The Dark Age of Memory Corruption Mitigations in the Spectre Era

Andrea Mambretti, IBM Research Europe - Zurich
Alexandra Sandulescu, Google (formerly: IBM Research Europe - Zurich)

About us

Andrea Mambretti

System Security Research @ IBM Research Europe - Zurich

PhD in Cybersecurity @ Northeastern University, Boston

@m4mbr3 - https://mbr.sh

Alexandra Sandulescu

Security @ Google

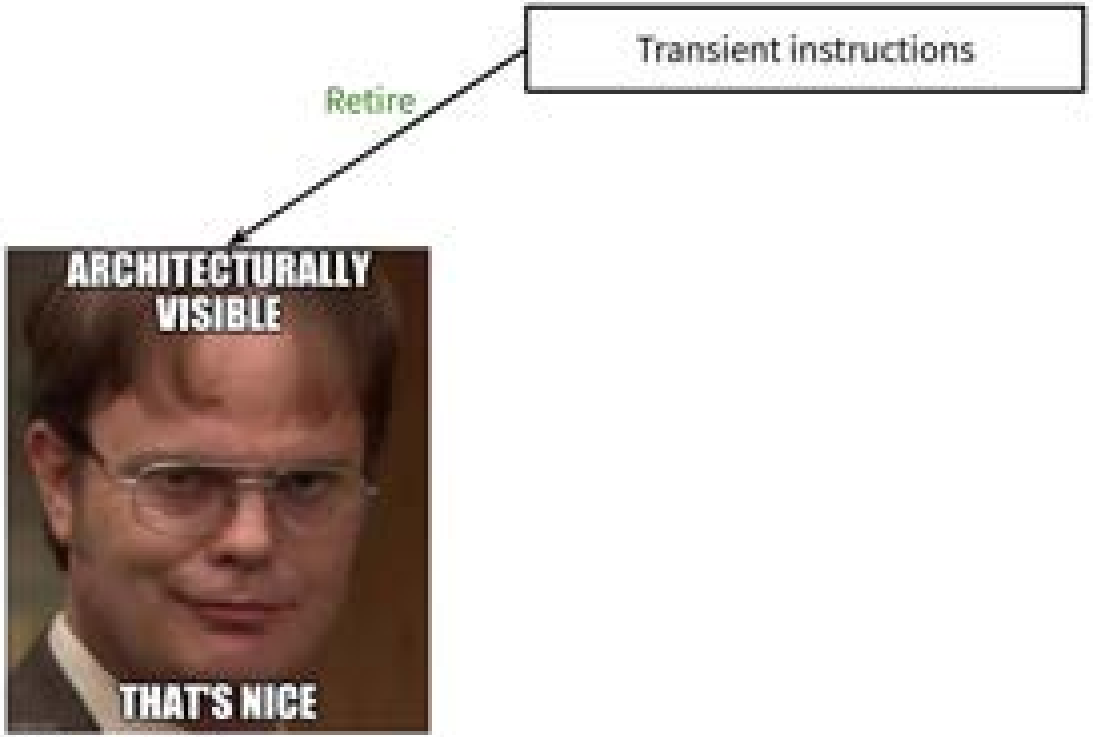
(work done previously at IBM Research Europe - Zurich)

Background - transient execution

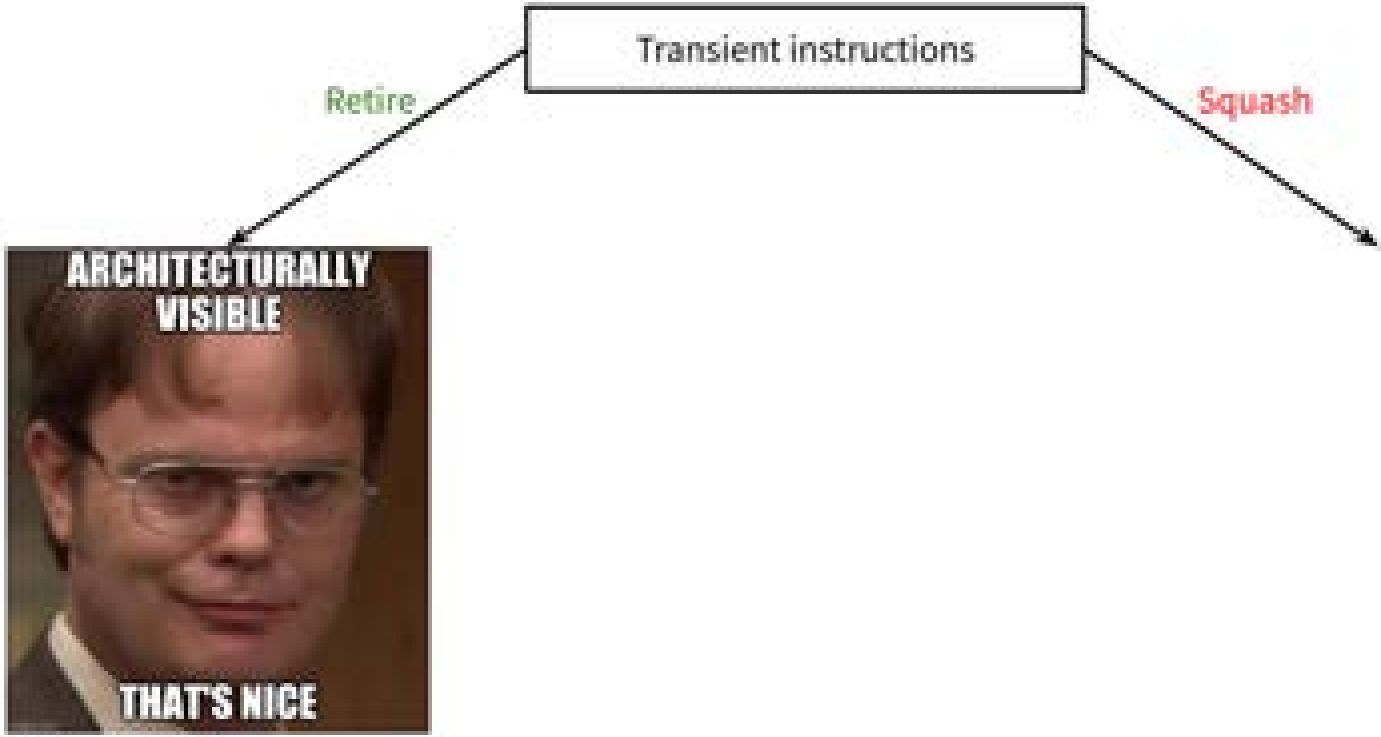
Background - transient execution

Background - transient execution

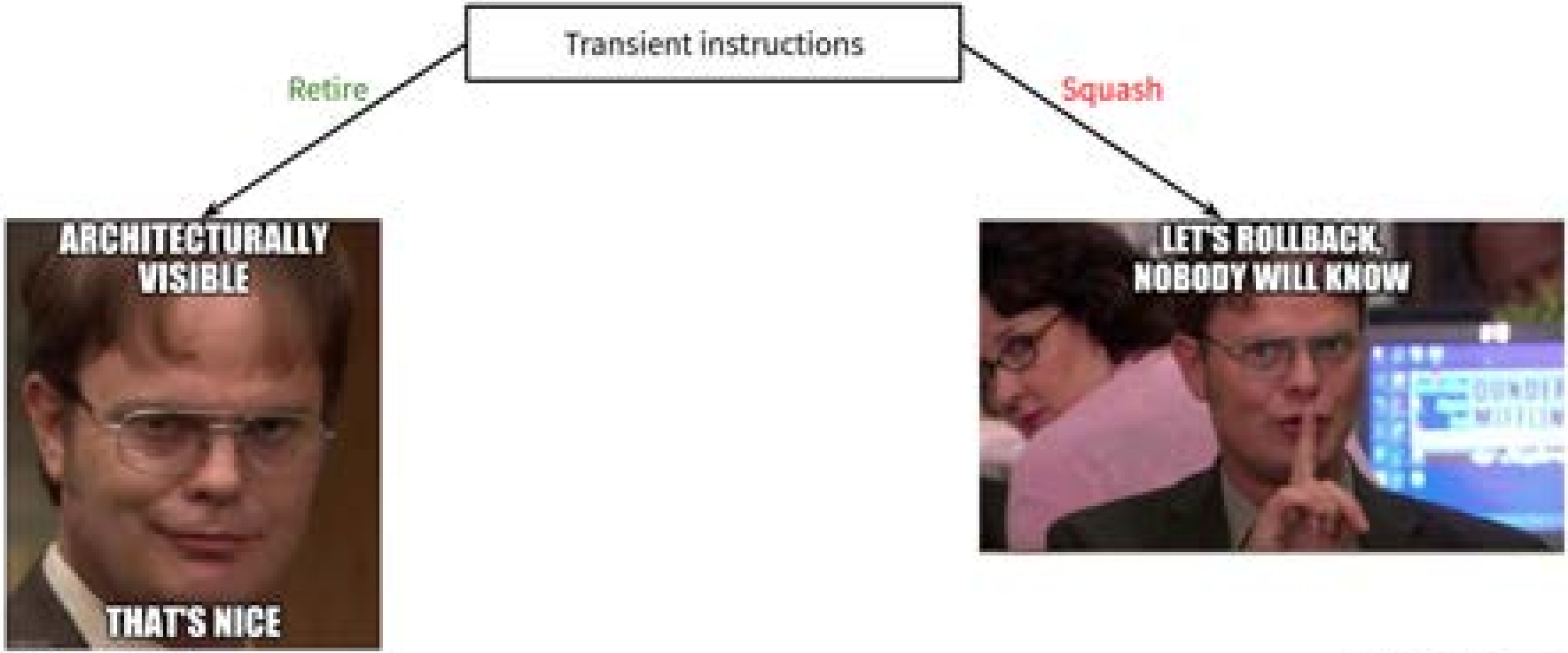
Background - transient execution



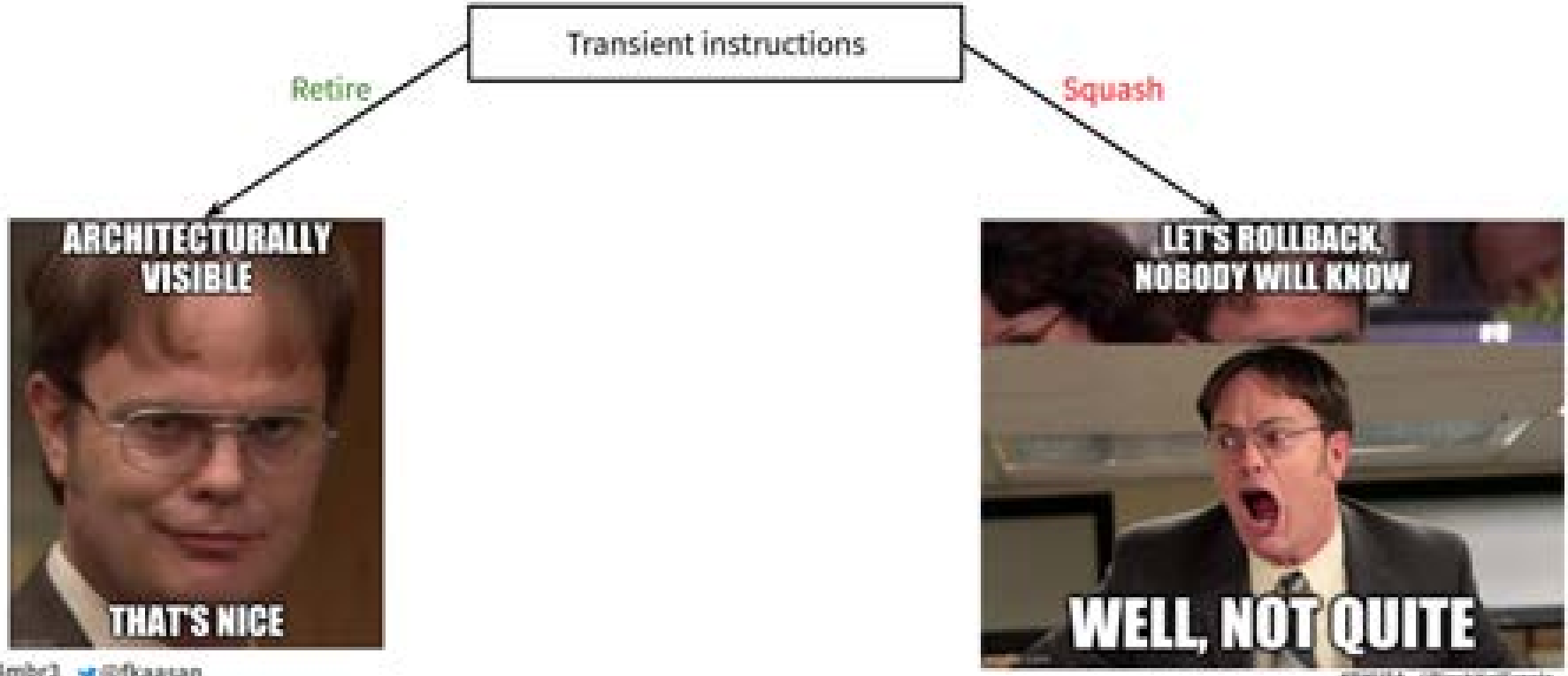
Background - transient execution



Background - transient execution



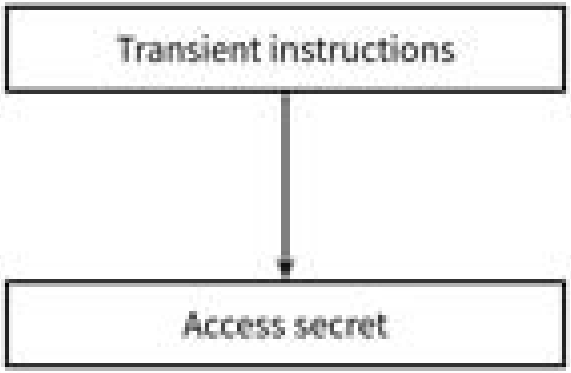
Background - transient execution



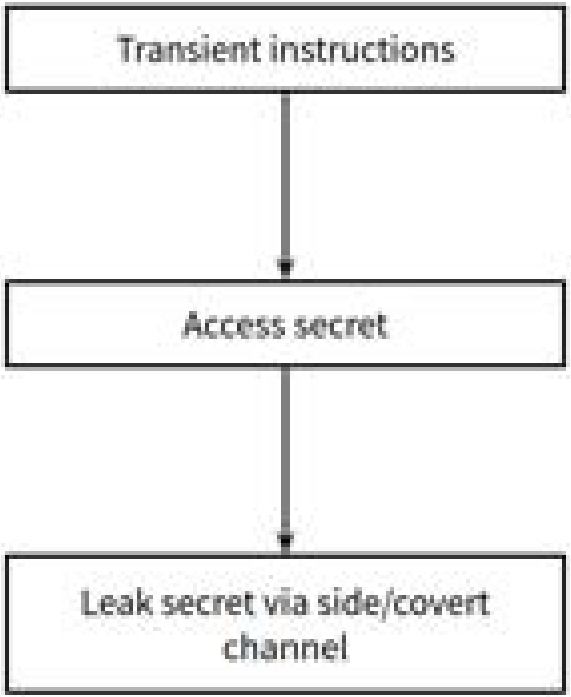
Background - transient execution attacks

