*


+

SpecROP: Speculative Execution of ROP chains

Atri Bhattacharyya*, Andrés Sánchez*, Esmaeil M. Koruyeh+,
Nael Abu-Ghazaleh+, Chengyu Song+, Mathias Payer*

Speculative Execution Attacks (SEA)

SpecROP is based on SEA

SEA execute *gadgets* speculatively

- Gadgets must already exist in the target
- Existing gadgets are *monolithic*

Monolithic gadget

- accesses secrets, and
- leaks secrets (side-channel dependent)

Misprediction

```
if (x < length)
```

Gadget

```
y = array1[x]
z = array2[y*4096]
```

Spectre-v1

The “power” of SEAs depend on the existing gadgets

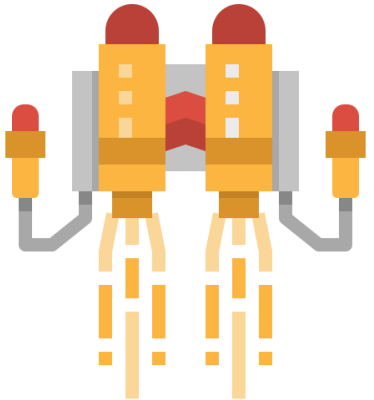
SEA requires powerful monolithic gadgets

Requirements:

1. Accesses the right secret (requires pointer to the secret), or
2. Leaks the right secret (requires the secret in the correct register)
3. Must exist (long, unusual sequences might not exist).

Reality:

- SMoTherSpectre cannot leak AES key from OpenSSL
  - No pointer to key (1)
  - Key not in a register (2)
- No monolithic Spectre gadget in real programs (3)



Jetpack:
- Fuel tank
- Jet engine
- Controls

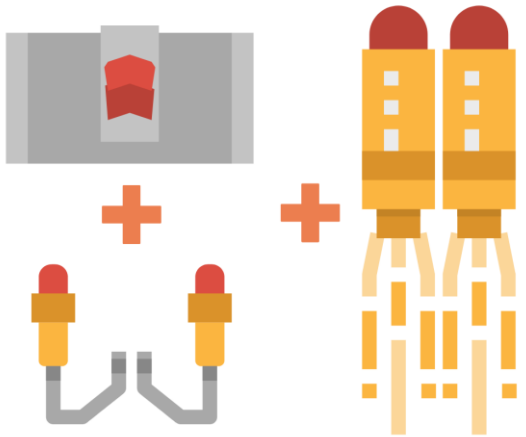
SEA suffer from the lack of appropriate gadgets

Divide and conquer

SpecROP is inspired by ROP attacks

Return Oriented Programming (ROP) attacks

- Non-speculative, code reuse attacks
- Sequence of small gadgets (more likely to exist)
- End in `ret` (can be chained to next gadget)

