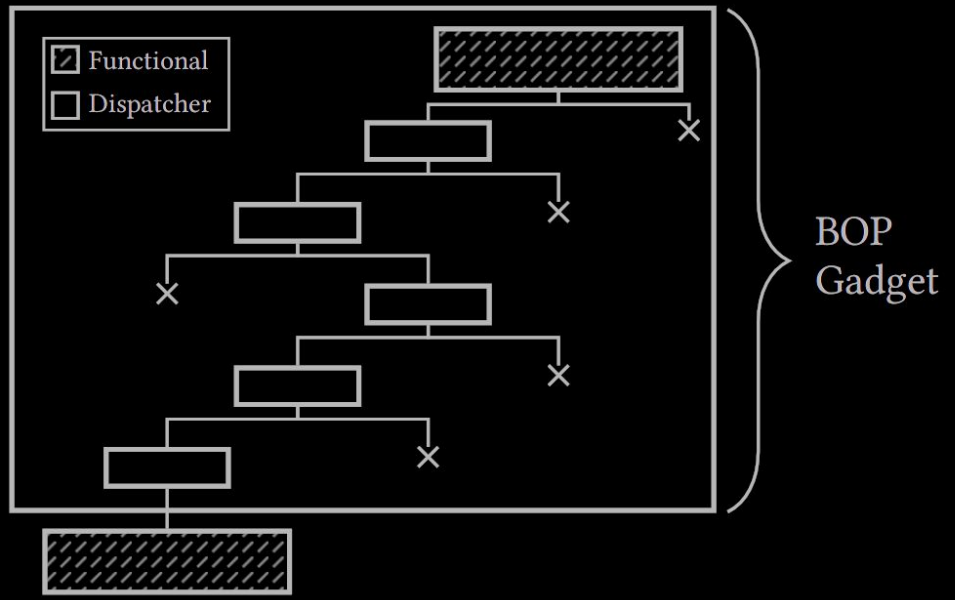
Contributions

- A language for expressing exploit payloads (SPL)
- Block Oriented Programming (BOP)
- Effective concolic execution algorithm to stitch BOP gadgets
- Open source implementation

Block Oriented Programming (BOP)

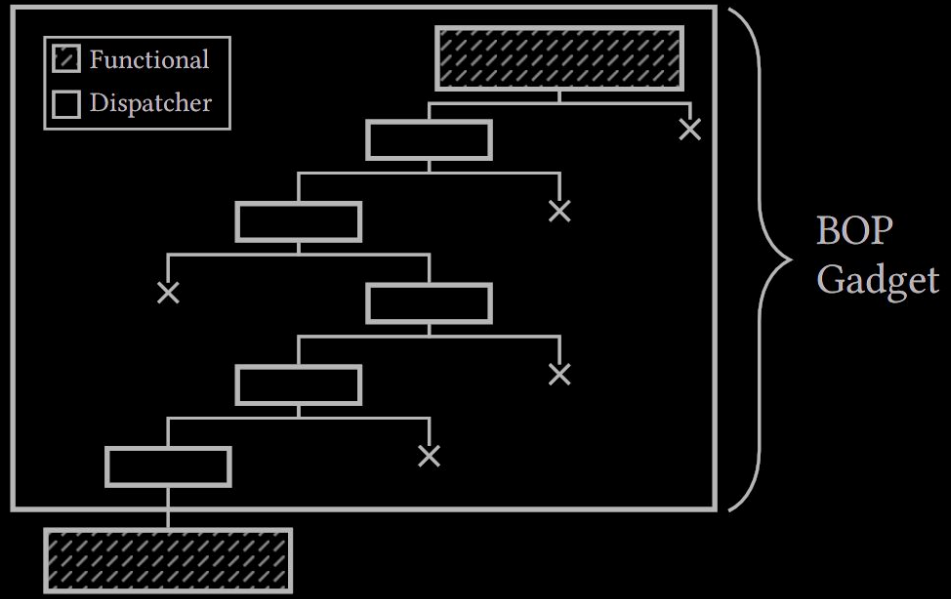
Block Oriented Programming: Gadget

- Gadget: Sequence of basic blocks
- Functional blocks
  - Perform useful computations
- Dispatcher blocks
  - Connect functional blocks
  - Avoid clobbering blocks
- Clobbering blocks
  - Destruct execution context



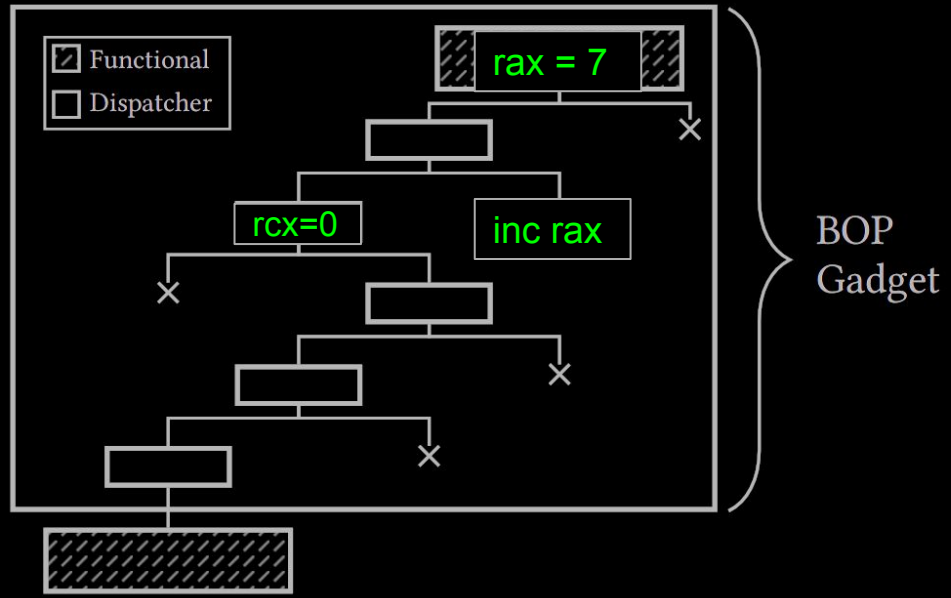
Block Oriented Programming: Example

```
rax ⟵ 7
```