Transient instructions

Background - transient execution attacks



Background - transient execution attacks



Threat model for Speculative Execution Attacks (SEAs)

Threat model for Speculative Execution Attacks (SEAs)

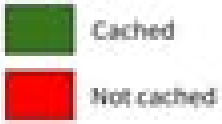
Threat model for Speculative Execution Attacks (SEAs)

Spectre v1.1 - speculative buffer overflow [KW18]

Spectre v1.1 - speculative buffer overflow [KW18]

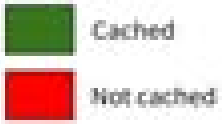

```
/* with x out of bound attacker controlled */
int array1[size_array1];
...
if (x < size_array1) {
   array1[x] = val;
}
return;
```

Spectre v1.1 - speculative buffer overflow [KW18]




```
/* with x out of bound attacker controlled */
int array1[size_array1];
...
if (x < size_array1) {
   array1[x] = val;
}
return;
```

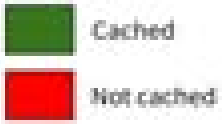
Spectre v1.1 - speculative buffer overflow [KW18]




```
/* with x out of bound attacker controlled */
int array1[size_array1];
...
if (x < size_array1) {
   array1[x] = val;
}
return;
```

Speculative Execution Trigger

Spectre v1.1 - speculative buffer overflow [KW18]

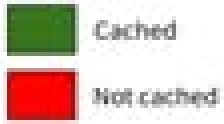



```
/* with x out of bound attacker controlled */
int array1[size_array1];
...
if (x < size_array1) {
  array1[x] = val;
}
return;
```

Speculative overflow if x out of bound

saved RET
saved RBP
....
array1[size_array1 - 1]
array1[...]
....
array1[1]
array1[0]

Spectre v1.1 - speculative buffer overflow [KW18]

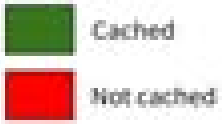



```
/* with x out of bound attacker controlled */
int array1[size_array1];
...
if (x < size_array1) {
  array1[x] = val;
}
return;
```

Speculative overflow if x out of bound

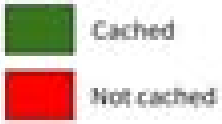
val
saved RBP
....
array1[size_array1 - 1]
array1[...]
....
array1[1]
array1[0]

Spectre v1.1 - speculative buffer overflow [KW18]




```
/* with x out of bound attacker controlled */
int array1[size_array1];
...
if (x < size_array1) {
   array1[x] = val;
}
return;
```

Spectre v1.1 - speculative buffer overflow [KW18]

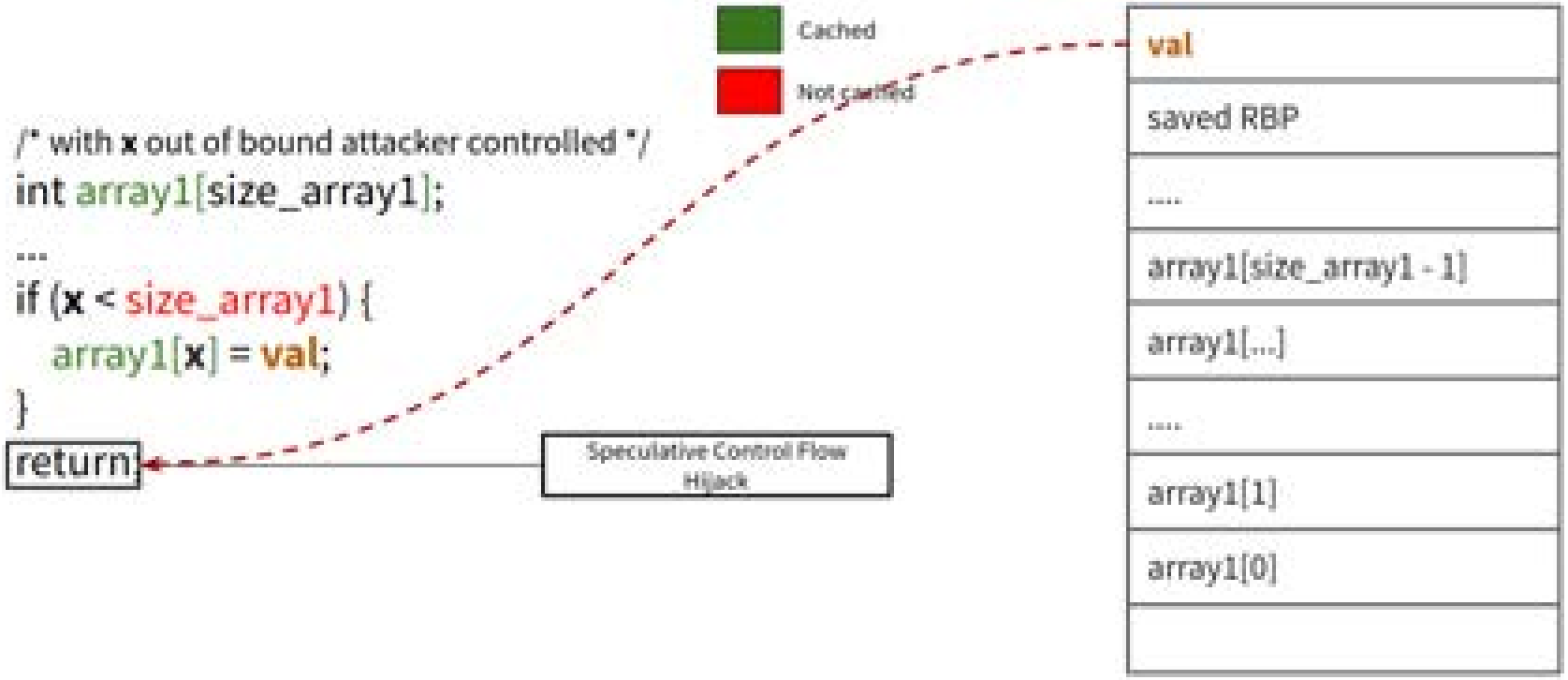



```
/* with x out of bound attacker controlled */
int array1[size_array1];
...
if (x < size_array1) {
   array1[x] = val;
}
return;
```

Spectre v1.1 - speculative buffer overflow [KW18]

```
/* with x out of bound attacker controlled */
int array1[size_array1];
...
if (x < size_array1) {
   array1[x] = val;
}
return;
```

Spectre v1.1 - speculative buffer overflow [KW18]

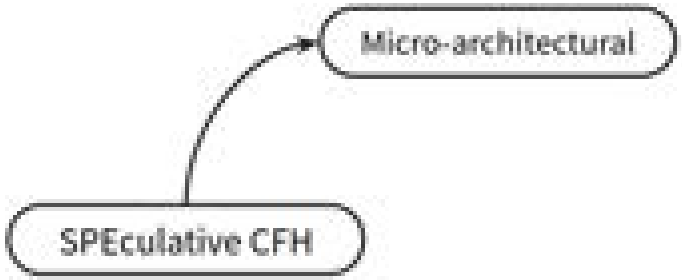


SPEculative ARchitectural control flow hijacks (SPEAR)

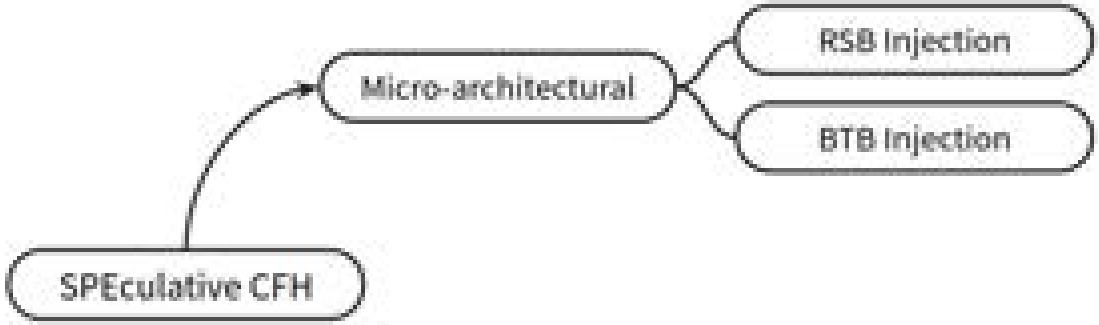
SPEculative ARchitectural control flow hijacks (SPEAR)

SPEculative CFH

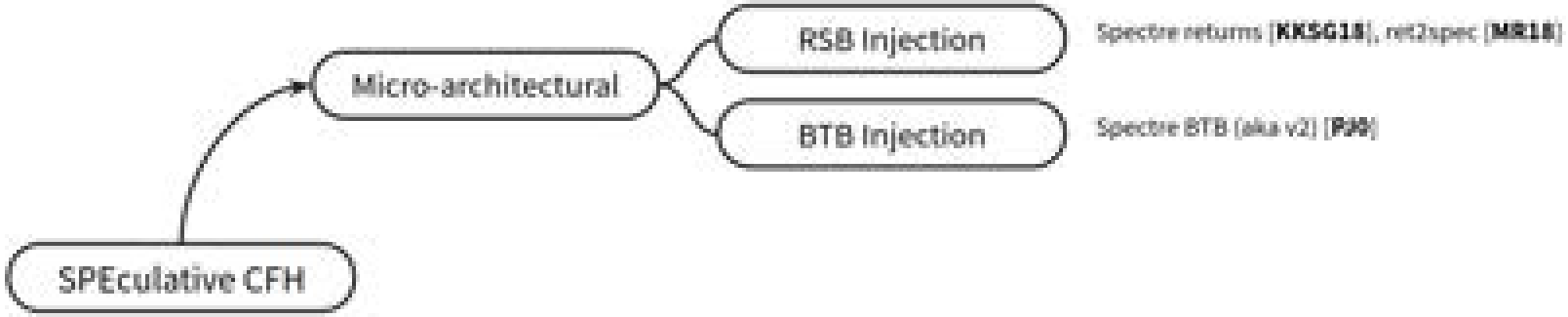
SPEculative ARchitectural control flow hijacks (SPEAR)



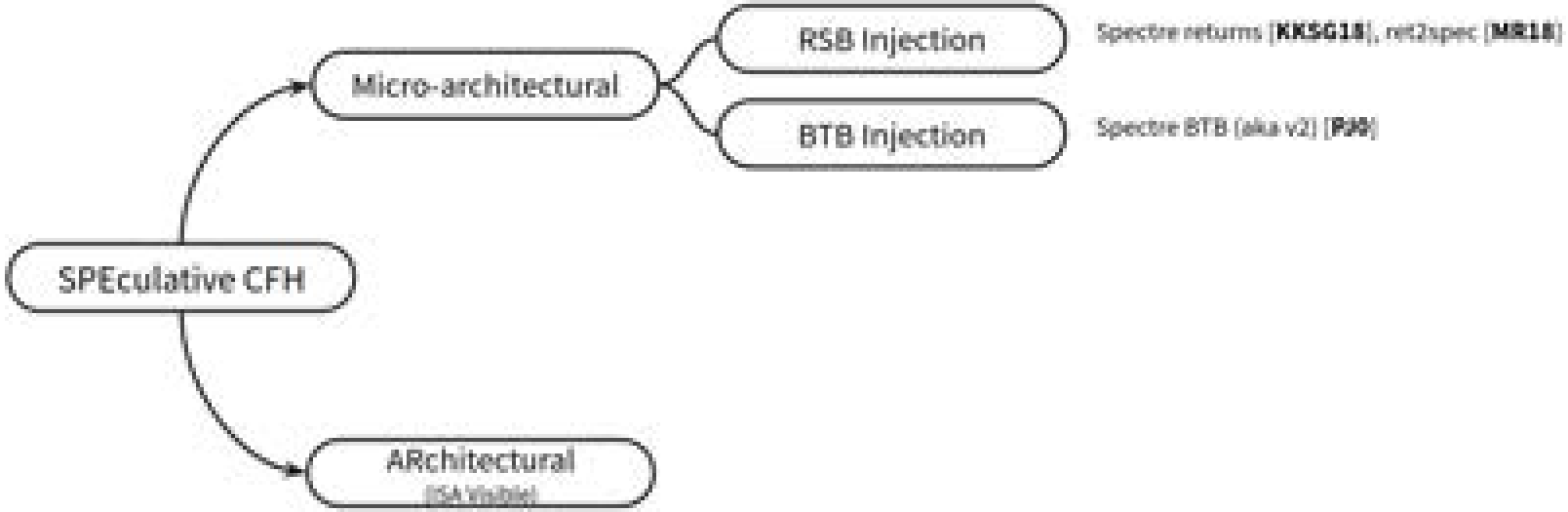
SPEculative ARchitectural control flow hijacks (SPEAR)