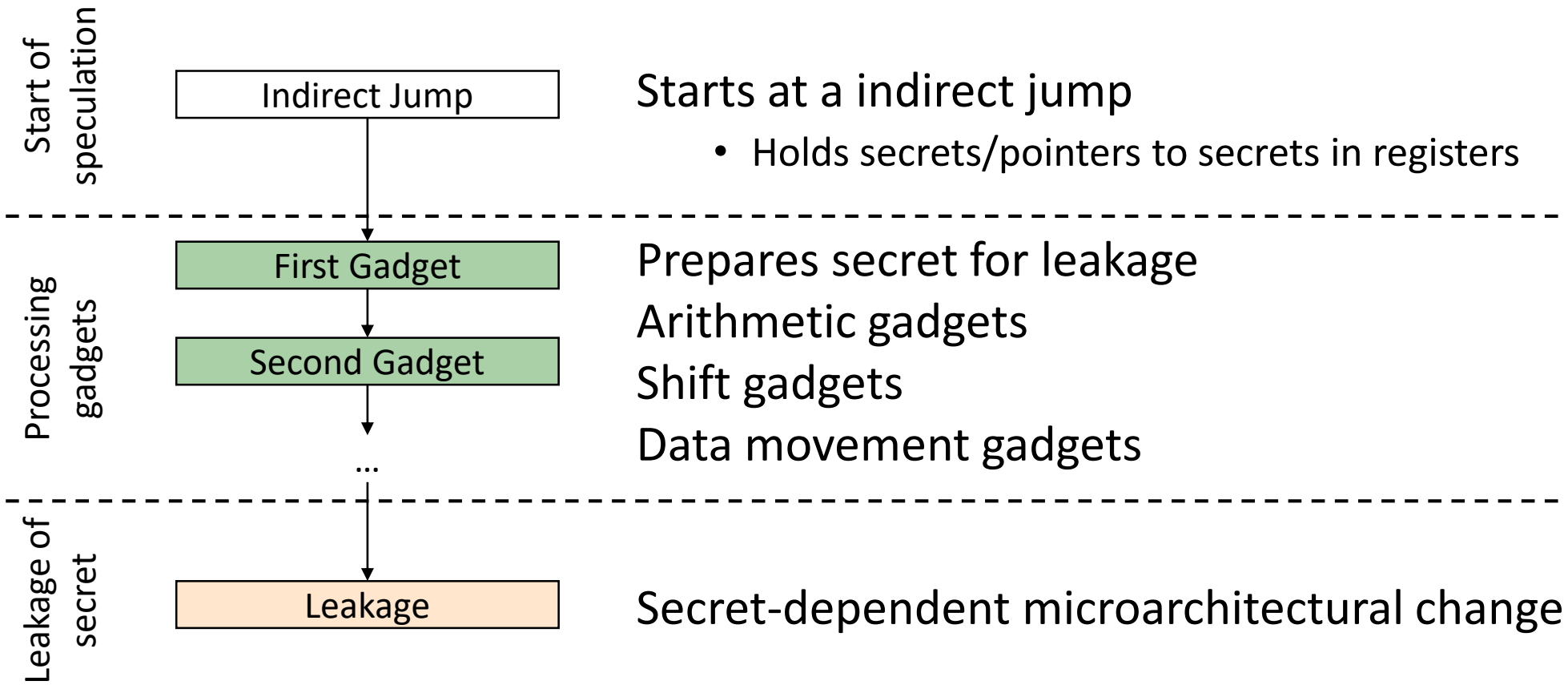


SpecROP principles

- Use *simpler gadgets* ending in `jmp`/`ret`
- *Train branch predictor* to chain gadgets
- Use intermediate gadgets to *modify state*
  (increment pointers, left/right shift registers, move data between registers)

SpecROP enables SEA and makes it more powerful

SpecROP principle



SpecROP links small, simple gadgets into a powerful gadget chain

Spectre v1 with chains

The Spectre v1 gadget reads a secret, then makes a secret-dependent load

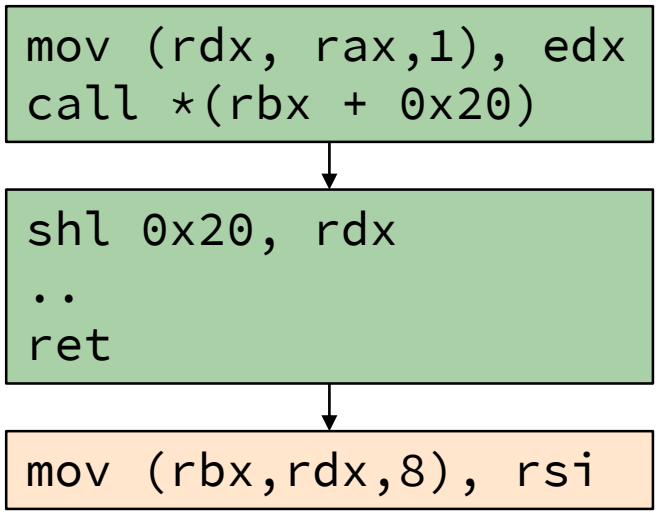
```
C: y = array1[x]
   z = array2[y*4096]
```