Block Oriented Programming: Example

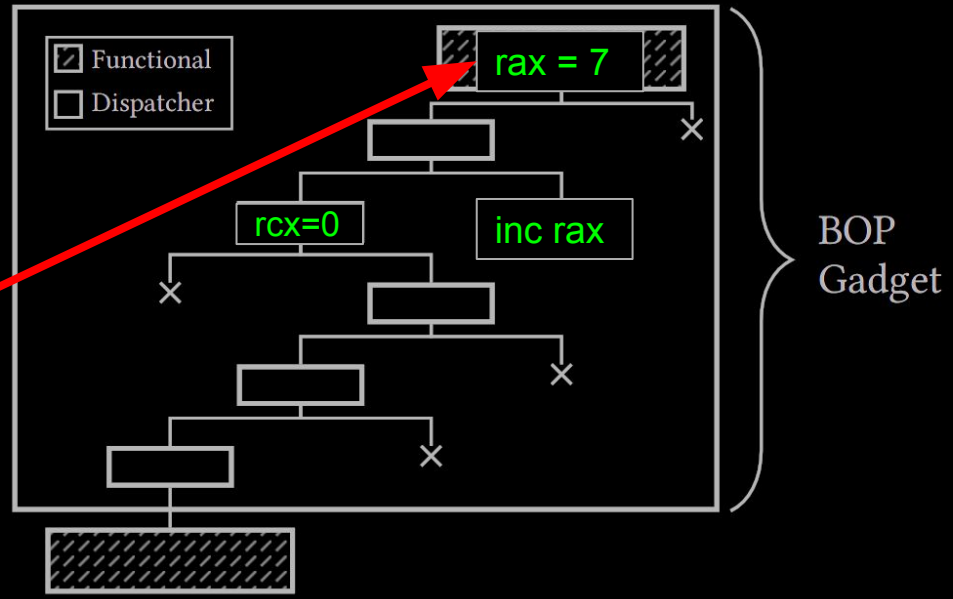
`rax` ⟵ 7



Block Oriented Programming: Example

`rax` ⟵ 7

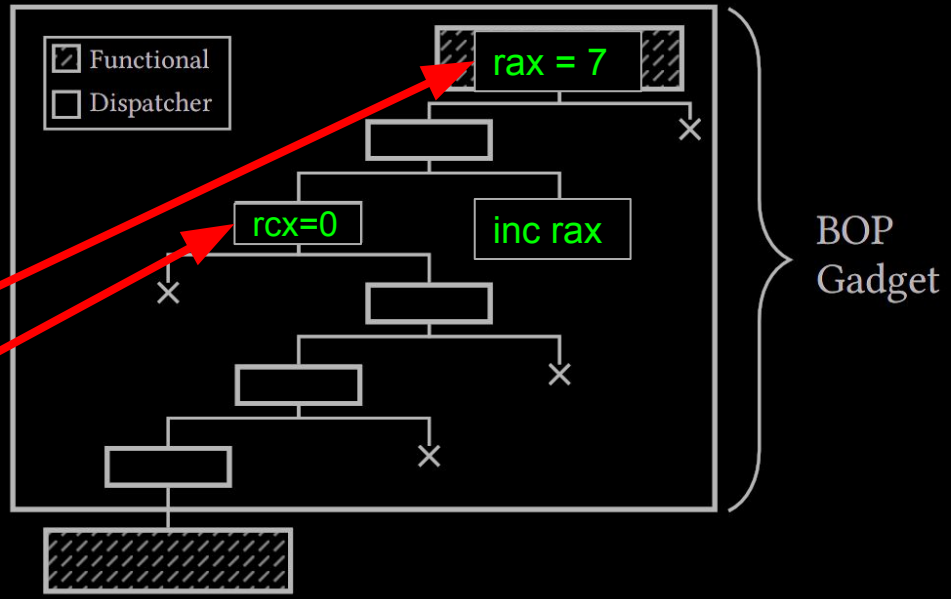
Functional



Block Oriented Programming: Example

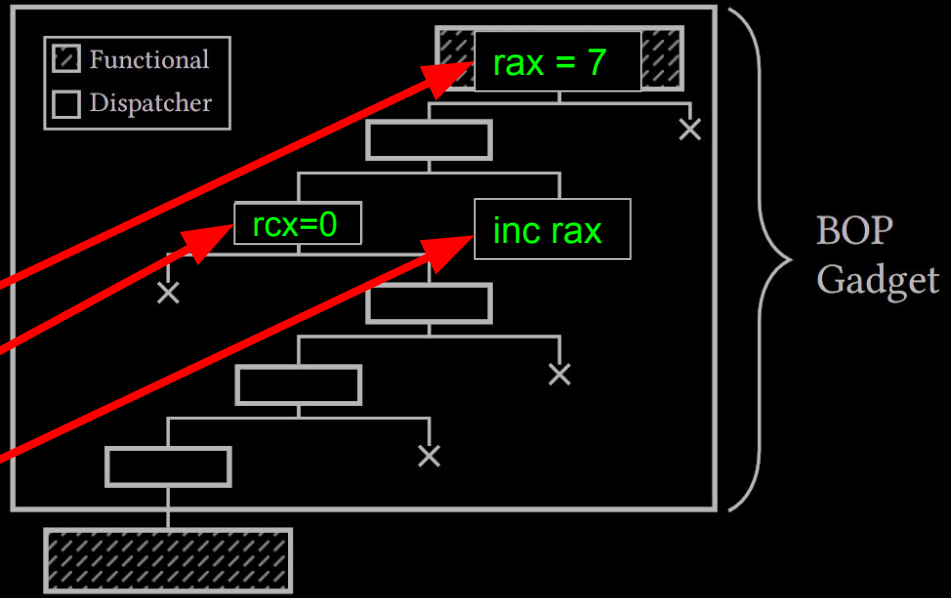
`rax` ⟵ 7

Functional
Dispatcher



Block Oriented Programming: Example

`rax` ⟵ 7



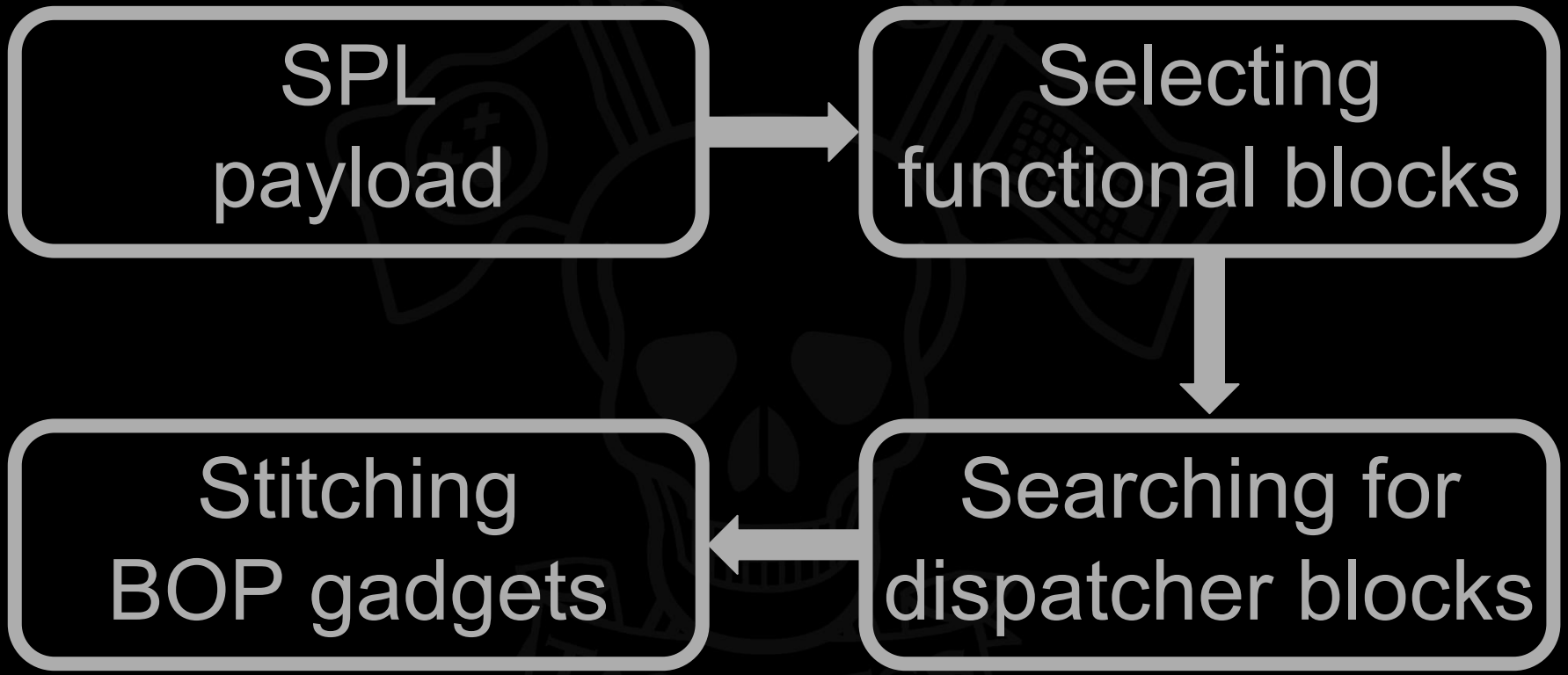
Functional
Dispatcher
Clobbering

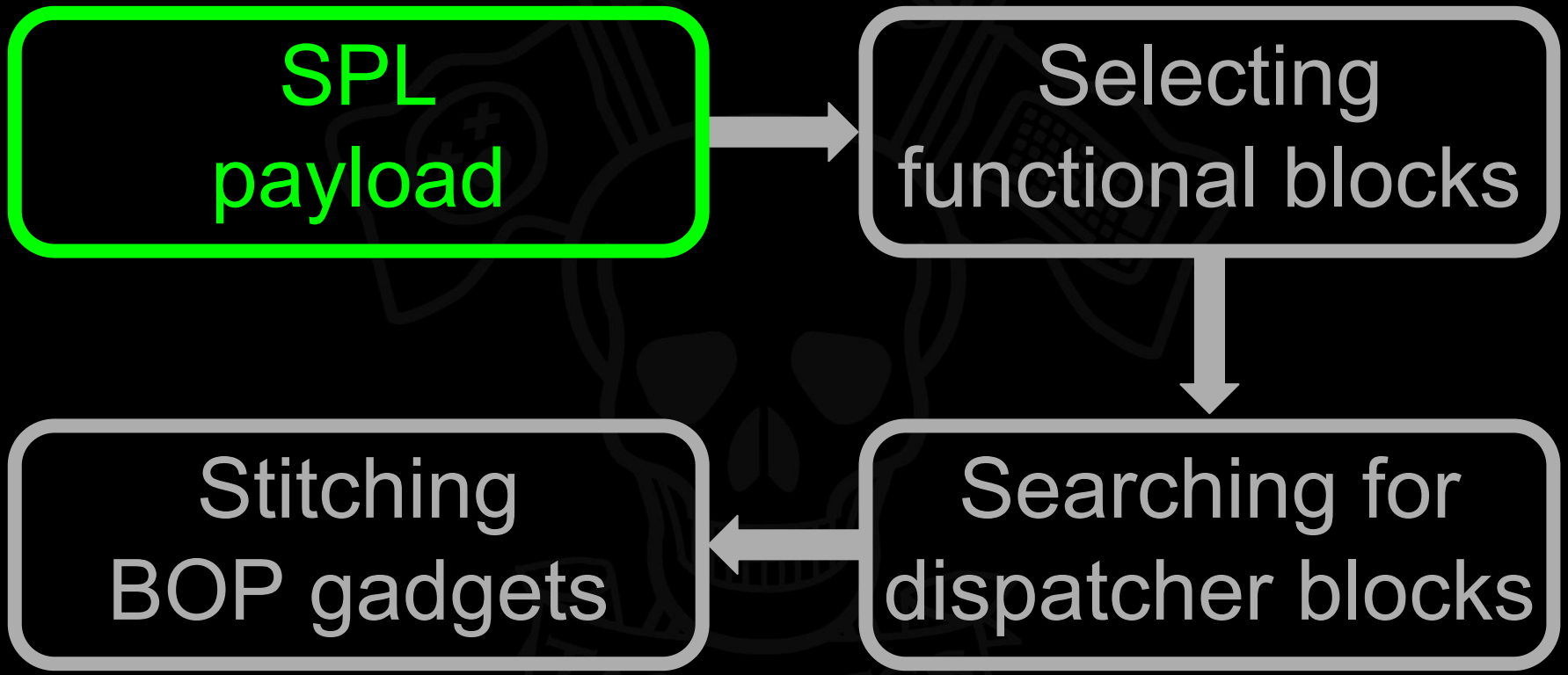
Block Oriented Programming: Concept

- Construct attacker's intended execution
  - As an "exploit program"
- Find and stitch BOP gadgets to implement "exploit program"
  - Follow CFG
- Encode execution trace as a set of memory writes
  - Data-Only Attack

Design

How it’s made


SPL payload
Selecting functional blocks
Searching for dispatcher blocks
Stitching BOP gadgets


SPL payload
Selecting functional blocks
Searching for dispatcher blocks
Stitching BOP gadgets

SPL payload

- **Payload expression language**
  - **Architecture independent**
  - **Turing-complete**

- **High level; Subset of C**
  - **Variables**
  - **Library Calls**
  - **Conditionals**
  - **…**

- **Abstract registers as volatile vars**

```
void payload() {
  string prog = "/bin/sh\0";
  int64* argv = {&prog, 0x0};

  __r0 = &prog;
  __r1 = &argv;
  __r2 = 0;

  execve(__r0, __r1, __r2);
}
```