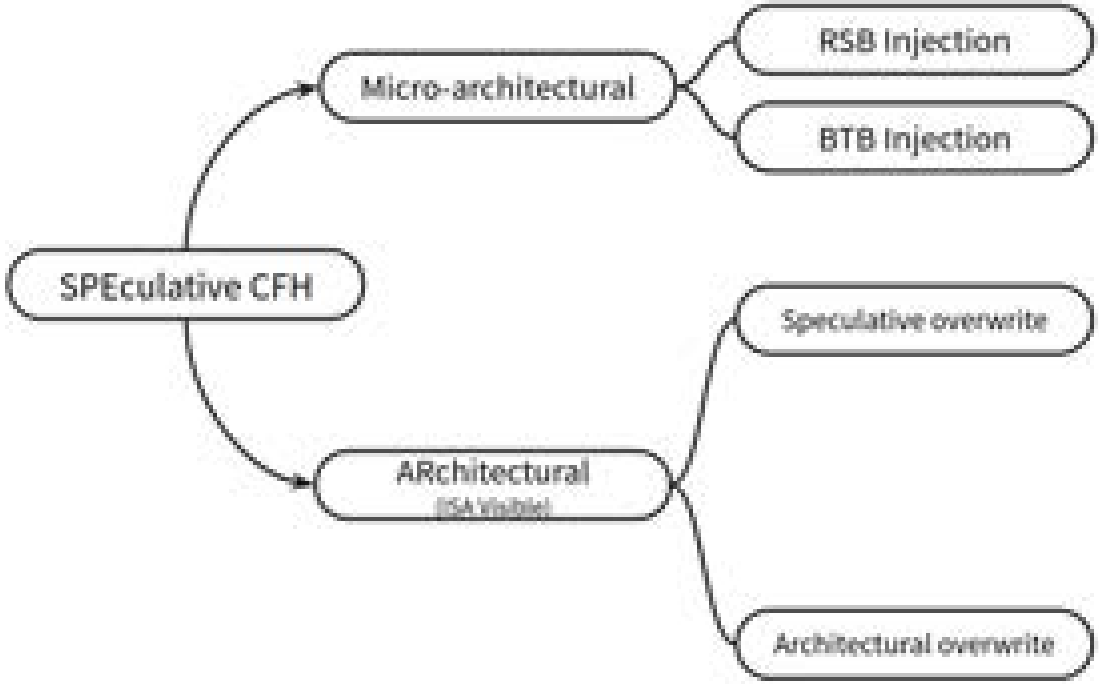
SPEculative ARchitectural control flow hijacks (SPEAR)



SPEculative ARchitectural control flow hijacks (SPEAR)



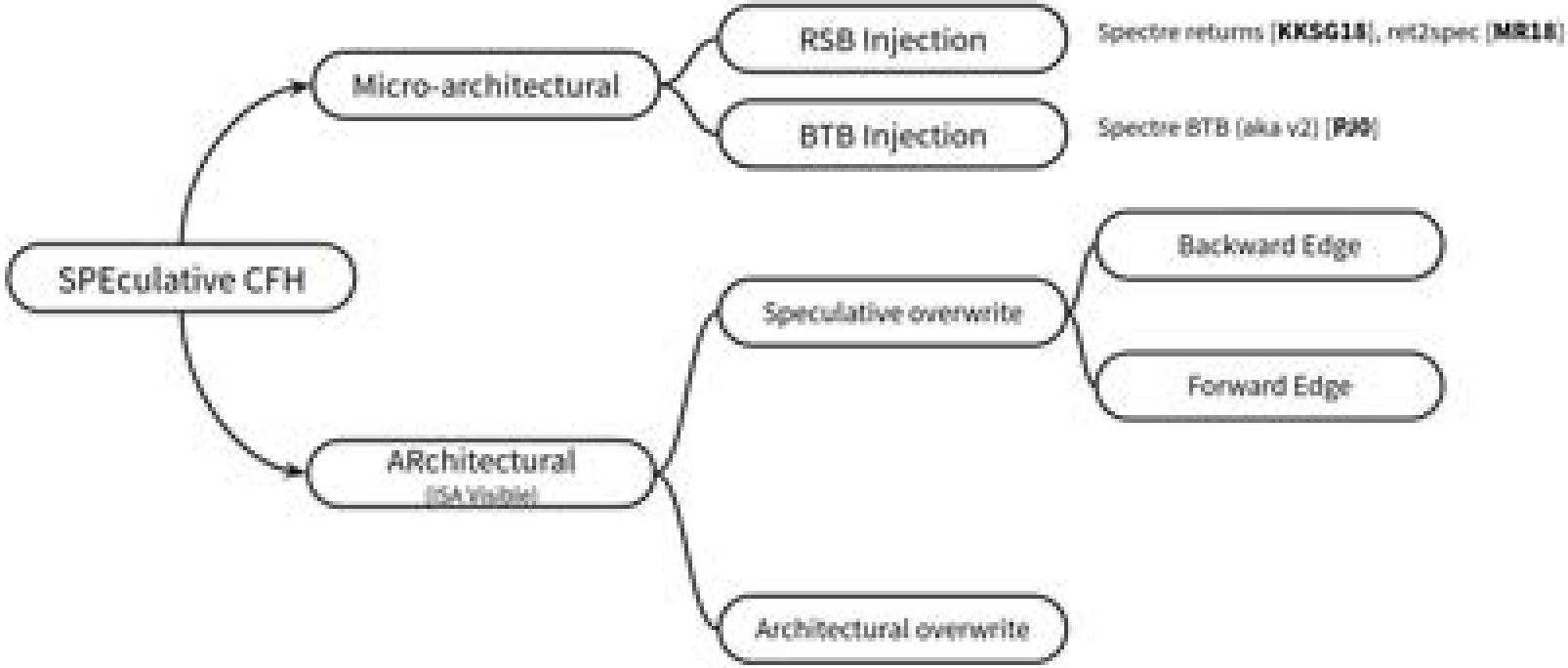
SPEculative ARchitectural control flow hijacks (SPEAR)



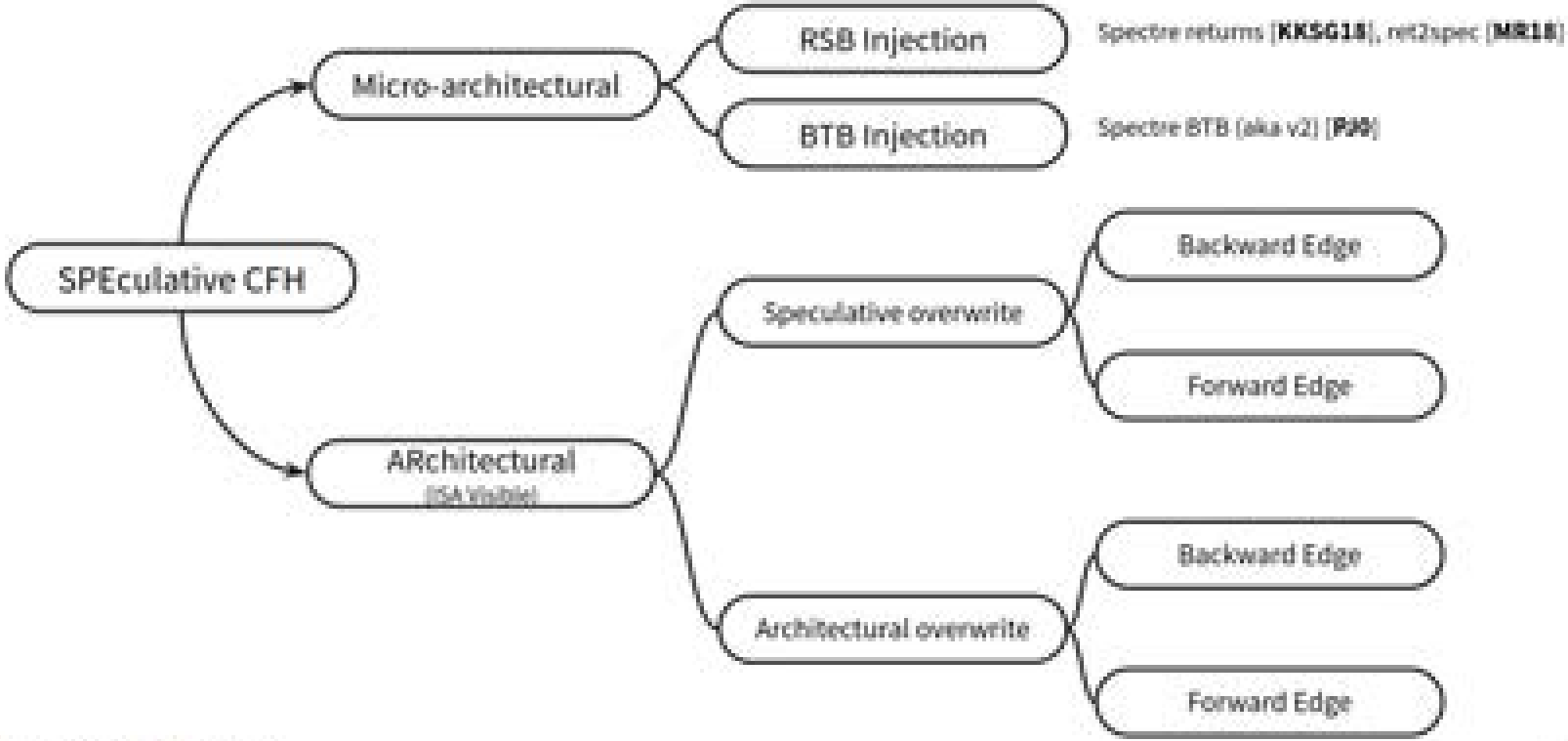
Spectre returns [**KKSG18**], ret2spec [**MR18**]

Spectre BTB (aka v2) [**PJ0**]

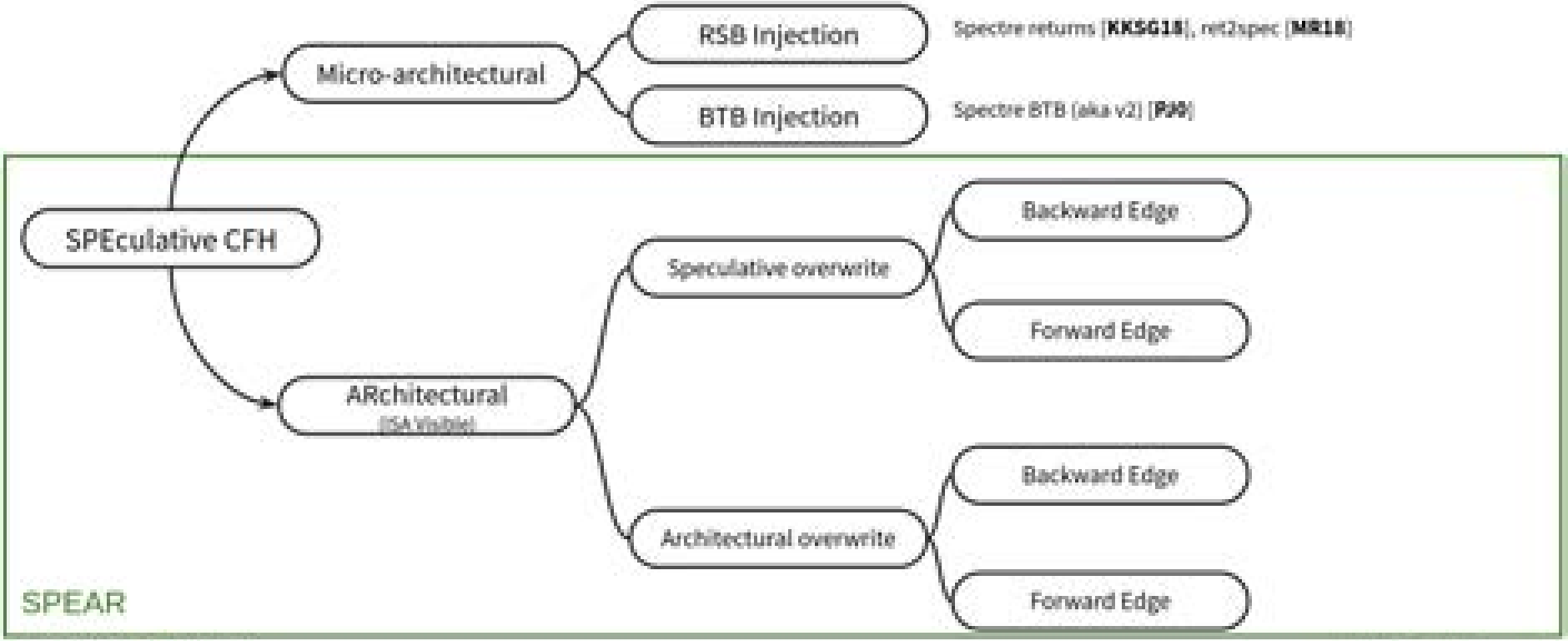
SPEculative ARchitectural control flow hijacks (SPEAR)