Monolithic gadget not found in the wild

```
mov (rax,rdi,8), rax
shl 0xc, rax
mov (rdx,rax,8), rax
```

Assumptions:

```
rax = array1
rdx = array2
rdi = x
```



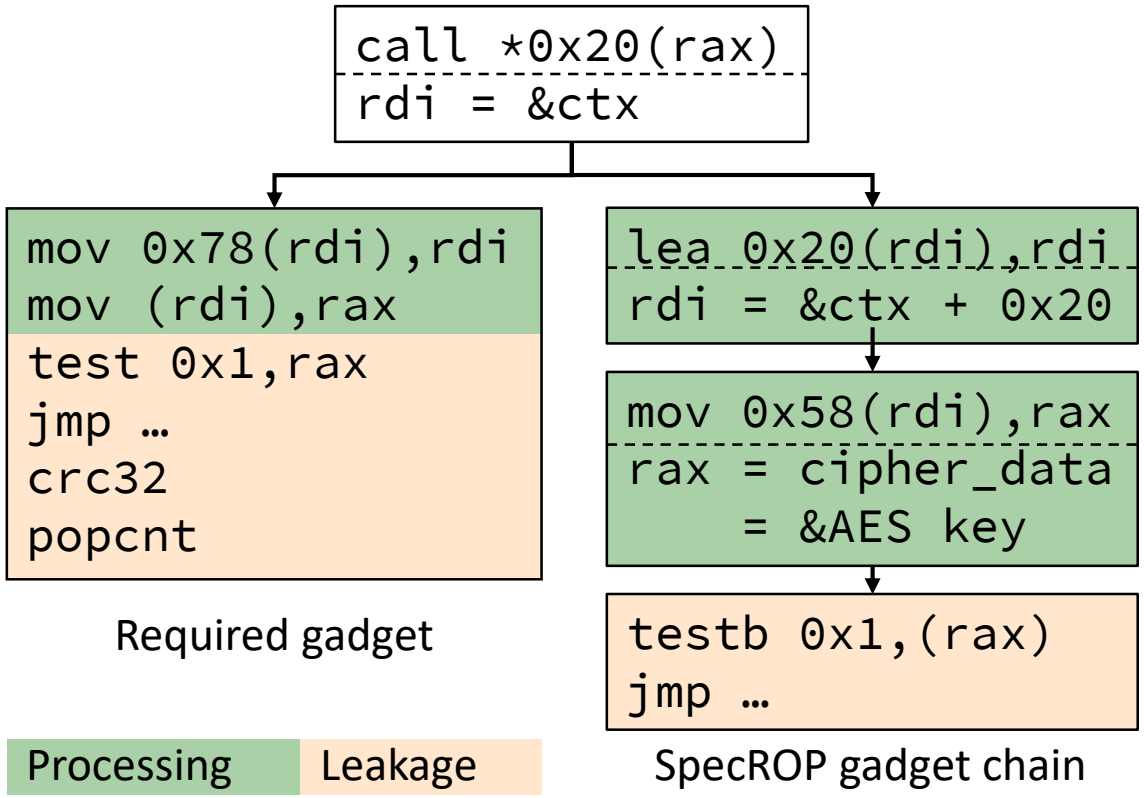
Assumptions:

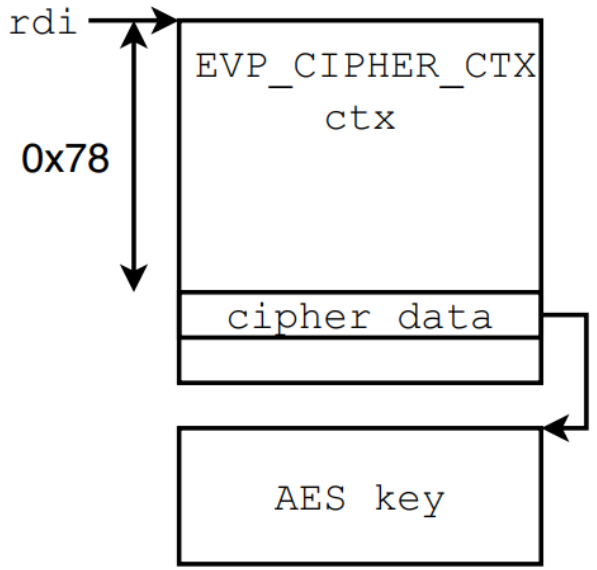
```
rdx = array1
rbx = array2
rax = x
```

SpecROP chain for Spectre v1 exists!

OpenSSL key leakage

(De)Encryption calls `do_cipher(ctx, …)` using indirect call





SpecROP chains are expressive

Evaluation (1/3): Attacker models

Tried 3 attacker models:

- Cross process between SMT threads
- Cross thread between SMT threads
- Single thread, aliased instructions

4 “generations” of Intel processors

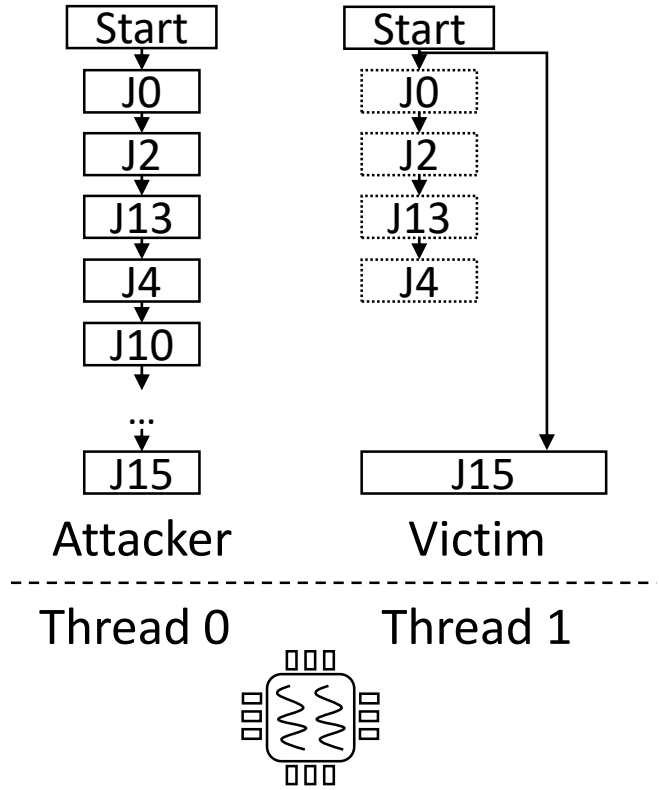
	i7-6700K	i7-8700	i7-9700	i7-10510U
Cross process	N	N	N	N
Cross thread	Y	Y	N	N
Aliased	Y	Y	Y	Y

Aliased attacks are a practical threat

Evaluation (2/3): Length of gadget chain

Setup:

- Attacker trains BTB
  - Goes through gadgets J_0 to J_{15}
  - Each gadget ends with indirect jump
- Victim
  - (architecturally) jumps to J_{15}
  - (speculatively) uses predictions from BTB, executes *some* gadgets
- We track executed gadgets, each loading a unique address