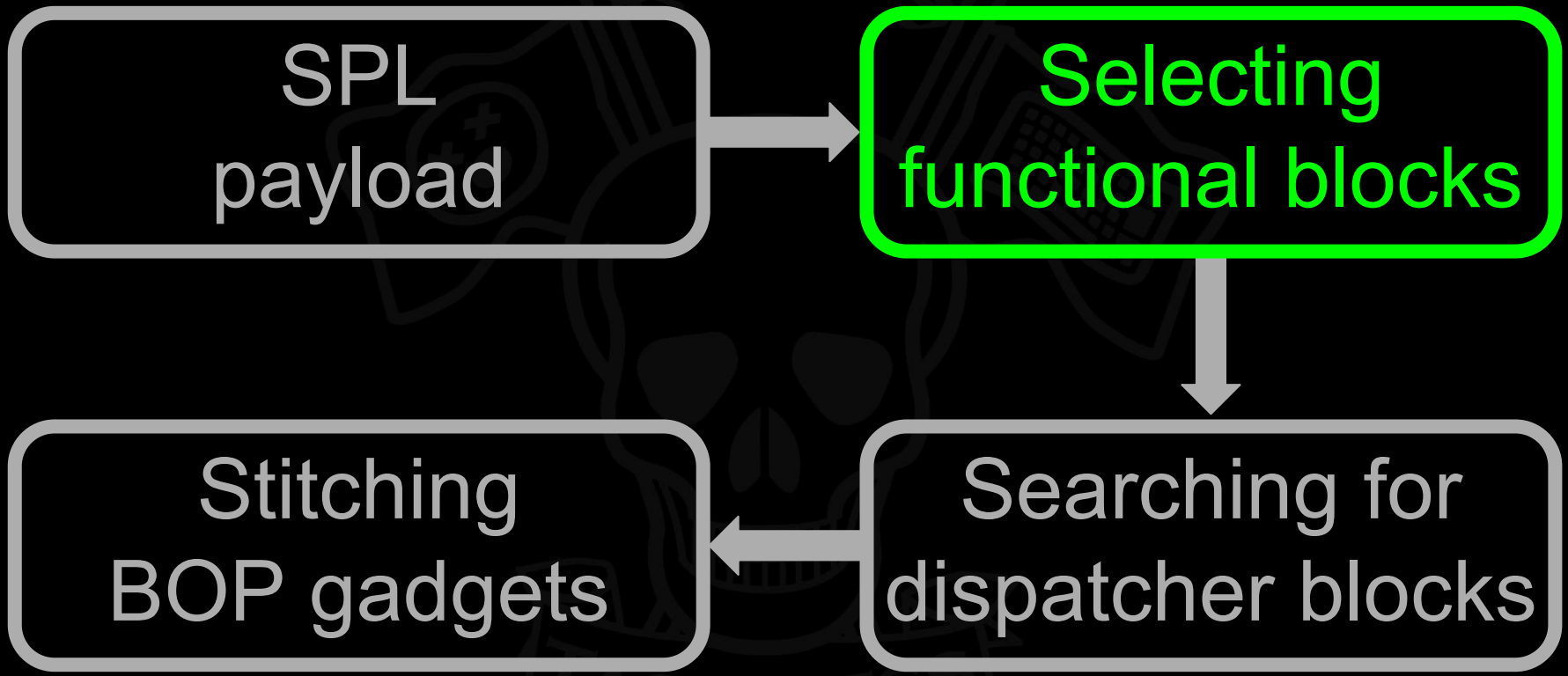
SPL payload

- Payload expression language
  - Architecture independent
  - Turing-complete
- High level; Subset of C
  - Variables
  - Library Calls
  - Conditionals
  - …
- Abstract registers as volatile vars

```
void payload() {
  string prog = "/bin/sh\0";
  int64* argv = {&prog, 0x0};

  __r0 = &prog;
  __r1 = &argv;
  __r2 = 0;

  execve(__r0, __r1, __r2);
}
```


SPL payload
Selecting functional blocks
Searching for dispatcher blocks
Stitching BOP gadgets

Selecting Functional Blocks

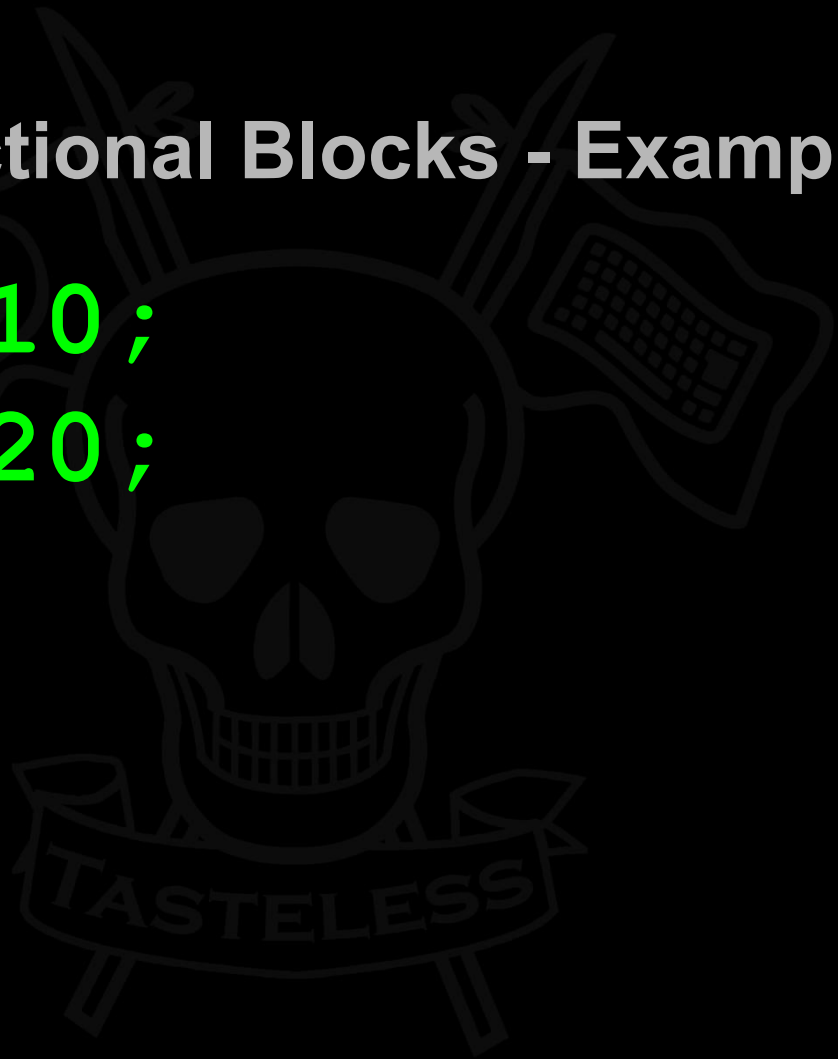
- For each SPL statement: find implementing blocks
  - “Candidate” functional blocks
  - Results in a set of candidates for each SPL statement

- Select all candidate blocks for search process (next step)

- Maximum bipartite matching problem

Selecting Functional Blocks - Example

```
__r0 = 10;
__r1 = 20;
```



Selecting Functional Blocks - Example

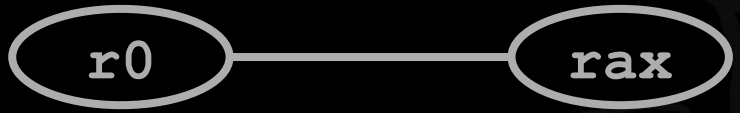
```
__r0 = 10;
__r1 = 20;
```

```
rax = 10
```

Selecting Functional Blocks - Example

```
__r0 = 10;
__r1 = 20;
```

```
rax = 10
```

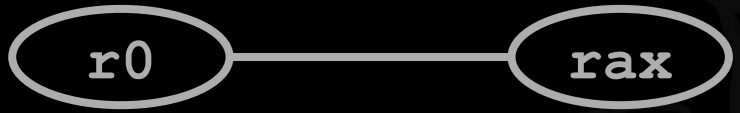


Selecting Functional Blocks - Example

```
__r0 = 10;
__r1 = 20;
```

```
rax = 10
```

```
rdi = 10
```

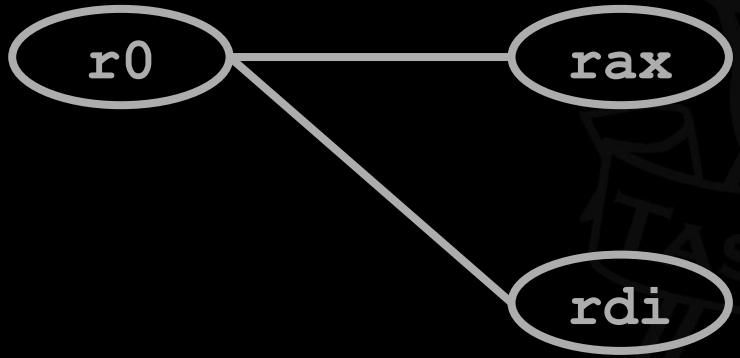


Selecting Functional Blocks - Example

```
__r0 = 10;
__r1 = 20;
```

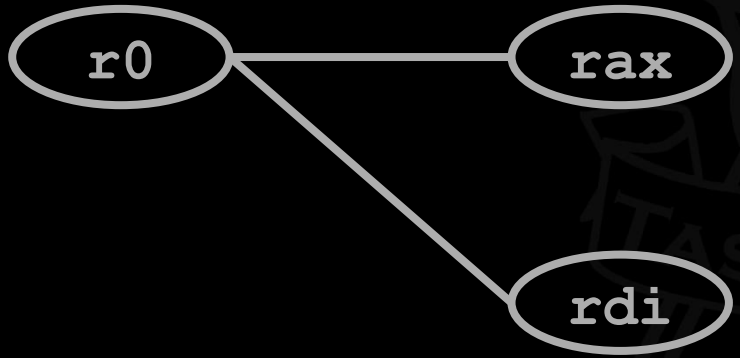
rax = 10

rdi = 10



Selecting Functional Blocks - Example

```
__r0 = 10;
__r1 = 20;
```



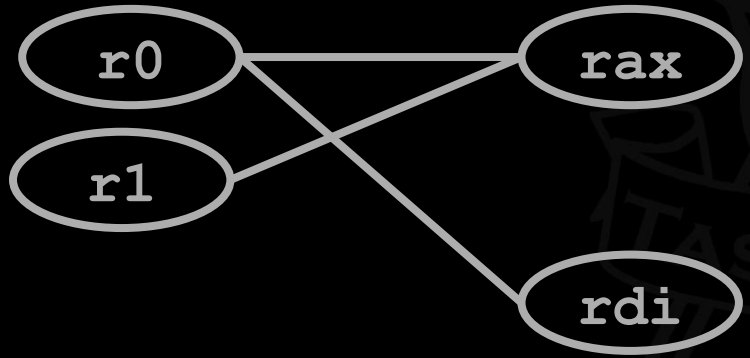

```
rax = 10
```

```
rdi = 10
```

```
rax = 20
```

Selecting Functional Blocks - Example

```
__r0 = 10;
__r1 = 20;
```



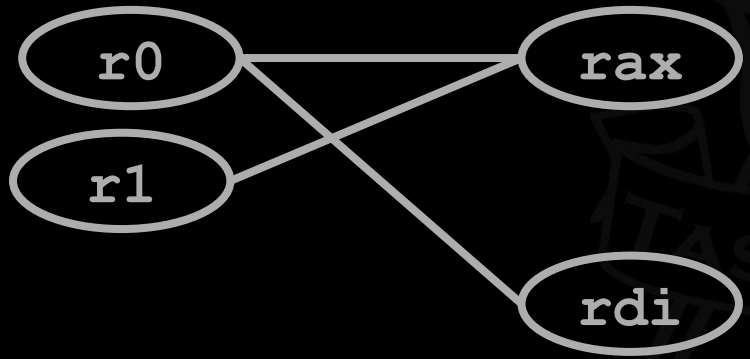

```
rax = 10
```

```
rdi = 10
```

```
rax = 20
```