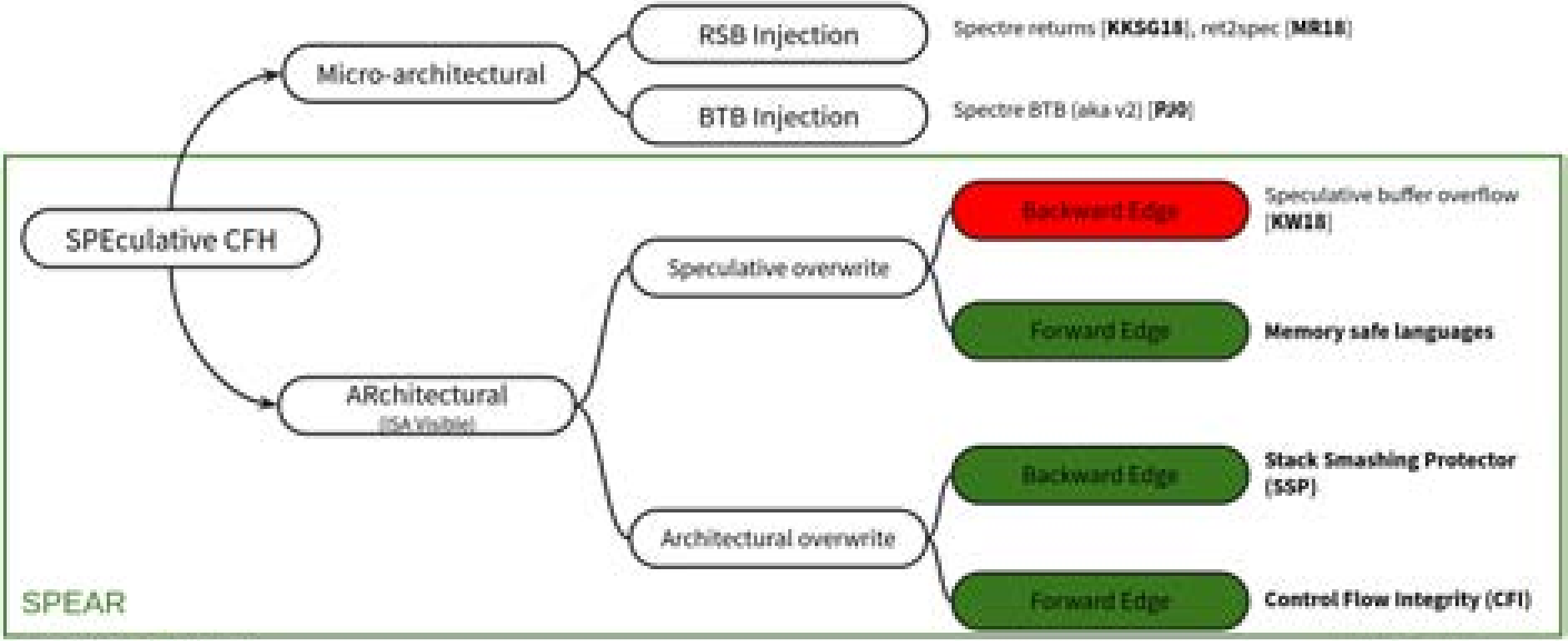
SPEculative ARchitectural control flow hijacks (SPEAR)



SPEculative ARchitectural control flow hijacks (SPEAR)



SPEculative ARchitectural control flow hijacks (SPEAR)



Speculator [ACSAC19] - Control Flow Hijack Verification

Speculator [ACSAC19] - Control Flow Hijack Verification

```
1   ; Copy of RET Value
2       mov rax, [rsp]
3       mov QWORD[stored_ret], rax
4
5   ; Architectural Overwrite
6   ;   (Attacker Controlled)
7       mov rax, QWORD[target]
8       mov [rsp], rax
9
10  ; Evict RET Value Copy
11      clflush [stored_ret]
12      lfence
13
14  ; Backward Edge Integrity Check
15  ;   (Speculation Trigger)
16      mov rax, [rsp]
17      cmp rax, QWORD [stored_ret]
18      jne my_exit
19
20  ; Backward Edge Hijack
21      ret
```

Test architectural overwrite backward edge

Speculator [ACSAC19] - Control Flow Hijack Verification

```
1   ; Copy of RET Value
2       mov rax, [rsp]
3       mov QWORD[stored_ret], rax
4
5   ;Architectural Overwrite
6   ;   (Attacker Controlled)
7       mov rax, QWORD[target]
8       mov [rsp], rax
9
10  ; Evict RET Value Copy
11      clflush [stored_ret]
12      lfence
13
14  ; Backward Edge Integrity Check
15  ;   (Speculation Trigger)
16      mov rax, [rsp]
17      cmp rax, QWORD [stored_ret]
18      jne my_exit
19
20  ; Backward Edge Hijack
21      ret
```

Test architectural overwrite backward edge

Speculator [ACSAC19] - Control Flow Hijack Verification

```
1   ; Copy of RET Value
2       mov rax, [rsp]
3       mov QWORD[stored_ret], rax
4
5   ; Architectural Overwrite
6   ;   (Attacker Controlled)
7       mov rax, QWORD[target]
8       mov [rsp], rax
9
10  ; Evict RET Value Copy
11      clflush [stored_ret]
12      lfence
13
14  ; Backward Edge Integrity Check
15  ;   (Speculation Trigger)
16      mov rax, [rsp]
17      cmp rax, QWORD [stored_ret]
18      jne my_exit
19
20  ; Backward Edge Hijack
21      ret
```

Test architectural overwrite backward edge

Speculator [ACSAC19] - Control Flow Hijack Verification

```
1   ; Copy of RET Value
2       mov rax, [rsp]
3       mov QWORD[stored_ret], rax
4
5   ; Architectural Overwrite
6   ;   (Attacker Controlled)
7       mov rax, QWORD[target]
8       mov [rsp], rax
9
10  ; Evict RET Value Copy
11      clflush [stored_ret]
12      lfence
13
14  ; Backward Edge Integrity Check
15  ;   (Speculation Trigger)
16      mov rax, [rsp]
17      cmp rax, QWORD [stored_ret]
18      jne my_exit
19
20  ; Backward Edge Hijack
21      ret
```

Test architectural overwrite backward edge

Speculator [ACSAC19] - Control Flow Hijack Verification

```
1   ; Copy of RET Value
2       mov rax, [rsp]
3       mov QWORD[stored_ret], rax
4
5   ; Architectural Overwrite
6   ;   (Attacker Controlled)
7       mov rax, QWORD[target]
8       mov [rsp], rax
9
10  ; Evict RET Value Copy
11      clflush [stored_ret]
12      lfence
13
14  ; Backward Edge Integrity Check
15  ;   (Speculation Trigger)
16      mov rax, [rsp]
17      cmp rax, QWORD [stored_ret]
18      jne my_exit
19
20  ; Backward Edge Hijack
21      ret
```

Test architectural overwrite backward edge

Speculator [ACSAC19] - Control Flow Hijack Verification

```
1  ; Copy of RET Value
2      mov rax, [rsp]
3      mov QWORD[stored_ret], rax
4
5  ; Architectural Overwrite
6  ;   (Attacker Controlled)
7      mov rax, QWORD[target]
8      mov [rsp], rax
9
10 ; Evict RET Value Copy
11     clflush [stored_ret]
12     lfence
13
14 ; Backward Edge Integrity Check
15 ;   (Speculation Trigger)
16     mov rax, [rsp]
17     cmp rax, QWORD [stored_ret]
18     jne my_exit
19
20 ; Backward Edge Hijack
21     ret
```

Test architectural overwrite backward edge

Speculator [ACSAC19] - Control Flow Hijack Verification

```
1   ; Copy of RET Value
2       mov rax, [rsp]
3       mov QWORD[stored_ret], rax
4
5   ; Architectural Overwrite
6   ;   (Attacker Controlled)
7       mov rax, QWORD[target]
8       mov [rsp], rax
9
10  ; Evict RET Value Copy
11      clflush [stored_ret]
12      lfence
13
14  ; Backward Edge Integrity Check
15  ;   (Speculation Trigger)
16      mov rax, [rsp]
17      cmp rax, QWORD [stored_ret]
18      jne my_exit
19
20  ; Backward Edge Hijack
21      ret
```

Test architectural overwrite backward edge

	Architectural		Speculative	
Family	*Fwd*	*Bwd*	*Fwd*	*Bwd*
Intel Broadwell	99.5	94.9	99.5	98.7
Intel Skylake	97.6	98.3	98.2	92.1
Intel Coffee Lake	99.8	98.1	99.7	99.4
Intel Kabylake	99.5	95.9	100	99.5
AMD Ryzen	100	100	100	100

Success rates in % of our tests

Speculator [ACSAC19] - Control Flow Hijack Verification

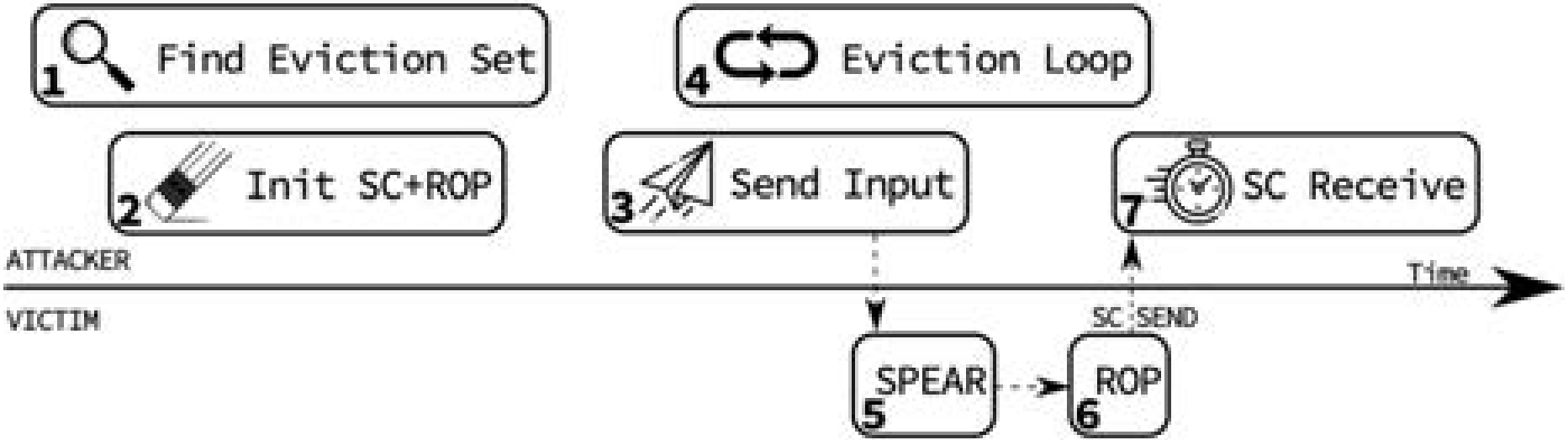
```
1   ; Copy of RET Value
2       mov rax, [rsp]
3       mov QWORD[stored_ret], rax
4
5   ; Architectural Overwrite
6   ;   (Attacker Controlled)
7       mov rax, QWORD[target]
8       mov [rsp], rax
9
10  ; Evict RET Value Copy
11      clflush [stored_ret]
12      lfence
13
14  ; Backward Edge Integrity Check
15  ;   (Speculation Trigger)
16      mov rax, [rsp]
17      cmp rax, QWORD [stored_ret]
18      jne my_exit
19
20  ; Backward Edge Hijack
21      ret
```

Test architectural overwrite backward edge

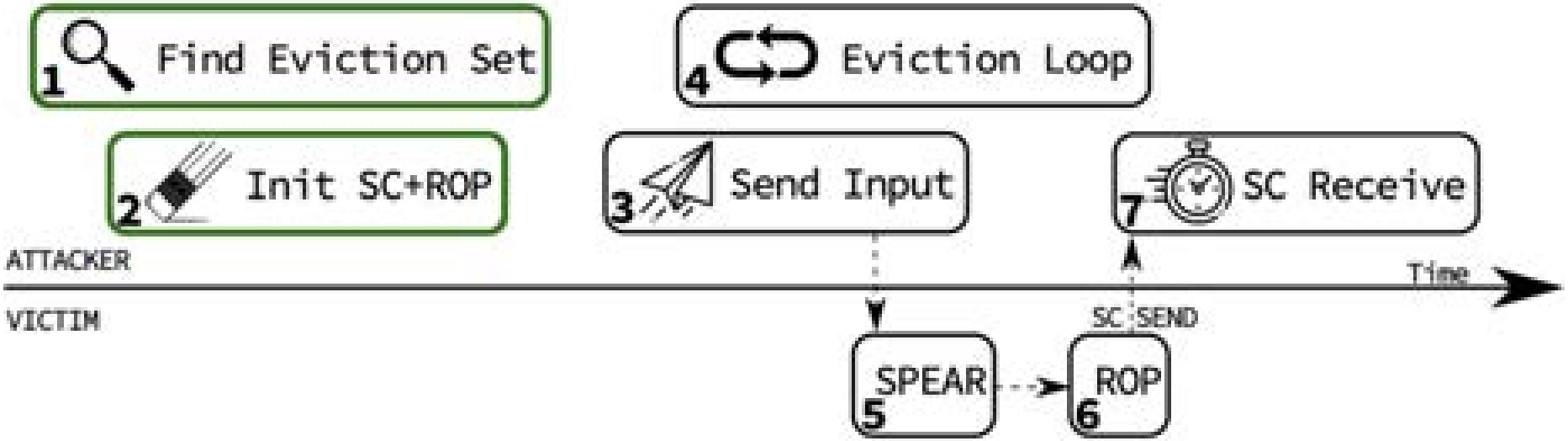
	Architectural		Speculative	
Family	*Fwd*	*Bwd*	*Fwd*	*Bwd*
Intel Broadwell	99.5	94.9	99.5	98.7
Intel Skylake	97.6	98.3	98.2	92.1
Intel Coffee Lake	99.8	98.1	99.7	99.4
Intel Kabylake	99.5	95.9	100	99.5
AMD Ryzen	100	100	100	100