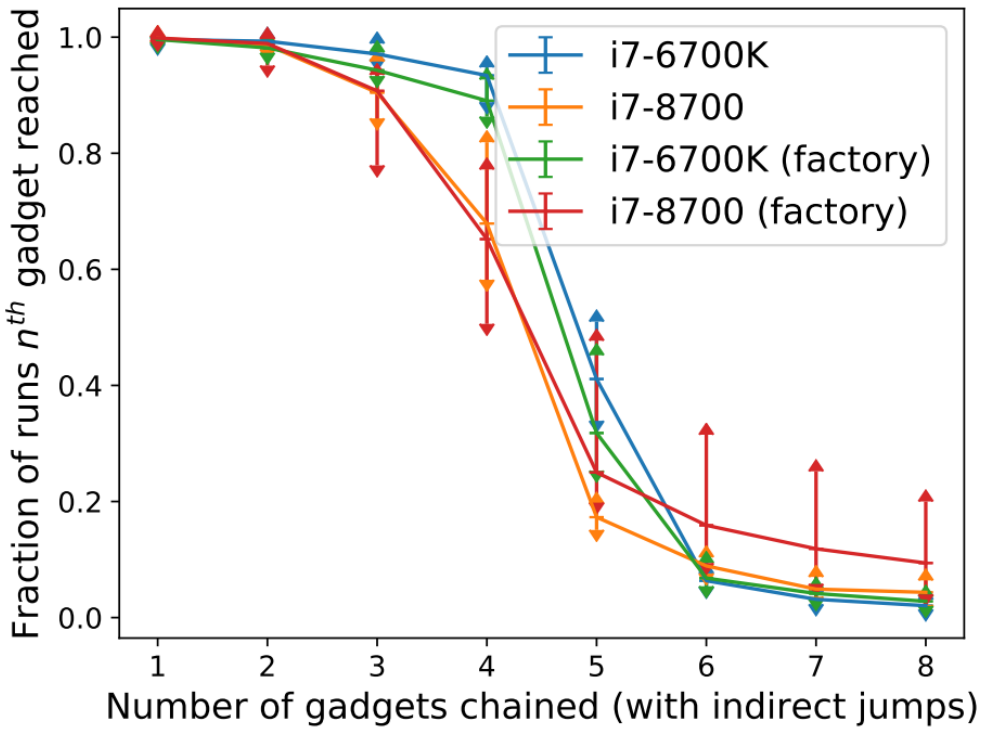
Evaluation (2/3): Length of gadget chain

Testbed:

- Tested on i7-6700K and i7-8700
- With and without microcode updates

Results:

- Up-to 4 gadgets can reasonably be chained
- Microcode does not affect success rate



Practical attacks may use up-to 4 gadgets

Evaluation (3/3): Characterization of gadgets

We created a gadget-search tool: SpecFication

SpecFication phases:

- Disassembly: Get a list of potential processing gadgets
- Characterisation: Express gadgets semantics
- Solving: Express wanted gadget as constraints, check constraints

Library	Binary size	Gadgets
libcrypto	3.3M	13k
libc	1.8M	15k
libdl	15K	266
mod_ssl	235K	490
mod_proxy	131K	338
mod_http2	244K	1,113

Evaluation (3/3): Characterization of gadgets

Large skew in availability of arithmetic gadgets for different registers

Library	rax	rbx	rcx	rdx	rdi	r11
libcrypto	665	259	34	78	69	0
libc	889	317	128	171	419	0
libdl	25	6	0	0	0	0
mod_ssl	12	8	0	4	0	0
mod_proxy	12	6	0	0	2	0
mod_http2	46	5	0	5	0	0

Plentiful data-movement gadgets between pairs of <source, destination> registers (max 240)