Selecting Functional Blocks - Example

```
__r0 = 10;
__r1 = 20;
```




```
rax = 10
```

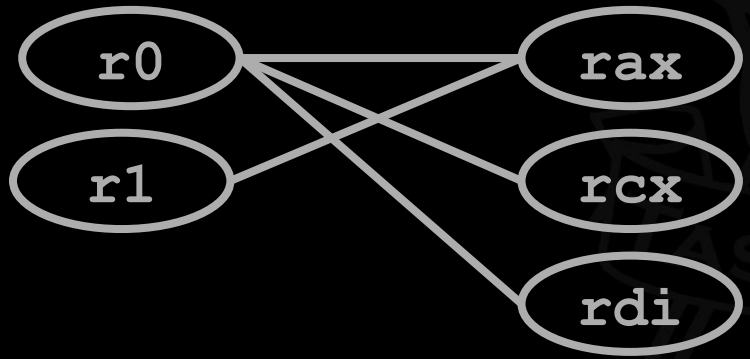
```
rdi = 10
```

```
rax = 20
```

```
rcx = 10
```

Selecting Functional Blocks - Example

```
__r0 = 10;
__r1 = 20;
```




```
rax = 10
```

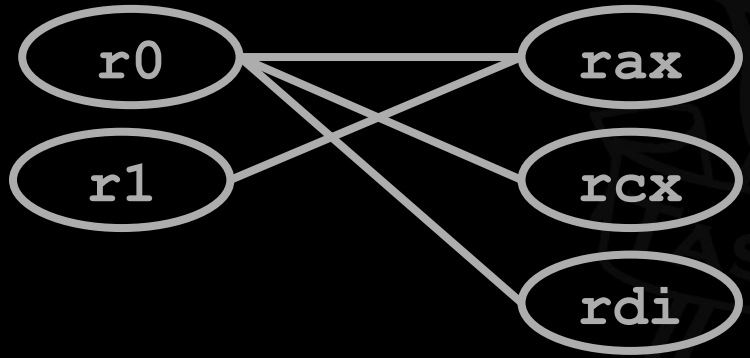
```
rdi = 10
```

```
rax = 20
```

```
rcx = 10
```

Selecting Functional Blocks - Example

```
__r0 = 10;
__r1 = 20;
```




```
rax = 10
```

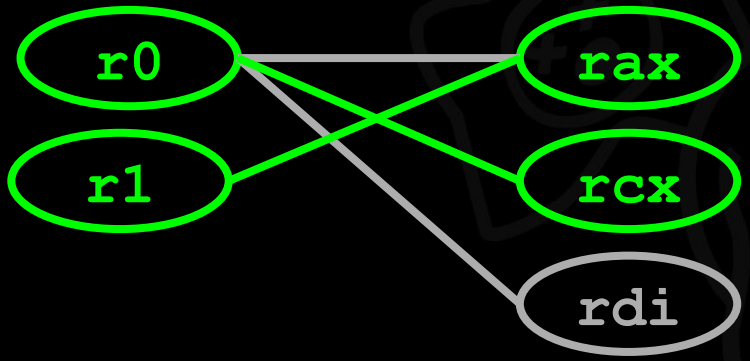
```
rdi = 10
```

```
rax = 20
```

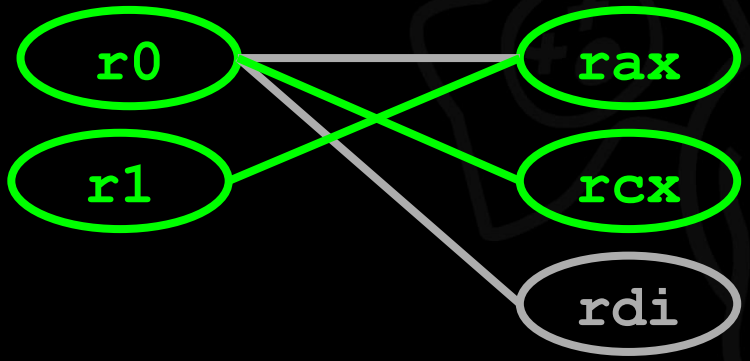
```
rcx = 10
```

```
rcx = 30
```

Selecting Functional Blocks: Example



Selecting Functional Blocks: Example




```
rax = 10
```

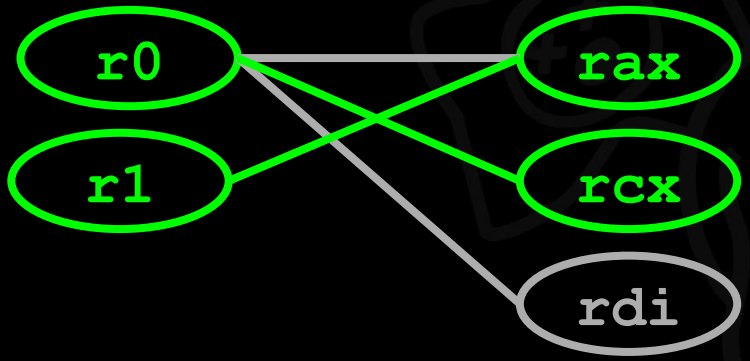
```
rdi = 10
```

```
rax = 20
```

```
rcx = 10
```

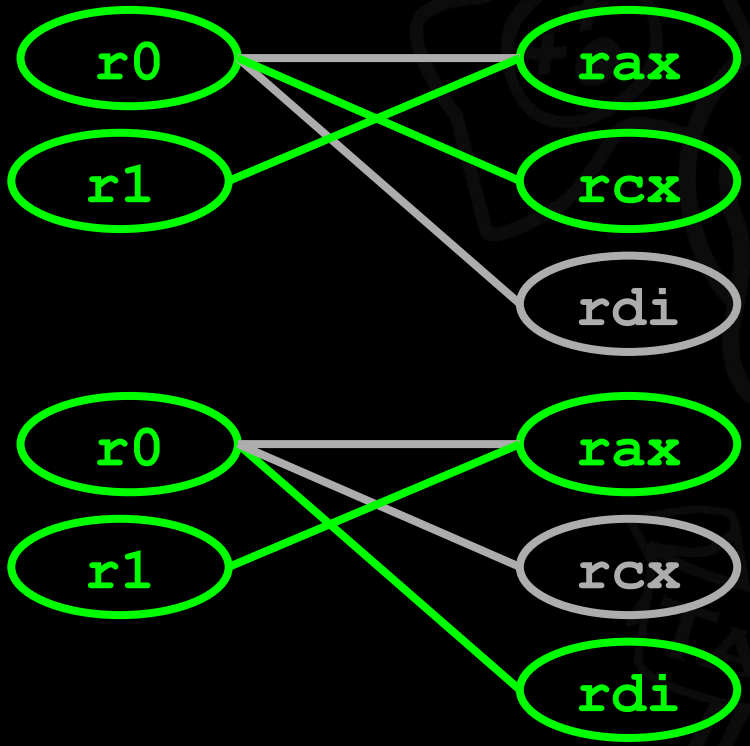
```
rcx = 30
```

Selecting Functional Blocks: Example



`rax = 10`	Clobbering
`rdi = 10`	Dispatcher
`rax = 20`	Functional
`rcx = 10`	Functional
`rcx = 30`	Clobbering

Selecting Functional Blocks: Example




```
rax = 10
```
Clobbering

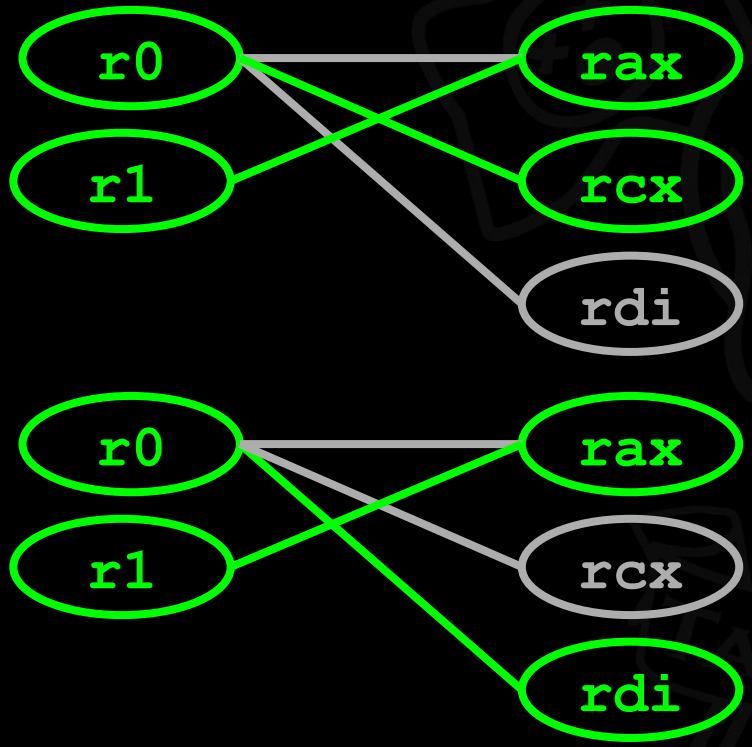
```
rdi = 10
```
Dispatcher

```
rax = 20
```
Functional

```
rcx = 10
```
Functional