Success rates in % of our tests

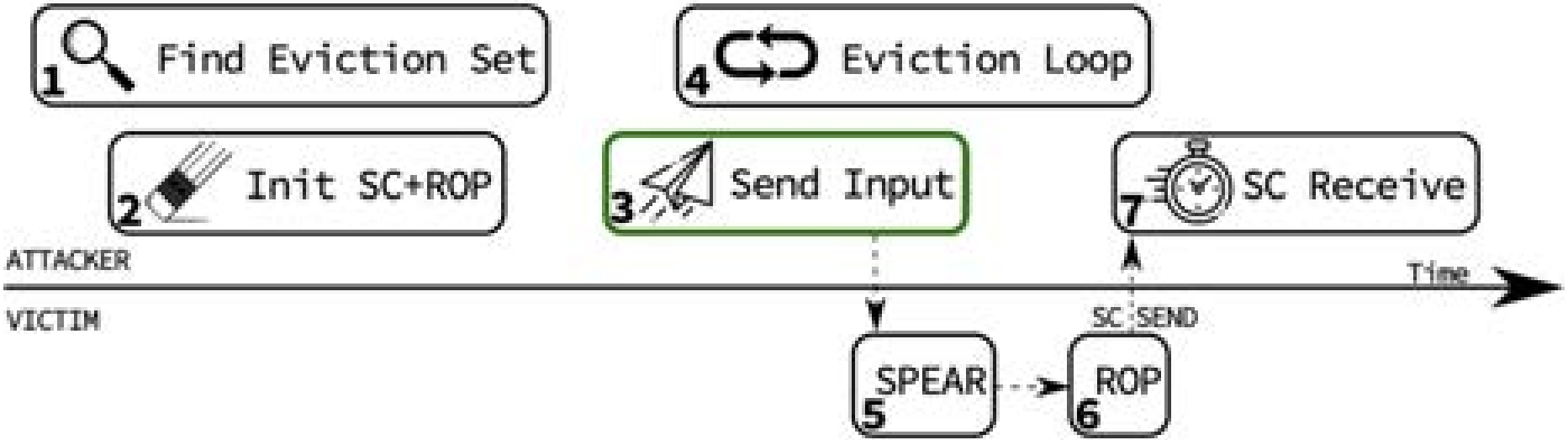
Local arbitrary memory read



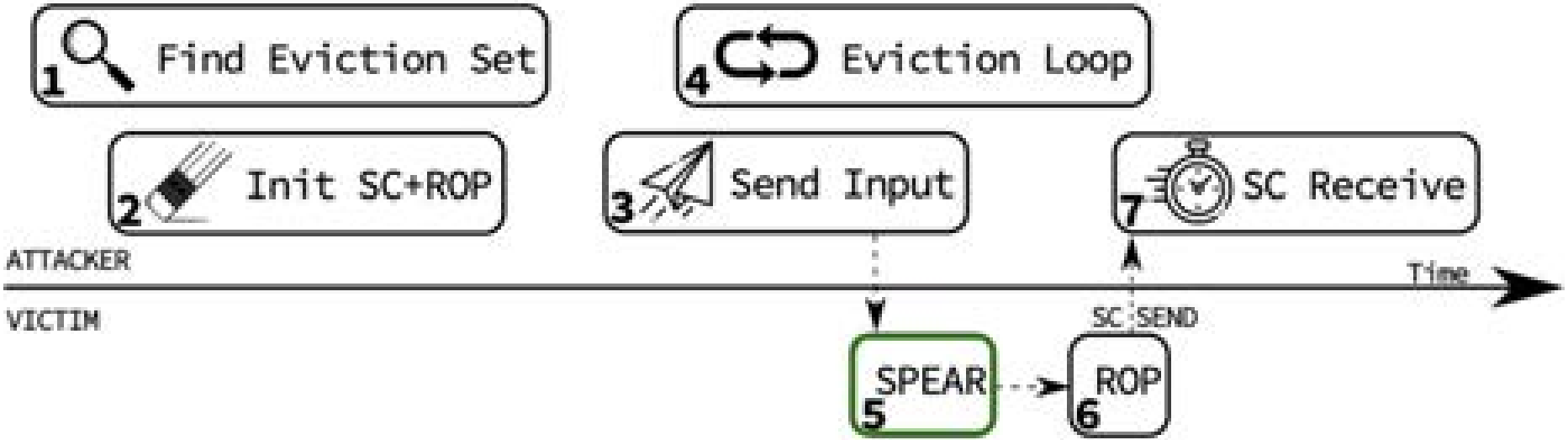
Local arbitrary memory read



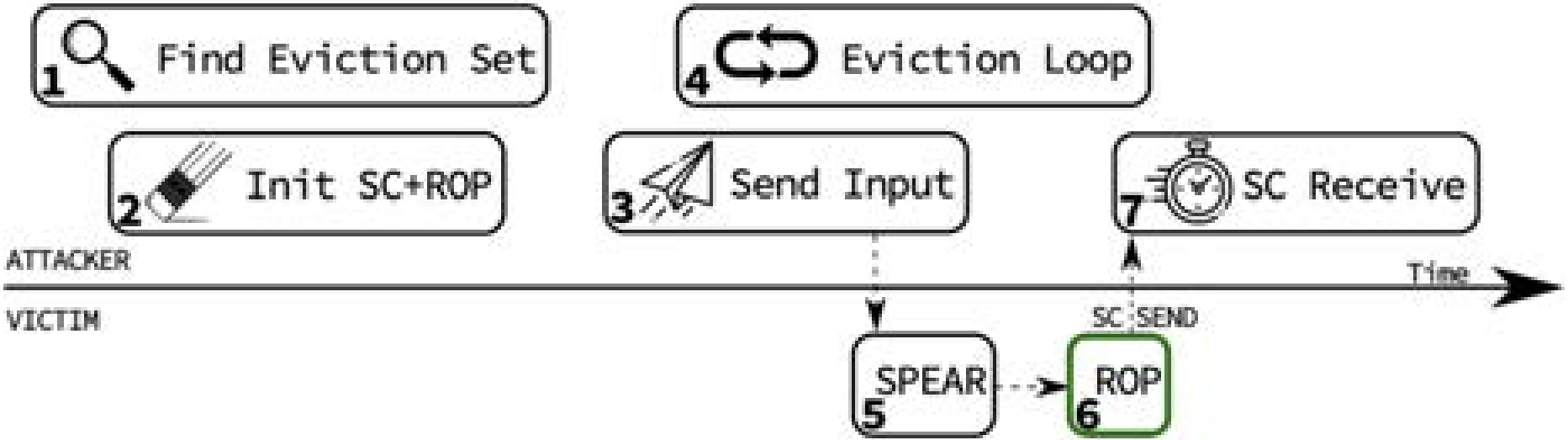
Local arbitrary memory read



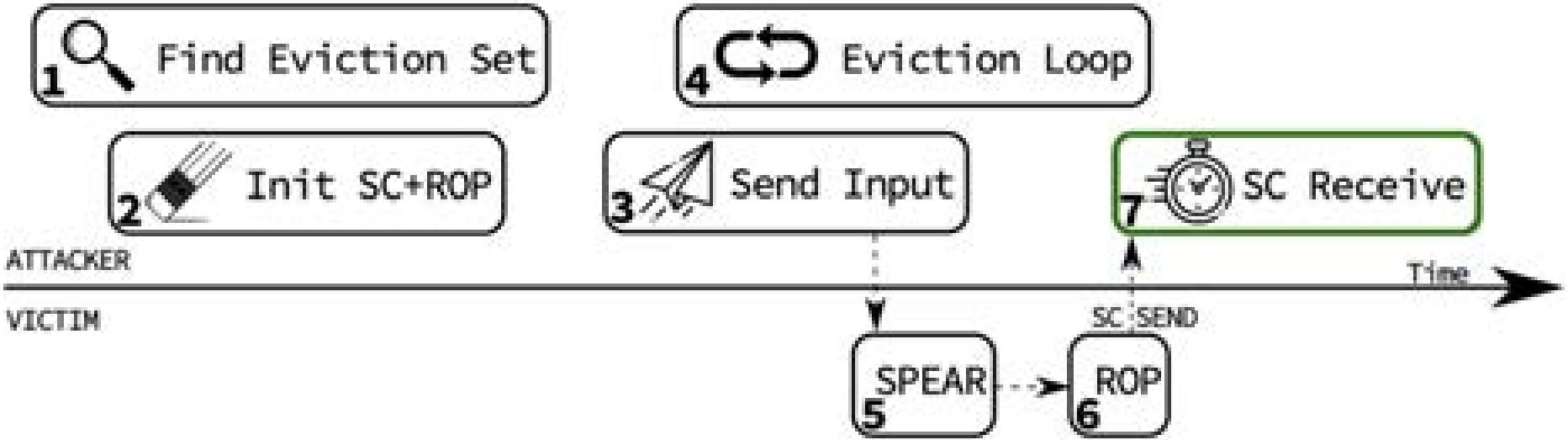
Local arbitrary memory read



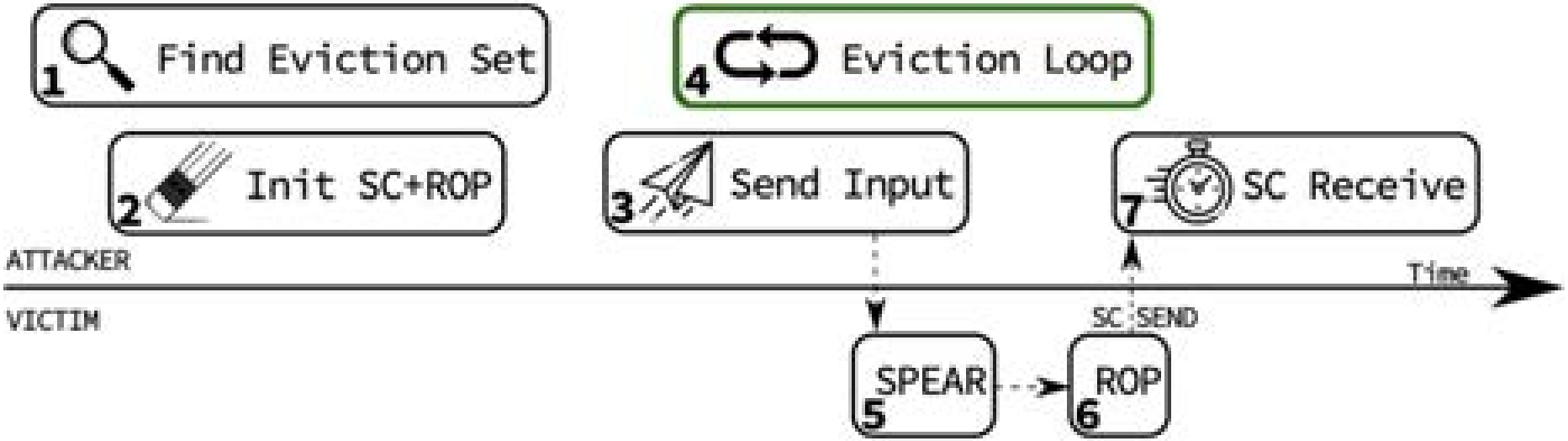
Local arbitrary memory read



Local arbitrary memory read



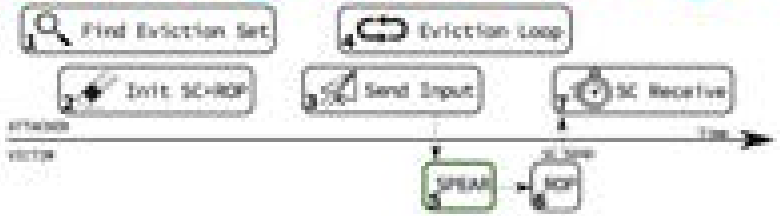
Local arbitrary memory read



Attacking Stack Smashing Protection (SSP)

Attacking Stack Smashing Protection (SSP)

Attacking Stack Smashing Protection (SSP)



Attacking Stack Smashing Protection (SSP)

```
void f() {
   char buf[size];
   memcpy(buf, src, src_size);
   if (!ok(canary)) {
      abort();
   }
   return;
}
```

Attacking Stack Smashing Protection (SSP)

```
void f() {
   char buf[size];
   memcpy(buf, src, src_size);
   if (!ok(canary)) {
      abort();
   }
   return;
}
```

Attacking Stack Smashing Protection (SSP)

```
void f() {
   char buf[size];
   memcpy(buf, src, src_size);
   if (!ok(canary)) {
      abort();
   }
   return;
}
```

Stack buffer overflow

Attacking Stack Smashing Protection (SSP)

```
void f() {
   char buf[size];
   memcpy(buf, src, src_size);
   if (!ok(canary)) {
      abort();
   }
   return;
}
```

Stack buffer overflow

ROP gadget
saved RBP
canary
....
buf[size - 1]
....
buf[1]
buf[0]

Attacker controlled

Not cached

Attacking Stack Smashing Protection (SSP)

```
void f() {
   char buf[size];
   memcpy(buf, src, src_size);
   if (!ok(canary)) {
      abort();
   }
   return;
}
```

Speculative execution trigger

Attacker controlled

Not cached

Attacking Stack Smashing Protection (SSP)

```
void f() {
   char buf[size];
   memcpy(buf, src, src_size);
   if (!ok(canary)) {
      abort();
   }
   return;
}
```

Attacking Stack Smashing Protection (SSP)

```
void f() {
   char buf[size];
   memcpy(buf, src, src_size);
   if (!ok(canary)) {
      abort();
   }
   return;
}
```

```
; Store canary on stack
      mov rbx, QWORD[fs:0x28]
      mov QWORD[stack_canary], rbx

; Function body

; Check for corrupted canary
      mov rbx, QWORD[stack_canary]
      xor QWORD[fs:0x28], rbx
      je exit
      call __stack_chk_fail

exit:
   ; Epilogue
      ret
```

Attacking Stack Smashing Protection (SSP)

```
void f() {
   char buf[size];
   memcpy(buf, src, src_size);
   if (!ok(canary)) {
      abort();
   }
   return;
}
```

Attacking Stack Smashing Protection (SSP)

```
void f() {
   char buf[size];
   memcpy(buf, src, src_size);
   if (!ok(canary)) {
      abort();
   }
   return;
}
```

Speculative control flow hijack
architectural edge overwrite

Attacker controlled

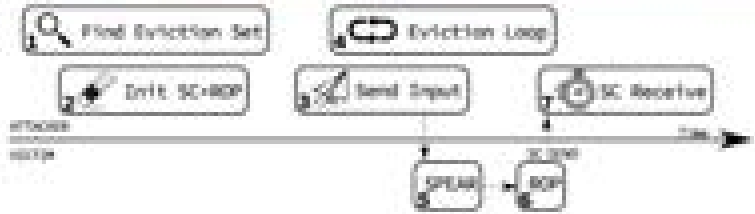
Not cached

Attacking Stack Smashing Protection (SSP)

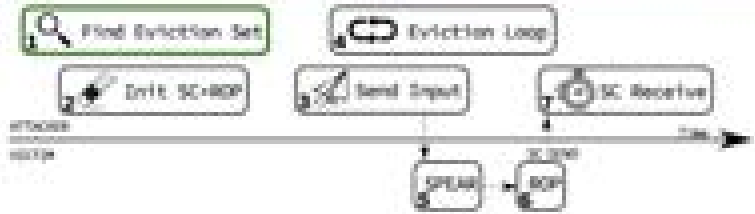
```
void f() {
   char buf[size];
   memcpy(buf, src, src_size);
   if (!ok(canary)) {
      abort();
   }
   return
}
```