Library	**Register-pairs**	**Chained**
libcrypto	116	210
libc	101	204

Arithmetic + data movement gadgets allow expressive computation

Limitations

Lower signal-to-noise ratio

- Leakage gadget reached less often

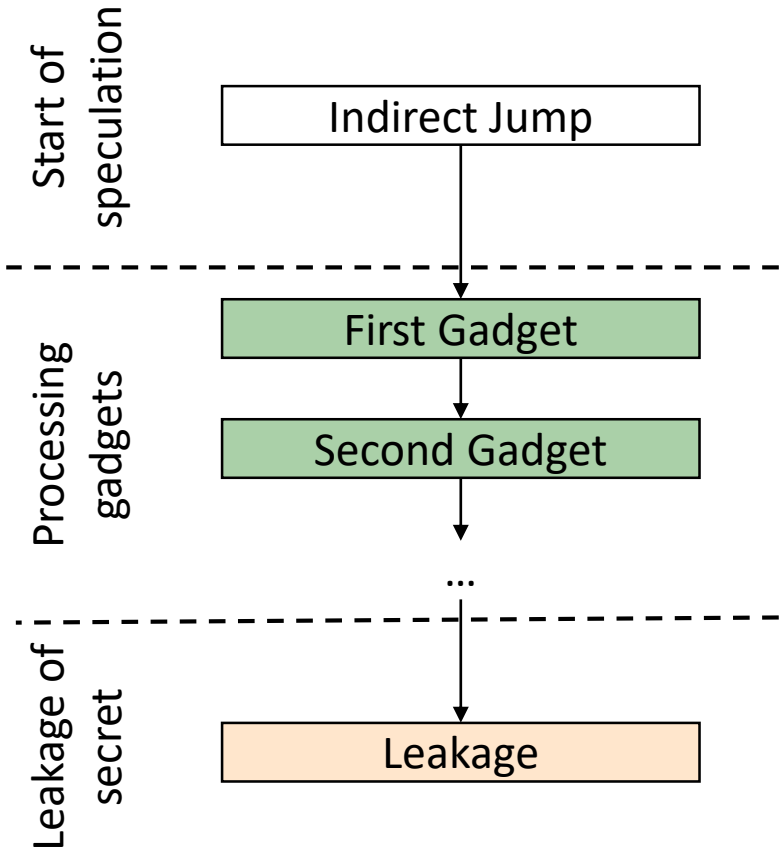
Processing gadgets:

- Are limited by speculation window
- Cannot reuse gadgets ending in indirect jump
- Cannot write to the jump target

Advantage:

- Can fault without ending speculation

For discussion of `ret`, see paper



SpecROP introduces noise from gadget chaining

Mitigations

Prevent branch misprediction:

- SW only: retpolines
- SW/HW: IBRS/IBPB
- HW only: Intel CET and other CFI (control-flow integrity) measures

Finding potential chains through static analysis:

- State explosion, potentially incomplete
- Side-channel specific

Practically:

- Find vulnerable branches (with sensitive information/pointers) statically
- Protect with retpolines

Comprehensive mitigation for SpecROP require speculative CFI in hardware

Conclusions

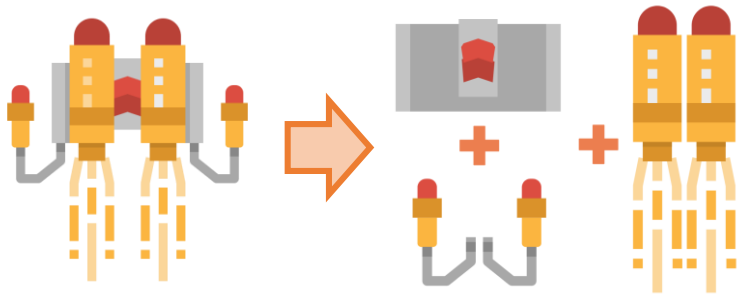
SpecROP breaks monolithic gadget into several, simple gadgets

- Gadgets chained by training branch predictor
- Enables certain attacks previously impossible (e.g. Spectre-v1)
- Extend leakage of other attacks (e.g. SMoTherSpectre)

Practicality of SpecROP is limited

- Branch poisoning much harder today
- The attack surface still remains
- Proper hardware CFI is needed

Gadget search using symbolic analysis of binaries is effective



Code available at https://github.com/HexHive/specrop
Questions in Q&A (or atri[dot]bhattacharyya[at]epfl[dot]ch)

Evaluation

Q1: Which attacker models allow SpecROP?

Q2: How many gadgets can I chain?

Q3: How many processing gadgets exist in real binaries?

Overview

- Introduction
- SpecROP attack principle
  - Spectre v1 case-study
  - OpenSSL case-study
- Evaluation
- Limitations
- Mitigations
- Conclusion