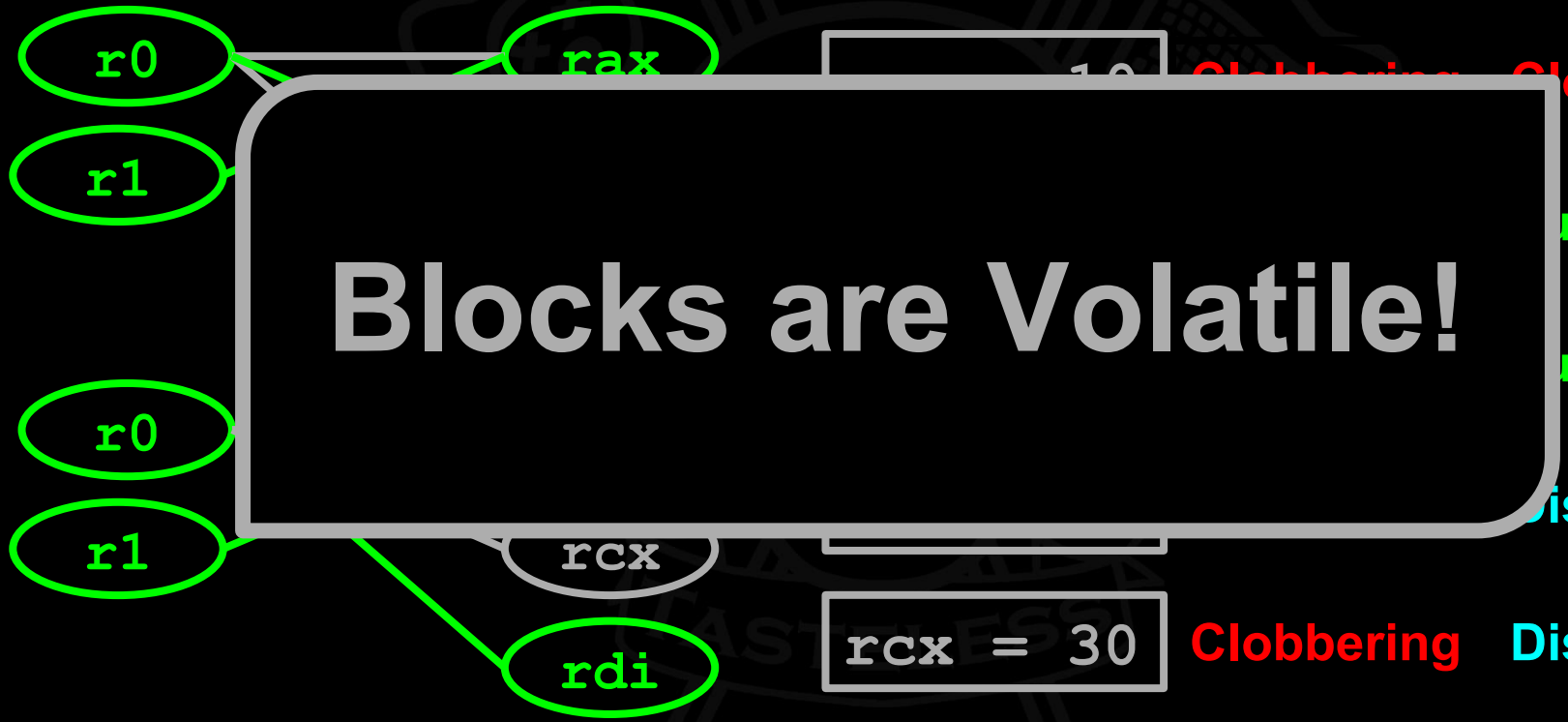
```
rcx = 30
```
Clobbering

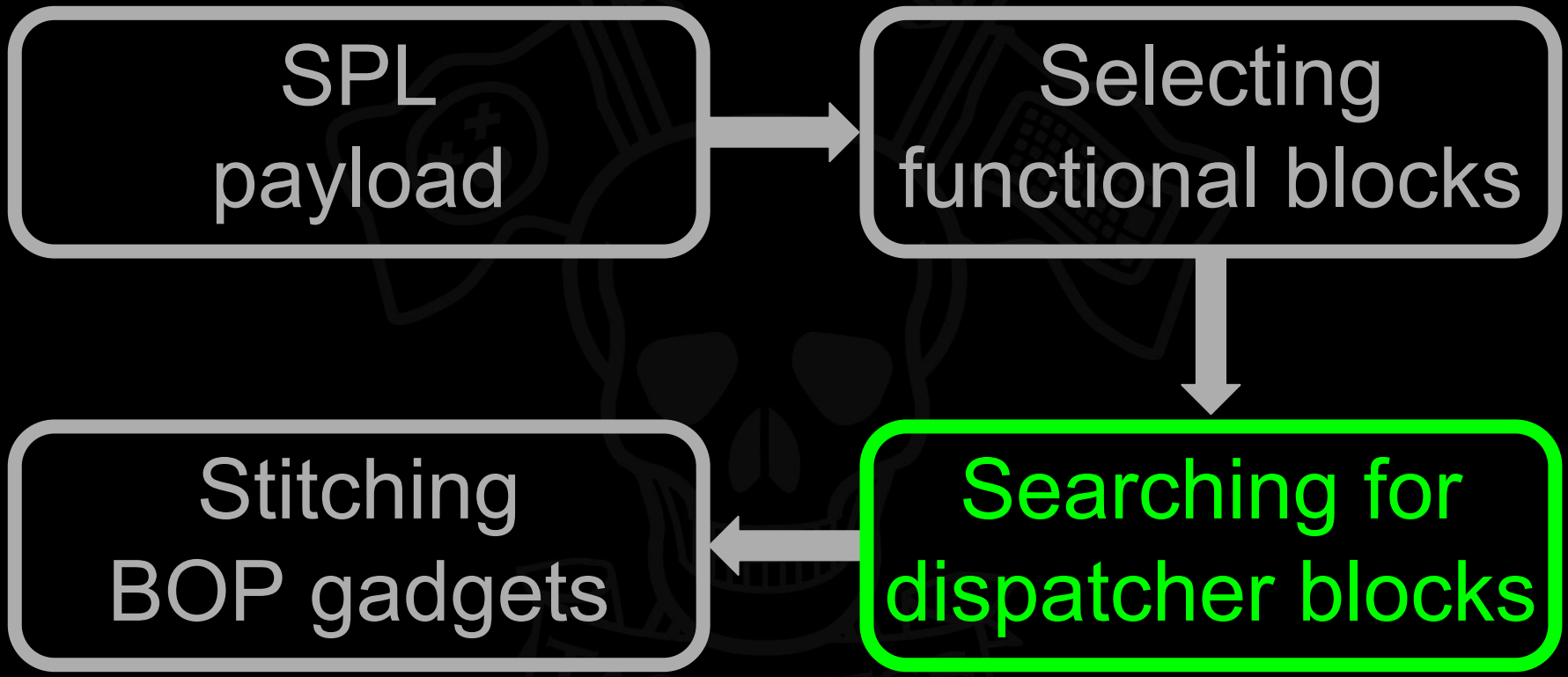
Selecting Functional Blocks: Example



`rax = 10`	Clobbering	Clobbering
`rdi = 10`	Dispatcher	Functional
`rax = 20`	Functional	Functional
`rcx = 10`	Functional	Dispatcher
`rcx = 30`	Clobbering	Dispatcher

Selecting Functional Blocks: Example

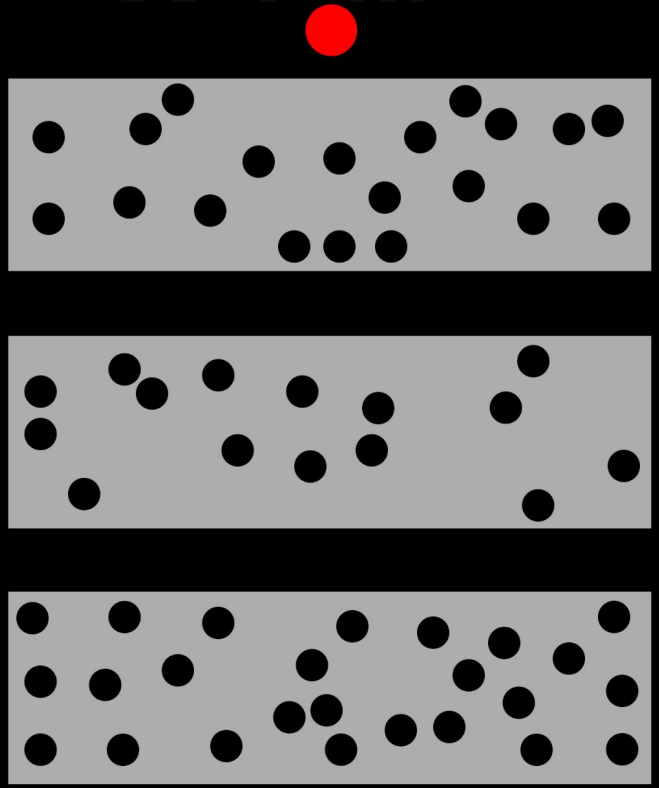



SPL payload
Selecting functional blocks
Searching for dispatcher blocks
Stitching BOP gadgets

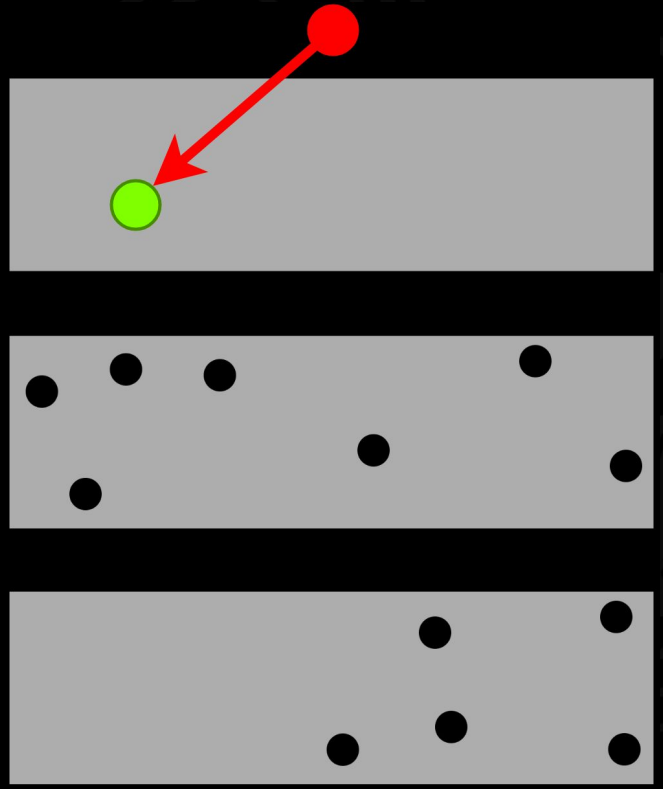
Searching for dispatcher blocks: Concerns

- **BOP gadgets are *volatile***
- **BOP gadgets not arbitrarily chainable**
- **Stitching of BOP Gadgets is NP-hard (unlike ROP Gadgets)**
  - **Problem cannot even be approximated (proven)**
- **Backtracking (iterative process)**