Canary eviction

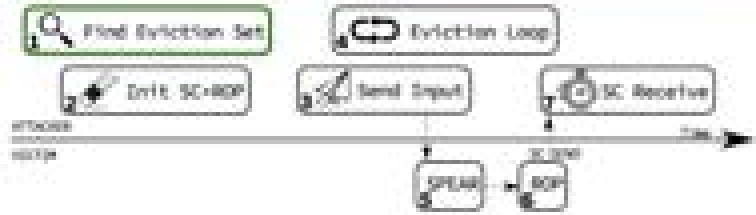
Canary eviction



Canary eviction



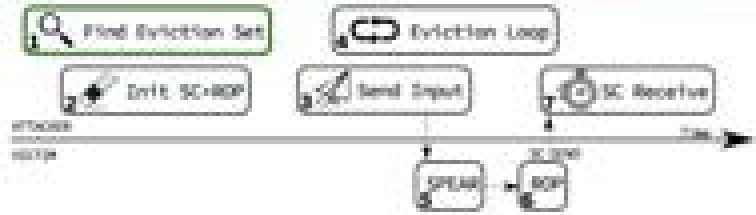
Canary eviction



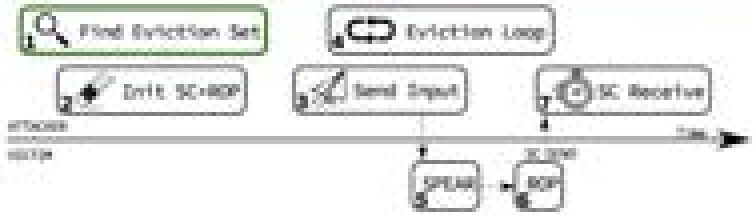
Attacker Program

Attacker Program

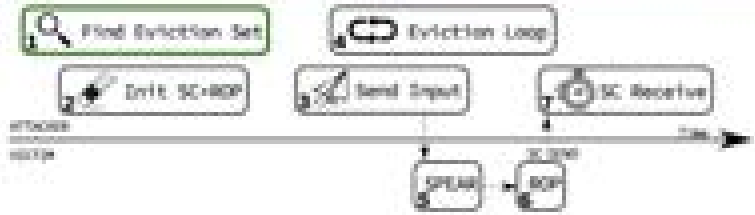
Canary eviction



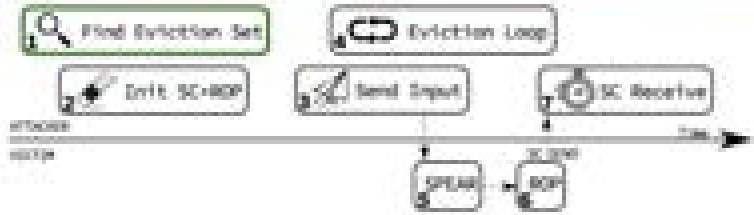
Canary eviction



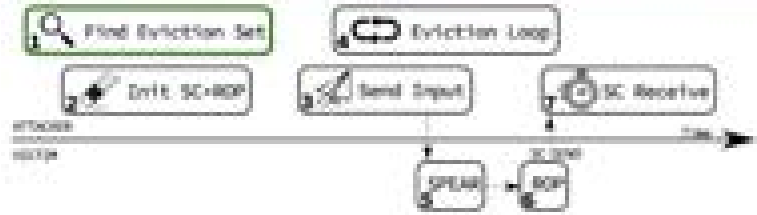
Canary eviction



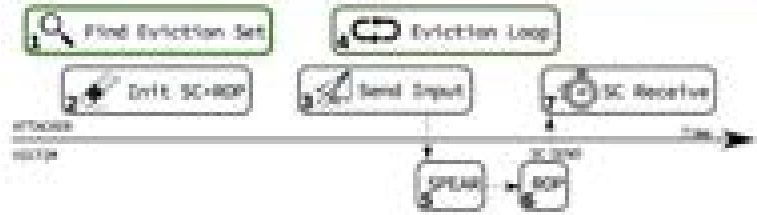
Canary eviction

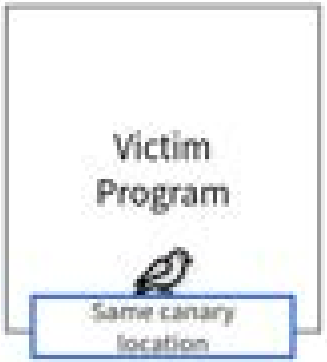


Canary eviction

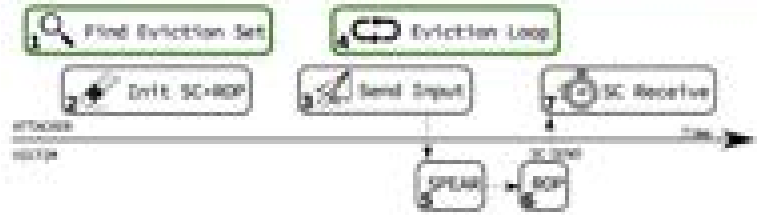


Canary eviction

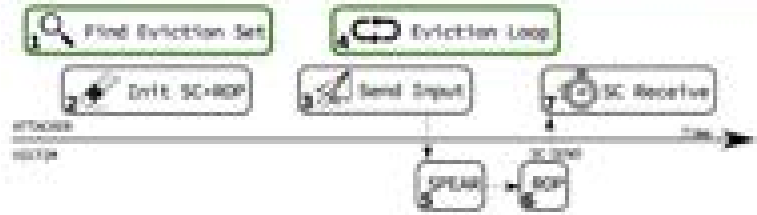




Canary eviction



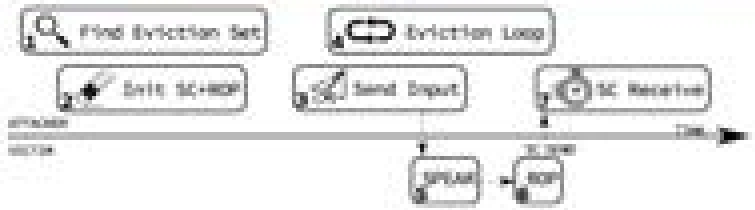
Canary eviction



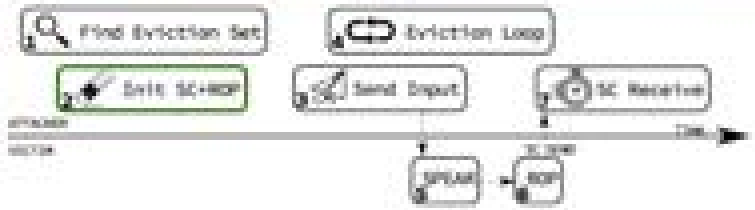
Not cached

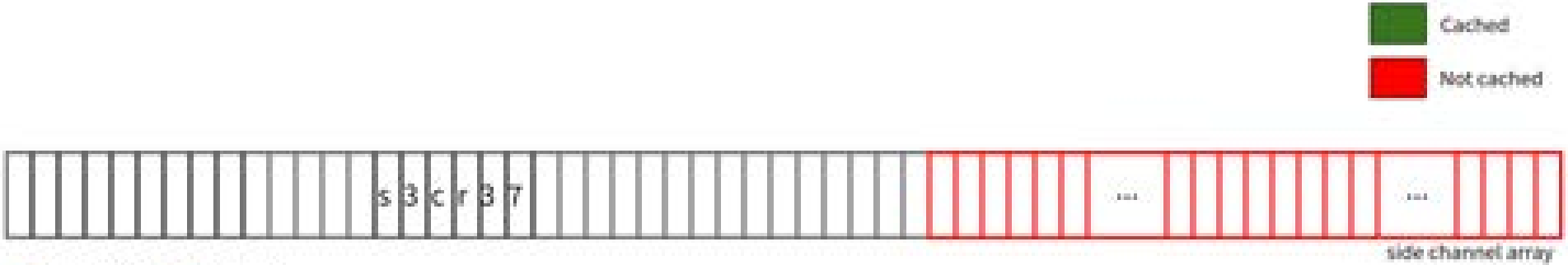
Speculative ROP

Speculative ROP



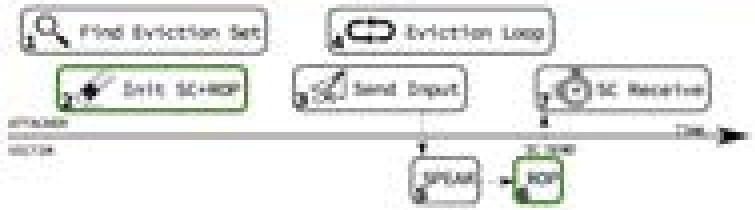
Speculative ROP

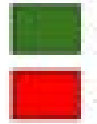


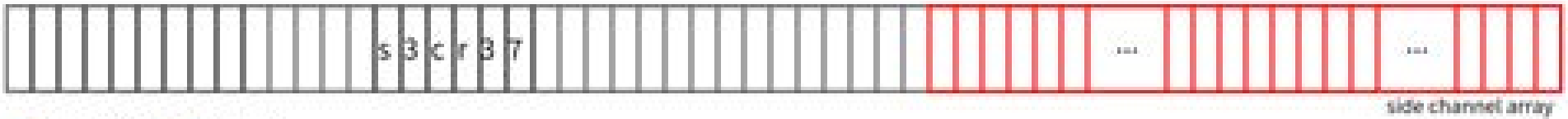


Speculative ROP

```
; Load victim secret
    mov rax, BYTE[victim_addr]
    ret

; Compute side-channel array entry
    shl rax, 8
    ret

; Add side-channel array base
    add rax, base_address
    ret

; Access side-channel array entry
;   (Side-channel Send Signal)
    mov rax, QWORD[rax]
    ret
```

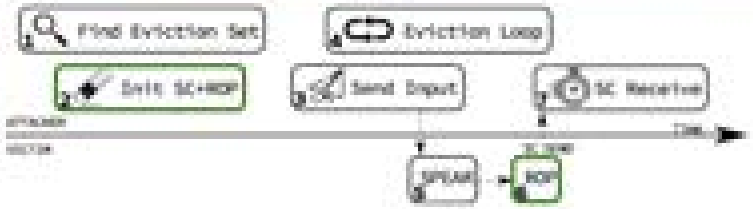


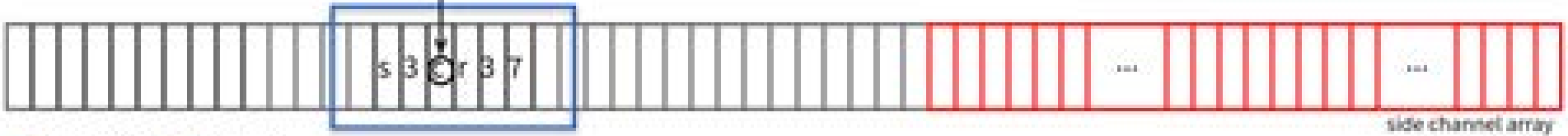




Speculative ROP

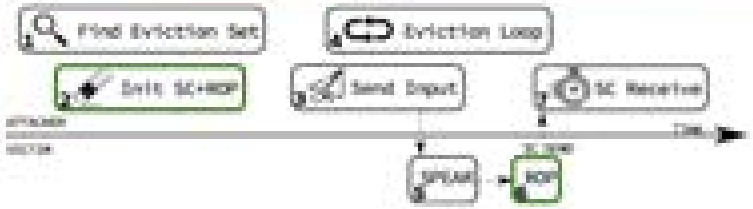
```
; Load victim secret
    mov rax, BYTE[victim_addr]
    ret

; Compute side-channel array entry
    shl rax, 8
    ret

; Add side-channel array base
    add rax, base_address
    ret

; Access side-channel array entry
;   (Side-channel Send Signal)
    mov rax, QWORD[rax]
    ret
```

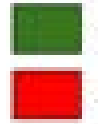


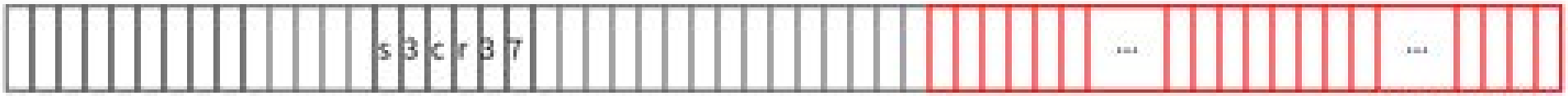


Speculative ROP

```
; Load victim secret
    mov rax, BYTE[victim_addr]
    ret

; Compute side-channel array entry
    shl rax, 8
    ret

; Add side-channel array base
    add rax, base_address
    ret

; Access side-channel array entry
;   (Side-channel Send Signal)
    mov rax, QWORD[rax]
    ret
```