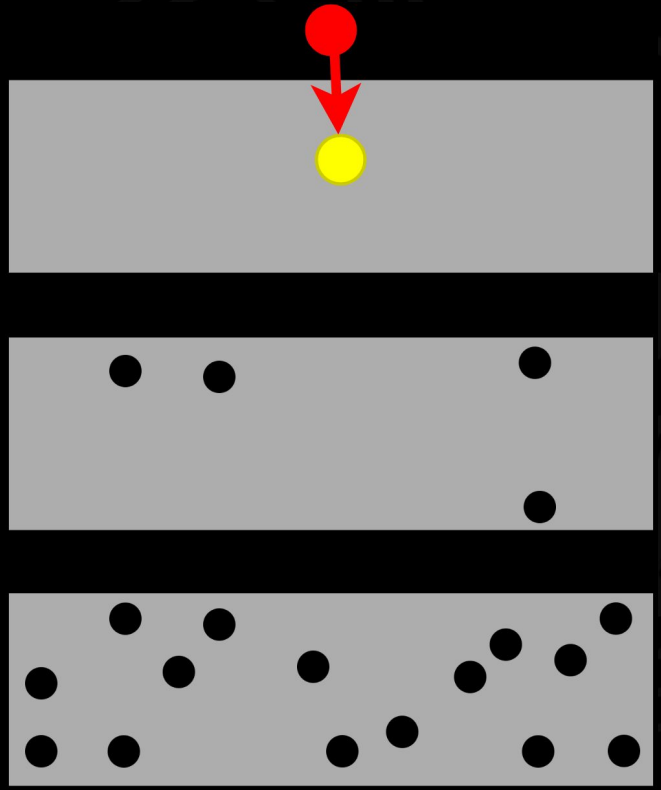
BOP Gadgets: Volatility



BOP Gadgets: Volatility

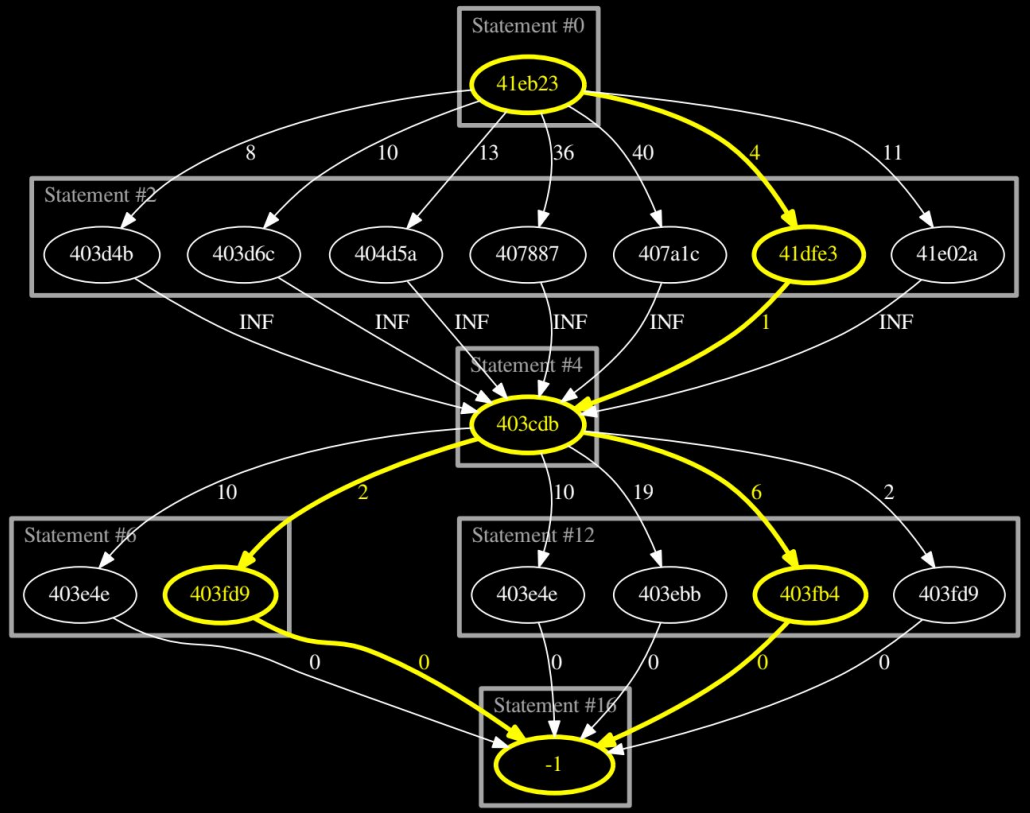


BOP Gadgets: Volatility



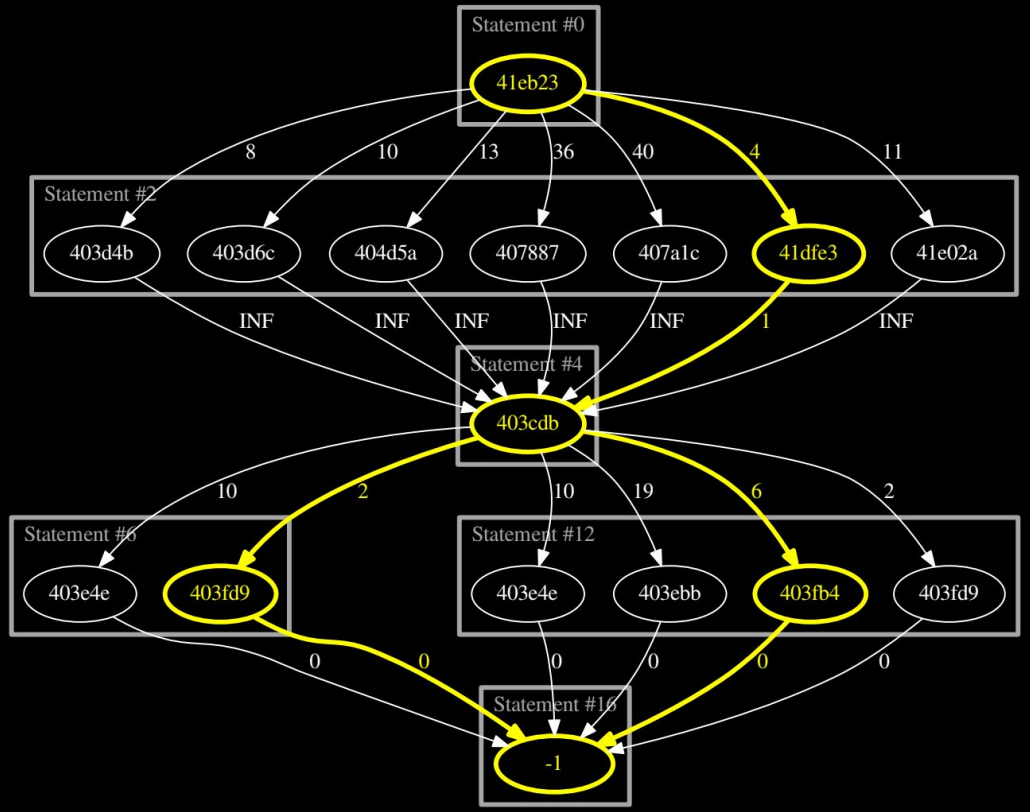
The Delta Graph

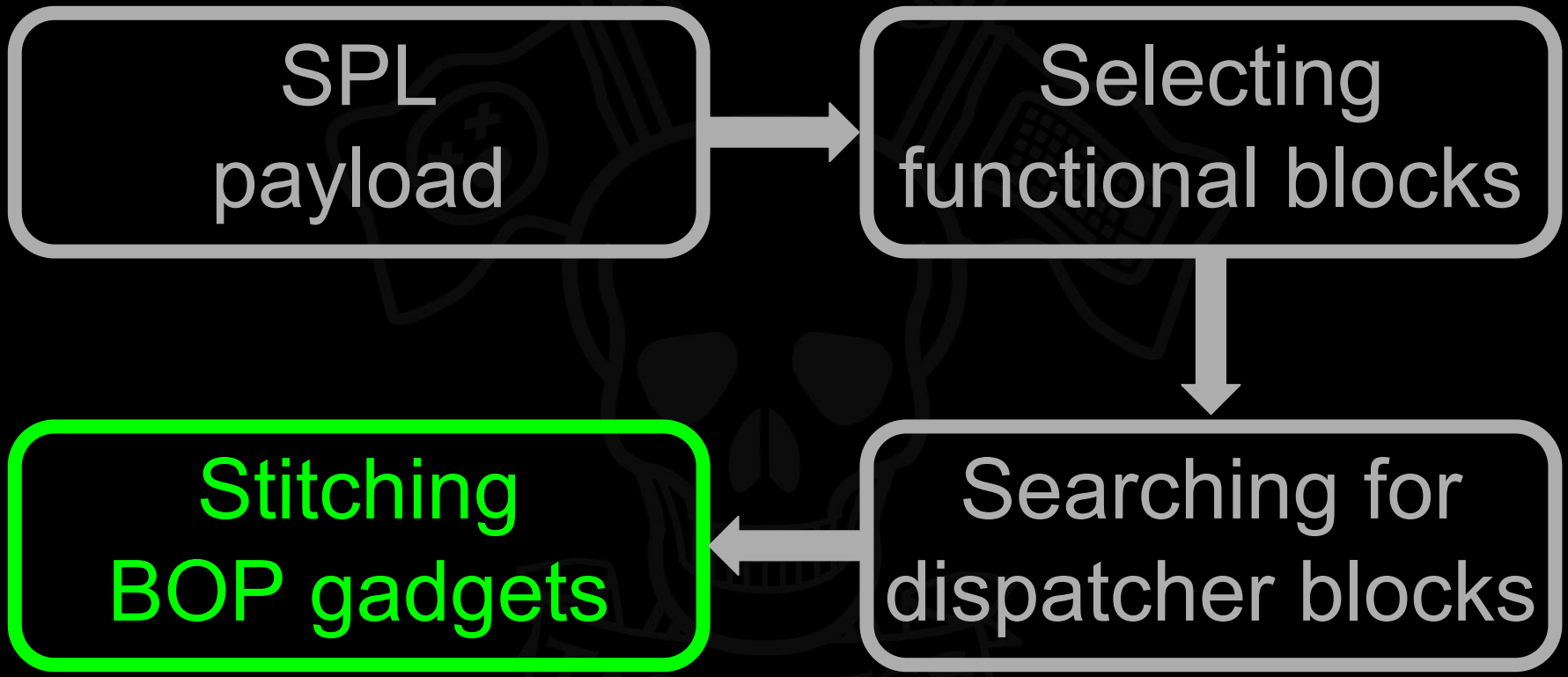
- **Layers: Functional blocks for each SPL statement**
- **Nodes: Functional blocks**
- **Edges: Proximity of functional blocks**
- **Goal: Select *exactly* one "node" from each layer (yellow)**



The Delta Graph

- Rank potential solutions
  - Block proximity as a metric
- Find N *min. induced subgraphs*
  - Candidate execution trace
  - Problem is NP-hard
- Functional blocks to reconstruct attacker's payload
- Context Sensitive Shortest Paths
  - Follow CFG + Shadow Stacks




SPL payload
Selecting functional blocks
Searching for dispatcher blocks
Stitching BOP gadgets

Stitching BOP Gadgets

- Induced subgraph fulfills exploit program
  - All functional blocks can be connected together

- “Verify” that induced subgraph is valid:
  - Many constraints come from multiple dimensions
    - Local constraints
    - SPL constraints
    - Execution constraints

- Verify using *concolic execution + constraint solving*

Stitching BOP Gadgets

- **Utilize concolic execution**
  - **Keep track of <u>every</u> action and extract constraints**

- **Upon failure, try to patch “locally”**
  - **K shortest paths for every edge of the induced subgraph**

- **Once a solution found encode constraints as memory writes**

Evaluation

Proof of Work

Vulnerable Applications & SPL Payloads

Program	Vulnerability	Nodes	RegSet	RegMod	MemRd	MemWr	Call	Cond	Total
ProFTPd	CVE-2006-5815	27,087	40,143	387	1,592	199	77	3,029	45,427
nginx	CVE-2013-2028	24,169	31,497	1,168	1,522	279	35	3375	37,876
sudo	CVE-2012-0809	3,399	5,162	26	157	18	45	307	5715
orzhttpd	BugtraqID 41956	1,345	2,317	9	39	8	11	89	2473
wuftpd	CVE-2000-0573	8,899	14,101	62	274	11	94	921	15,463
nullhttpd	CVE-2002-1496	1,488	2,327	77	54	7	19	125	2,609
opensshd	CVE-2001-0144	6,688	8,800	98	214	19	63	558	9,752
wireshark	CVE-2014-2299	74,186	124,053	639	1,736	193	100	4555	131276
apache	CVE-2006-3747	18,790	33,615	212	490	66	127	1,768	36,278
smbclient	CVE-2009-1886	166,081	265,980	1,481	6,791	951	119	28,705	304,027

RegSet: Register Assignment Gadgets
RegMod: Register Modification Gadgets
MemRd: Memory Read Gadgets
MemWr: Memory Write Gadgets
Call: Function/System Call Gadgets
Cond: Conditional Statement Gadgets
Total: Total number of Functional Gadgets

Vulnerable Applications & SPL Payloads

Program	Vulnerability	Nodes	RegSet	RegMod	MemRd	MemWr	Call	Cond	Total
ProFTPd	CVE-2006-5815	27,087	40,143	387	1,592	199	77	3,029	45,427
nginx	CVE-2013-2028	24,169	31,497	1,168	1,522	279	35	3375	37,876
sudo	CVE-2012-0809	3,399	5,162	26	157	18	45	307	5715
orzhttpd	BugtraqID 41956	1,345	2,317	9	39	8	11	89	2473
wuftpd	CVE-2000-0573	8,899	14,101	62	274	11	94	921	15,463
nullhttpd	CVE-2002-1496	1,488	2,327	77	54	7	19	125	2,609
opensshd	CVE-2001-0144	6,688	8,800	98	214	19	63	558	9,752
wireshark	CVE-2014-2299	74,186	124,053	639	1,736	193	100	4555	131276
apache	CVE-2006-3747	18,790	33,615	212	490	66	127	1,768	36,278
smbclient	CVE-2009-1886	166,081	265,980	1,481	6,791	951	119	28,705	304,027

RegSet: Register Assignment Gadgets
RegMod: Register Modification Gadgets
MemRd: Memory Read Gadgets
MemWr: Memory Write Gadgets
Call: Function/System Call Gadgets
Cond: Conditional Statement Gadgets
Total: Total number of Functional Gadgets

Vulnerable Applications & SPL Payloads

Program	Vulnerability	Nodes	RegSet	RegMod	MemRd	MemWr	Call	Cond	Total
ProFTPd	CVE-2006-5815	27,087	40,143	387	1,592	199	77	3,029	45,427
nginx	CVE-2013-2028	24,169	31,497	1,168	1,522	279	35	3375	37,876
sudo	CVE-2012-0809	3,399	5,162	26	157	18	45	307	5715
orzhttpd	BugtraqID 41956	1,345	2,317	9	39	8	11	89	2473
wuftpd	CVE-2000-0573	8,899	14,101	62	274	11	94	921	15,463
nullhttpd	CVE-2002-1496	1,488	2,327	77	54	7	19	125	2,609
opensshd	CVE-2001-0144	6,688	8,800	98	214	19	63	558	9,752
wireshark	CVE-2014-2299	74,186	124,053	639	1,736	193	100	4555	131276
apache	CVE-2006-3747	18,790	33,615	212	490	66	127	1,768	36,278
smbclient	CVE-2009-1886	166,081	265,980	1,481	6,791	951	119	28,705	304,027

RegSet: Register Assignment Gadgets
RegMod: Register Modification Gadgets
MemRd: Memory Read Gadgets
MemWr: Memory Write Gadgets
Call: Function/System Call Gadgets
Cond: Conditional Statement Gadgets
Total: Total number of Functional Gadgets

Payload	Description
"regset4	Initialize 4 registers with arbitrary values
regref4	Initialize 4 registers with pointers to arbitrary memory
regset5	Initialize 5 registers with arbitrary values
regref5	Initialize 5 registers with pointers to arbitrary memory
regmod	Initialize a register with an arbitrary value and modify it
memrd	Read from arbitrary memory
memwr	Write to arbitrary memory
print	Display a message to stdout using write
execve	Spawn a shell through execve
abloop	Perform an arbitrarily long bounded loop utilizing regmod
infloop	Perform an infinite loop that sets a register in its body
ifelse	An if-else condition based on a register comparison
loop	Conditional loop with register modification

Vulnerable Applications & SPL Payloads

Program	Vulnerability	Nodes	RegSet	RegMod	MemRd	MemWr	Call	Cond	Total
ProFTPd	CVE-2006-5815	27,087	40,143	387	1,592	199	77	3,029	45,427
nginx	CVE-2013-2028	24,169	31,497	1,168	1,522	279	35	3375	37,876
sudo	CVE-2012-0809	3,399	5,162	26	157	18	45	307	5715
orzhttpd	BugtraqID 41956	1,345	2,317	9	39	8	11	89	2473
wuftpd	CVE-2000-0573	8,899	14,101	62	274	11	94	921	15,463
nullhttpd	CVE-2002-1496	1,488	2,327	77	54	7	19	125	2,609
opensshd	CVE-2001-0144	6,688	8,800	98	214	19	63	558	9,752
wireshark	CVE-2014-2299	74,186	124,053	639	1,736	193	100	4555	131276
apache	CVE-2006-3747	18,790	33,615	212	490	66	127	1,768	36,278
smbclient	CVE-2009-1886	166,081	265,980	1,481	6,791	951	119	28,705	304,027

RegSet: Register Assignment Gadgets
RegMod: Register Modification Gadgets
MemRd: Memory Read Gadgets
MemWr: Memory Write Gadgets
Call: Function/System Call Gadgets
Cond: Conditional Statement Gadgets
Total: Total number of Functional Gadgets

Payload	Description
"regset4	Initialize 4 registers with arbitrary values
regref4	Initialize 4 registers with pointers to arbitrary memory
regset5	Initialize 5 registers with arbitrary values
regref5	Initialize 5 registers with pointers to arbitrary memory
regmod	Initialize a register with an arbitrary value and modify it
memrd	Read from arbitrary memory
memwr	Write to arbitrary memory
print	Display a message to stdout using write
execve	Spawn a shell through execve
ablоop	Perform an arbitrarily long bounded loop utilizing regmod
infloop	Perform an infinite loop that sets a register in its body
ifelse	An if-else condition based on a register comparison
loop	Conditional loop with register modification

Results of payload execution

Program	SPL payload												
	regset4	*regref4*	*regset5*	*regref5*	*regmod*	*memrd*	*memwr*	*print*	*execve*	*abloop*	*infloop*	*ifelse*	*loop*
ProFTPd	✓	✓	✓	✓	✓	✓	✓	✓ 32	✗$_1$	✓ 128+	✓ ∞	✓	✓ 3
nginx	✓	✓	✓	✓	✓	✓	✓	✗$_4$	✓	✓ 128+	✓ ∞	✓	✓ 128
sudo	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗$_4$	✓ 128+	✗$_4$	✗$_4$
orzhttpd	✓	✓	✓	✓	✓	✓	✓	✗$_4$	✗$_1$	✗$_4$	✓ 128+	✗$_4$	✗$_3$
wuftdp	✓	✓	✓	✓	✓	✓	✓	✓	✗$_1$	✓ 128+	✓ 128+	✗$_4$	✗$_3$
nullhttpd	✓	✓	✓	✓	✓	✓	✗$_3$	✗$_3$	✓	✓ 30	✓ ∞	✗$_4$	✗$_3$
opensshd	✓	✓	✓	✓	✓	✓	✗$_4$	✗$_4$	✗$_4$	✓ 512	✓ 128+	✓	✓ 99
wireshark	✓	✓	✓	✓	✓	✓	✓	✓ 4	✗$_1$	✓ 128+	✓ 7	✓	✓ 8
apache	✓	✓	✓	✓	✓	✓	✓	✗$_4$	✗$_4$	✓ ∞	✓ 128+	✓	✗$_4$
smbclient	✓	✓	✓	✓	✓	✓	✓	✓ 1	✗$_1$	✓ 1057	✓ 128+	✓	✓ 256

✓ The SPL payload was successfully executed on the target binary
✗$_1$ Not enough candidate blocks
✗$_2$ No valid register/variable mappings
✗$_3$ No valid paths between functional blocks
✗$_4$ Un-satisfiable constraints or solver timeout

Results of payload execution

Program	SPL payload												
	regset4	*regref4*	*regset5*	*regref5*	*regmod*	*memrd*	*memwr*	*print*	*execve*	*abloop*	*infloop*	*ifelse*	*loop*
ProFTPd	✓	✓	✓	✓	✓	✓	✓	✓ 32	✗$_1$	✓ 128+	✓ ∞	✓	✓ 3
nginx	✓	✓	✓	✓	✓	✓	✓	✗$_4$	✓	✓ 128+	✓ ∞	✓	✓ 128
sudo	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗$_4$	✓ 128+	✗$_4$	✗$_4$
orzhttpd	✓	✓	✓	✓	✓	✓	✓	✗$_4$	✗$_1$	✗$_4$	✓ 128+	✗$_4$	✗$_3$
wuftdp	✓	✓	✓	✓	✓	✓	✓	✓	✗$_1$	✓ 128+	✓ 128+	✗$_4$	✗$_3$
nullhttpd	✓	✓	✓	✓	✓	✓	✗$_3$	✗$_3$	✓	✓ 30	✓ ∞	✗$_4$	✗$_3$
opensshd	✓	✓	✓	✓	✓	✓	✗$_4$	✗$_4$	✗$_4$	✓ 512	✓ 128+	✓	✓ 99
wireshark	✓	✓	✓	✓	✓	✓	✓	✓ 4	✗$_1$	✓ 128+	✓ 7	✓	✓ 8
apache	✓	✓	✓	✓	✓	✓	✓	✗$_4$	✗$_4$	✓ ∞	✓ 128+	✓	✗$_4$
smbclient	✓	✓	✓	✓	✓	✓	✓	✓ 1	✗$_1$	✓ 1057	✓ 128+	✓	✓ 256