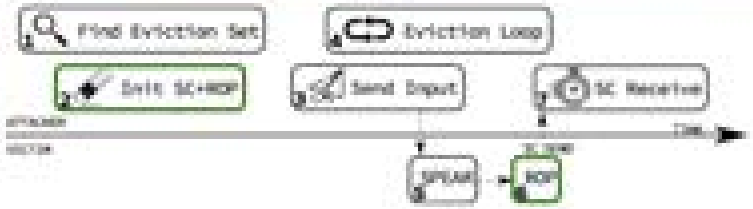


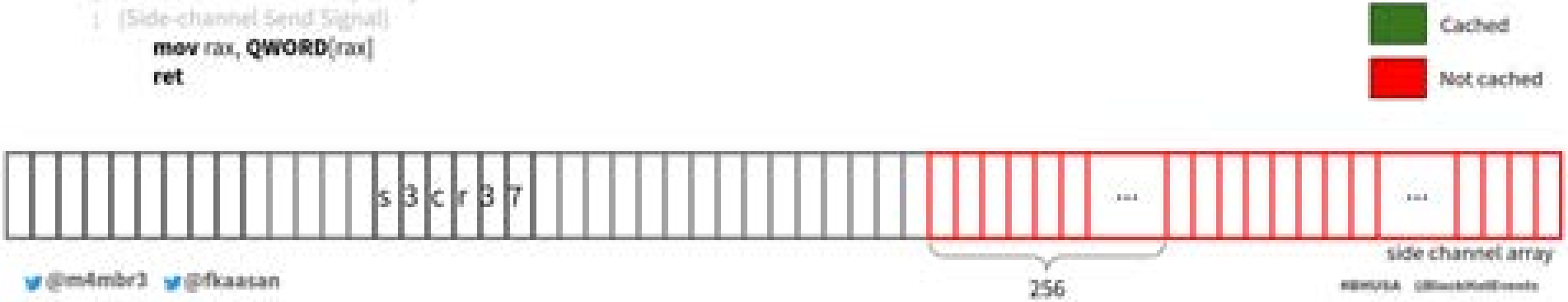


side channel array

Speculative ROP

```
; Load victim secret
    mov rax, BYTE[victim_addr]
    ret

; Compute side-channel array entry
    shl rax, 8
    ret

; Add side-channel array base
    add rax, base_address
    ret

; Access side-channel array entry
;   (Side-channel Send Signal)
    mov rax, QWORD[rax]
    ret
```

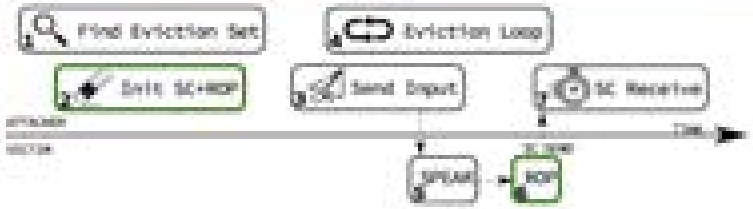


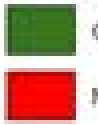


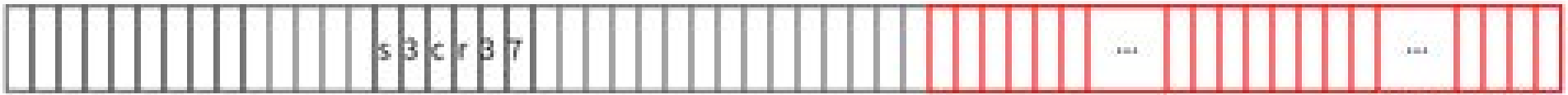
Speculative ROP

```
; Load victim secret
    mov rax, BYTE[victim_addr]
    ret

; Compute side-channel array entry
    shl rax, 8
    ret

; Add side-channel array base
    add rax, base_address
    ret
```

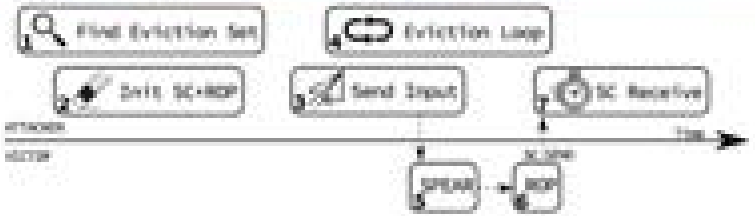
```
; Access side-channel array entry
;   (Side-channel Send Signal)
    mov rax, QWORD[rax]
    ret
```







Arbitrary memory read with a side-channel

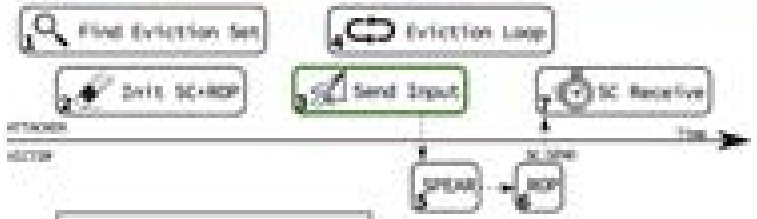


Arbitrary memory read with a side-channel

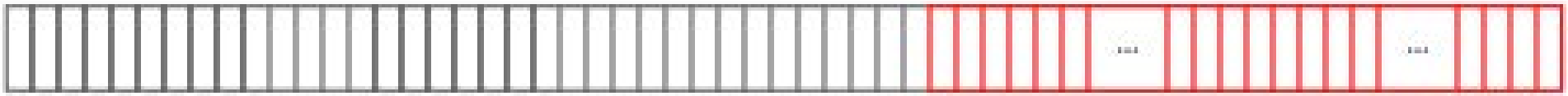
Attacker Program

Victim Program

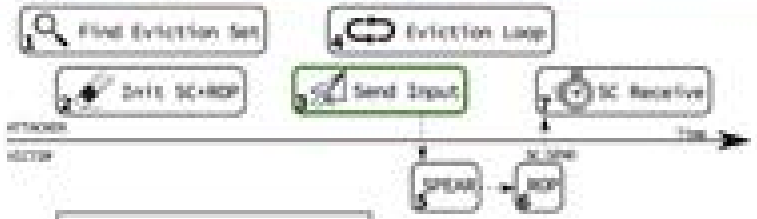
Arbitrary memory read with a side-channel

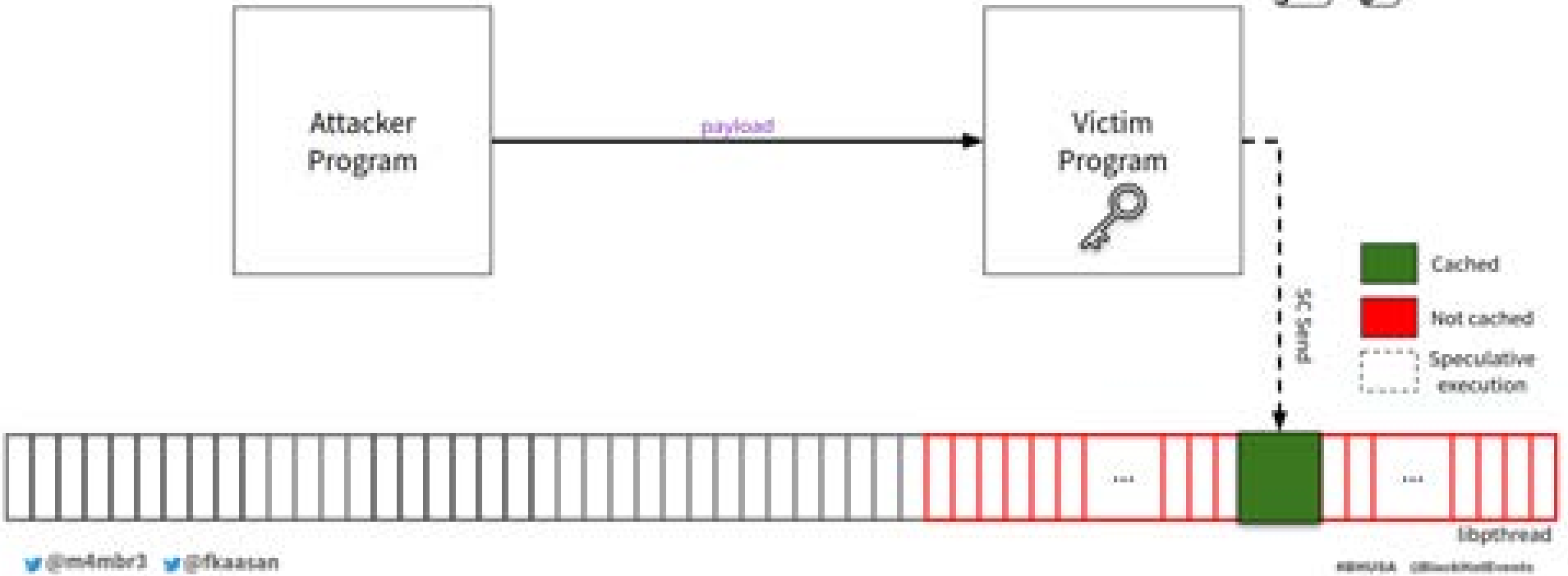


Arbitrary memory read with a side-channel

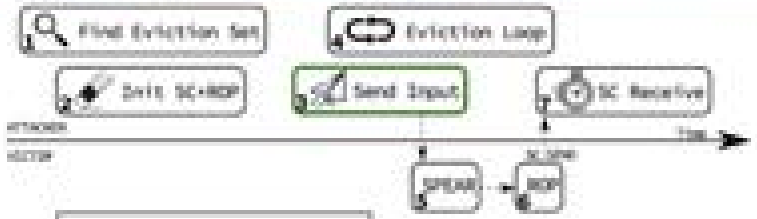


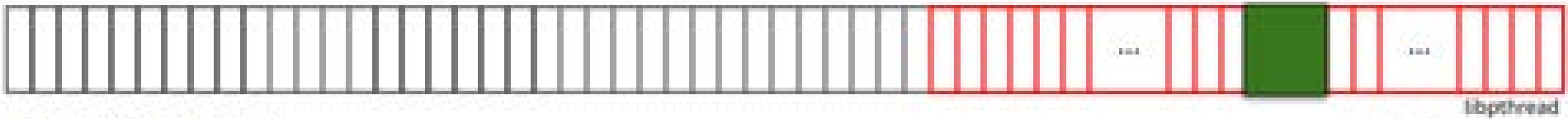
Arbitrary memory read with a side-channel



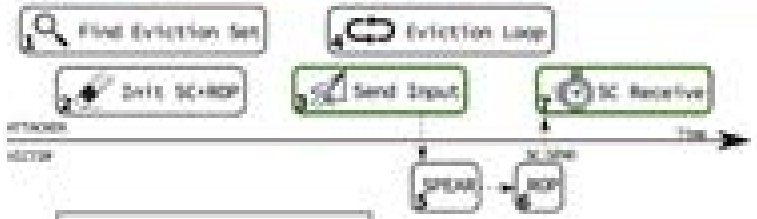


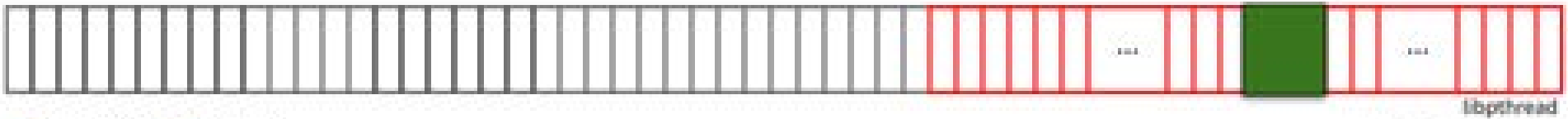
Arbitrary memory read with a side-channel



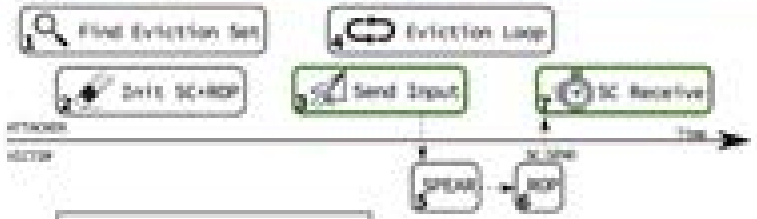


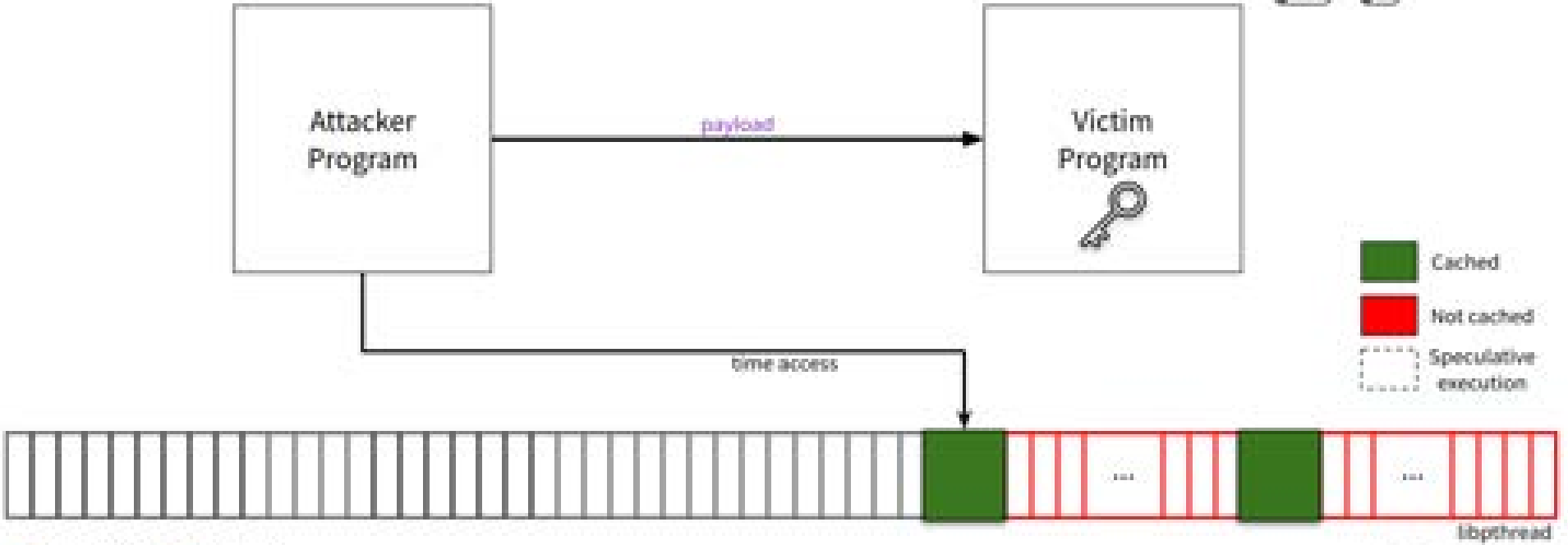
Arbitrary memory read with a side-channel