✓ The SPL payload was successfully executed on the target binary
✗$_1$ Not enough candidate blocks
✗$_2$ No valid register/variable mappings
✗$_3$ No valid paths between functional blocks
✗$_4$ Un-satisfiable constraints or solver timeout

Results of payload execution

Program	SPL payload												
	regset4	*regref4*	*regset5*	*regref5*	*regmod*	*memrd*	*memwr*	*print*	*execve*	*abloop*	*infloop*	*ifelse*	*loop*
ProFTPd	✓	✓	✓	✓	✓	✓	✓	✓ 32	✗$_1$	✓ 128+	✓ ∞	✓	✓ 3
nginx	✓	✓	✓	✓	✓	✓	✓	✗$_4$	✓	✓ 128+	✓ ∞	✓	✓ 128
sudo	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗$_4$	✓ 128+	✗$_4$	✗$_4$
orzhttpd	✓	✓	✓	✓	✓	✓	✓	✗$_4$	✗$_1$	✗$_4$	✓ 128+	✗$_4$	✗$_3$
wuftdp	✓	✓	✓	✓	✓	✓	✓	✓	✗$_1$	✓ 128+	✓ 128+	✗$_4$	✗$_3$
nullhttpd	✓	✓	✓	✓	✓	✓	✗$_3$	✗$_3$	✓	✓ 30	✓ ∞	✗$_4$	✗$_3$
opensshd	✓	✓	✓	✓	✓	✓	✗$_4$	✗$_4$	✗$_4$	✓ 512	✓ 128+	✓	✓ 99
wireshark	✓	✓	✓	✓	✓	✓	✓	✓ 4	✗$_1$	✓ 128+	✓ 7	✓	✓ 8
apache	✓	✓	✓	✓	✓	✓	✓	✗$_4$	✗$_4$	✓ ∞	✓ 128+	✓	✗$_4$
smbclient	✓	✓	✓	✓	✓	✓	✓	✓ 1	✗$_1$	✓ 1057	✓ 128+	✓	✓ 256

- ✓ The SPL payload was successfully executed on the target binary
- ✗$_1$ Not enough candidate blocks
- ✗$_2$ No valid register/variable mappings
- ✗$_3$ No valid paths between functional blocks
- ✗$_4$ Un-satisfiable constraints or solver timeout

Results of payload execution

Program	SPL payload												
	regset4	*regref4*	*regset5*	*regref5*	*regmod*	*memrd*	*memwr*	*print*	*execve*	*abloop*	*infloop*	*ifelse*	*loop*
ProFTPd	✓	✓	✓	✓	✓	✓	✓	✓ 32	✗$_1$	✓ 128+	✓ ∞	✓	✓ 3
nginx	✓	✓	✓	✓	✓	✓	✓	✗$_4$	✓	✓ 128+	✓ ∞	✓	✓ 128
sudo	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗$_4$	✓ 128+	✗$_4$	✗$_4$
orzhttpd	✓	✓	✓	✓	✓	✓	✓	✗$_4$	✗$_1$	✗$_4$	✓ 128+	✗$_4$	✗$_3$
wuftdp	✓	✓	✓	✓	✓	✓	✓	✓	✗$_1$	✓ 128+	✓ 128+	✗$_4$	✗$_3$
nullhttpd	✓	✓	✓	✓	✓	✓	✗$_3$	✗$_3$	✓	✓ 30	✓ ∞	✗$_4$	✗$_3$
opensshd	✓	✓	✓	✓	✓	✓	✗$_4$	✗$_4$	✗$_4$	✓ 512	✓ 128+	✓	✓ 99
wireshark	✓	✓	✓	✓	✓	✓	✓	✓ 4	✗$_1$	✓ 128+	✓ 7	✓	✓ 8
apache	✓	✓	✓	✓	✓	✓	✓	✗$_4$	✗$_4$	✓ ∞	✓ 128+	✓	✗$_4$
smbclient	✓	✓	✓	✓	✓	✓	✓	✓ 1	✗$_1$	✓ 1057	✓ 128+	✓	✓ 256

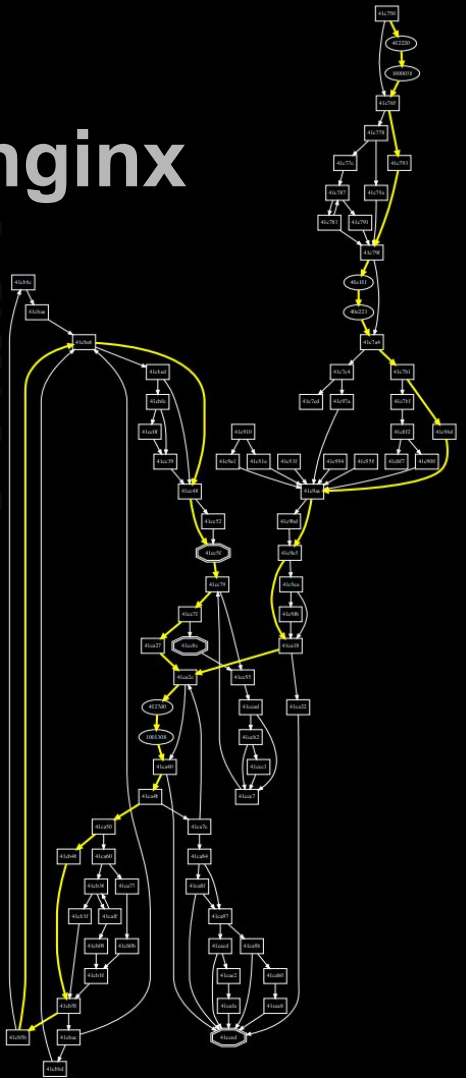
✓ The SPL payload was successfully executed on the target binary
✗$_1$ Not enough candidate blocks
✗$_2$ No valid register/variable mappings
✗$_3$ No valid paths between functional blocks
✗$_4$ Un-satisfiable constraints or solver timeout

Success Rate: *81%*

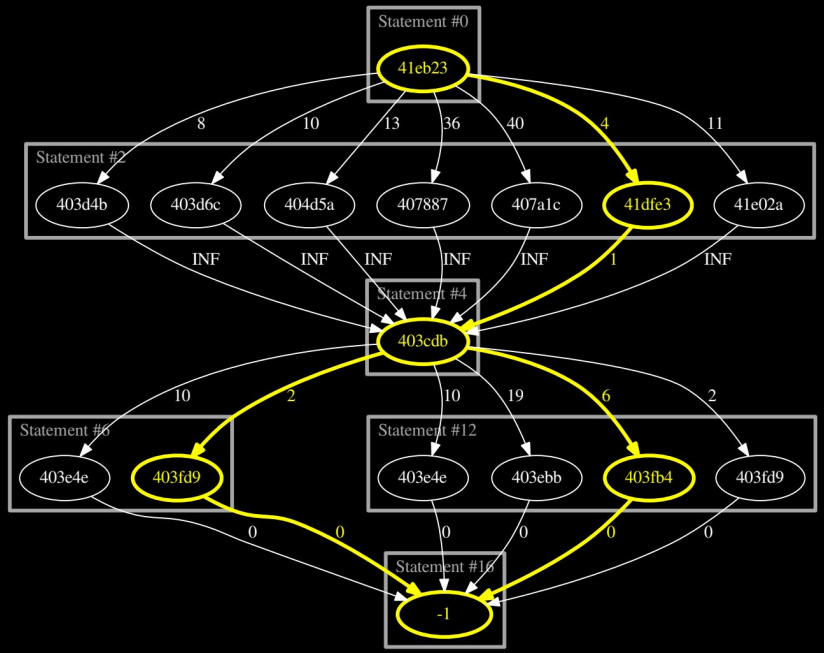
Case Study: infloop payload on nginx

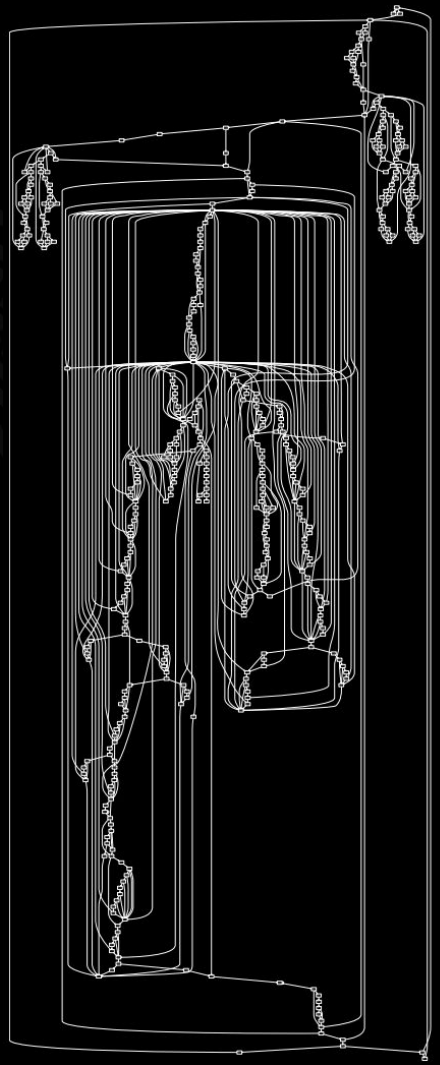
ngx_signal_handler()

```
41C765: signals.signo == 0
40E10F: ngx_time_lock != 0
41C7B1: ngx_process 3 > 1
41C9AC: ngx_cycle = $alloc_1
        $alloc_1 > log = $alloc_2
        $alloc_2 > log_level <= 5
41CA18: signo == 17
41CA4B: waitpid() return value != {0, 1}
41cA50: ngx_last_process == 0
41CB50: *($stack 0x03C) & 0x7F != 0
41CB5B: $alloc_2 > log_level <= 1
41CBE6: *($stack 0x03C + 1) != 2
41CC48: ngx_accept_mutex_ptr == 0
41CC5F: ngx_cycle> shared_memory.part.elts = 0
        __r0 = r14 = 0
41CC79: ngx_cycle > shared_memory.part.nelts <= 0
41CC7F: ngx_cycle > shared_memory.part.next == 0
```



Case Study: ifelse on nginx





Conclusion

- **Automatic CFI-aware Exploitation is feasible**
- **Block Oriented Programming automates Data-Only attacks**
- **Evaluation shows that this can be done in *81%* of the cases**
- **Open Source + VM: https://github.com/HexHive/BOPC**
- **Contact: ispo@purdue.edu**