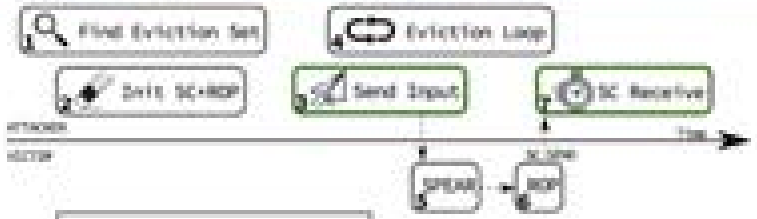


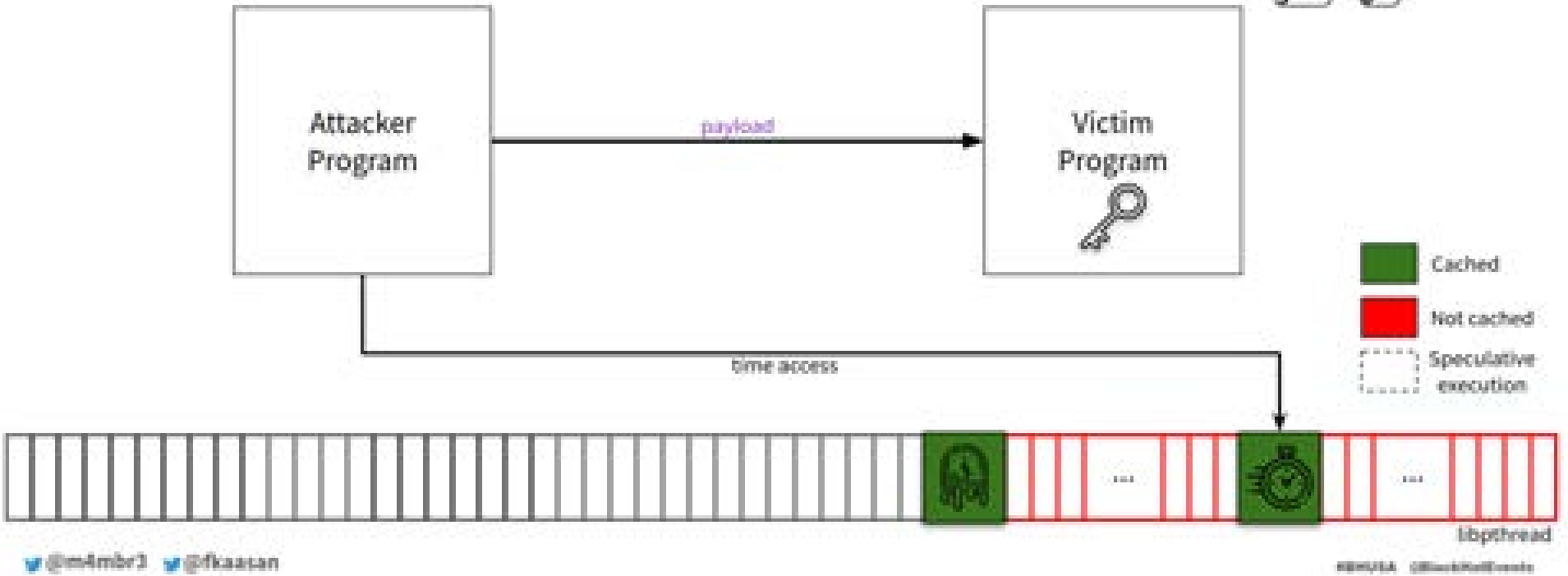
Arbitrary memory read with a side-channel



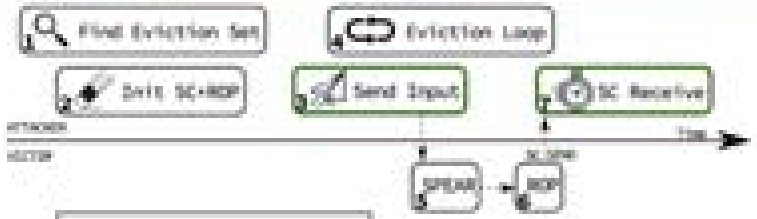


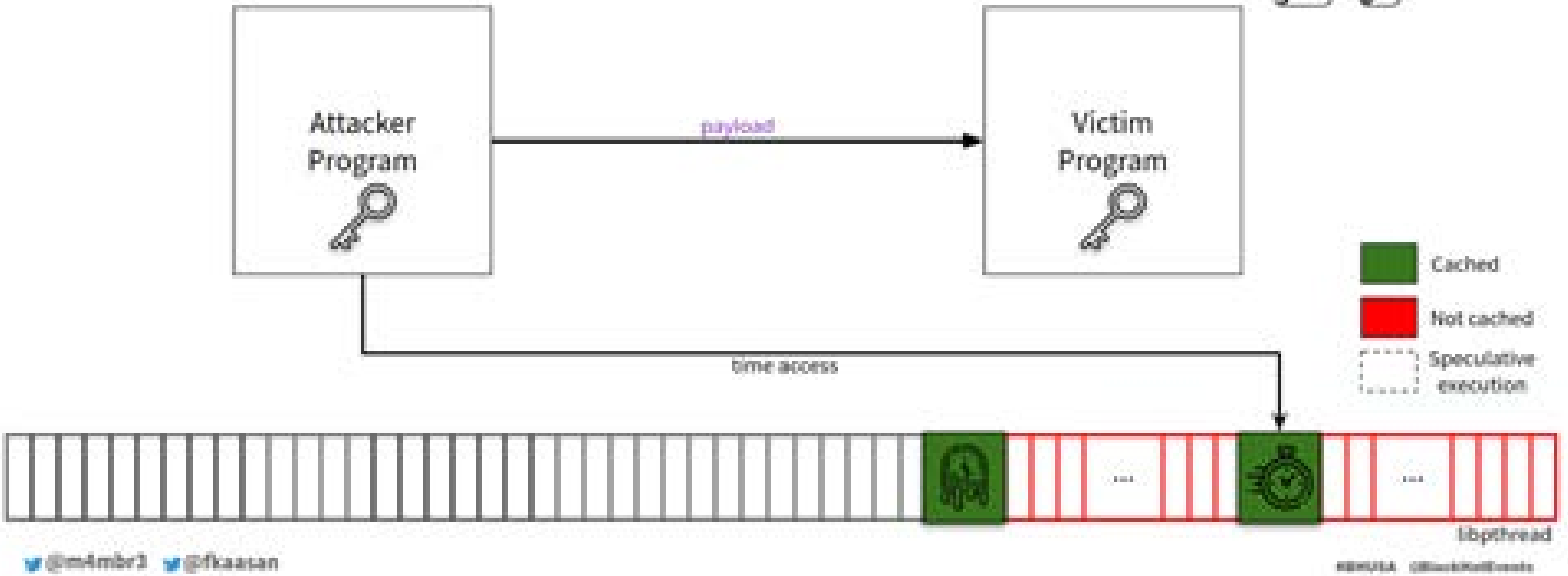
Arbitrary memory read with a side-channel





Arbitrary memory read with a side-channel





Results

Local arbitrary victim memory read

Leak rate for 2-digits sized secret: **0.3 Bytes per second**

100 attack runs per byte to reduce noise (shared library as side-channel array)

Works on Intel **Skylake** and **Coffee Lake** with all Spectre mitigations enabled

Slight success rate change between Ubuntu 16.04 and Ubuntu 20.04

Discussion about attack optimization and better synchronization are in our paper

Other SPEAR case - memory safe languages

[1] https://github.com/golang/go/wiki/Spectre

Other SPEAR case - memory safe languages

Golang and Rust are example of affected languages

[1] https://github.com/golang/go/wiki/Spectre

Other SPEAR case - memory safe languages

Golang and Rust are example of affected languages

For Go: reported in Nov 2019 and discussed with Go developers fixed in Go v1.15 (-spectre flag)[1]

[1] https://github.com/golang/go/wiki/Spectre

Other SPEAR case - memory safe languages

Golang and Rust are example of affected languages

For Go: reported in Nov 2019 and discussed with Go developers fixed in Go v1.15 (-spectre flag)[1]

For Rust: reported in Dec 2020 and currently in discussion

[1] https://github.com/golang/go/wiki/Spectre

Other SPEAR case - memory safe languages

Golang and Rust are example of affected languages

For Go: reported in Nov 2019 and discussed with Go developers fixed in Go v1.15 (-spectre flag)[1]

For Rust: reported in Dec 2020 and currently in discussion

More details in our EuroSP 2021 paper: https://arxiv.org/abs/2003.05503

[1] https://github.com/golang/go/wiki/Spectre

Other SPEAR case - control flow integrity

Other SPEAR case - control flow integrity

We considered both **LLVM-CFI** and **GCC VTV** control flow integrity implementations

Other SPEAR case - control flow integrity

We considered both **LLVM-CFI** and **GCC VTV** control flow integrity implementations

LLVM-CFI is **NOT** bypassable because of its design

Other SPEAR case - control flow integrity

We considered both **LLVM-CFI** and **GCC VTV** control flow integrity implementations

LLVM-CFI is **NOT** bypassable because of its design

GCC VTV mechanism can instead be used to start a SPEAR attack

Other SPEAR case - control flow integrity

We considered both **LLVM-CFI** and **GCC VTV** control flow integrity implementations

LLVM-CFI is **NOT** bypassable because of its design

GCC VTV mechanism can instead be used to start a SPEAR attack

Design choice of LLVM-CFI demonstrate how critical the implementation of these mitigations is

Other SPEAR case - control flow integrity

We considered both **LLVM-CFI** and **GCC VTV** control flow integrity implementations

LLVM-CFI is **NOT** bypassable because of its design

GCC VTV mechanism can instead be used to start a SPEAR attack

Design choice of LLVM-CFI demonstrate how critical the implementation of these mitigations is

More details in our EuroSP 2021 paper: https://arxiv.org/abs/2003.05503

Mitigations - inlining instrumentation

Mitigations - inlining instrumentation

These mitigations require instrumentation of the application

Mitigations - inlining instrumentation

These mitigations require instrumentation of the application

Some compiler pass already exists (e.g., LLVM Speculative Load Hardening)

Mitigations - inlining instrumentation

These mitigations require instrumentation of the application

Some compiler pass already exists (e.g., LLVM Speculative Load Hardening)

Generally, they are very hard to apply due to the trade-off coverage/overhead

Mitigations - inlining instrumentation

These mitigations require instrumentation of the application

Some compiler pass already exists (e.g., LLVM Speculative Load Hardening)

Generally, they are very hard to apply due to the trade-off coverage/overhead

Examples:

Mitigations - inlining instrumentation

These mitigations require instrumentation of the application

Some compiler pass already exists (e.g., LLVM Speculative Load Hardening)

Generally, they are very hard to apply due to the trade-off coverage/overhead

Examples:

Fencing instructions like **lfence** or **mfence**

Mitigations - inlining instrumentation

These mitigations require instrumentation of the application

Some compiler pass already exists (e.g., LLVM Speculative Load Hardening)

Generally, they are very hard to apply due to the trade-off coverage/overhead

Examples:

Fencing instructions like **lfence** or **mfence**

Branchless masking

Mitigations - inlining instrumentation

These mitigations require instrumentation of the application

Some compiler pass already exists (e.g., LLVM Speculative Load Hardening)

Generally, they are very hard to apply due to the trade-off coverage/overhead

Examples:

- Fencing instructions like **lfence** or **mfence**
- Branchless masking
- Retpoline

Mitigations - by design

Mitigations - by design

Data flow analysis within the CPU pipeline with blocking of unsafe operations (e.g., NDA, STT, and Dolma)

Mitigations - by design

Data flow analysis within the CPU pipeline with blocking of unsafe operations (e.g., NDA, STT, and Dolma)

New cache designs to remove side/covert channels (e.g., InvisiSpec, Usafe Speculative Loads, and CleanupSpec)

Mitigations - by design

Data flow analysis within the CPU pipeline with blocking of unsafe operations (e.g., NDA, STT, and Dolma)

New cache designs to remove side/covert channels (e.g., InvisiSpec, Usafe Speculative Loads, and CleanupSpec)

The latter approach protects from SPEAR only when a cache side channel is used but not other types (e.g., port contention [**CCS19**] or BTB [**WOOT19**])

Mitigations - by design

Data flow analysis within the CPU pipeline with blocking of unsafe operations (e.g., NDA, STT, and Dolma)

New cache designs to remove side/covert channels (e.g., InvisiSpec, Usafe Speculative Loads, and CleanupSpec)

The latter approach protects from SPEAR only when a cache side channel is used but not other types (e.g., port contention [**CCS19**] or BTB [**WOOT19**])

Only available in future iterations of CPUs

Takeaways

Takeaways

SPEAR attacks bypass mitigations and memory safety to leak confidential data

Takeaways

SPEAR attacks bypass mitigations and memory safety to leak confidential data

=> **New and old mitigations must be analyzed and possibly modified to withstand SPEAR attacks**

Takeaways

SPEAR attacks bypass mitigations and memory safety to leak confidential data

=> **New and old mitigations must be analyzed and possibly modified to withstand SPEAR attacks**

These attacks are complex but practical

=> **with new tools to aid building each attack stage, they could become more practical**

Takeaways

SPEAR attacks bypass mitigations and memory safety to leak confidential data

=> **New and old mitigations must be analyzed and possibly modified to withstand SPEAR attacks**

These attacks are complex but practical

=> **with new tools to aid building each attack stage, they could become more practical**

Speculative ROP is possible and eases the task of finding a spectre v1-like side channel send gadget

Takeaways

SPEAR attacks bypass mitigations and memory safety to leak confidential data
=> **New and old mitigations must be analyzed and possibly modified to withstand SPEAR attacks**

These attacks are complex but practical
=> **with new tools to aid building each attack stage, they could become more practical**

Speculative ROP is possible and eases the task of finding a spectre v1-like side channel send gadget

SEAs are a significant research and industry challenge for the next decade (tools, attacks and defences)

References

[PJ0] J. Horn Tech. Report https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html, 2018

[KW18] V. Kiriansky and C. Waldspurger, Speculative Buffer Overflows: Attacks and Defenses, Tech. Report https://people.csail.mit.edu/vlk/spectre11.pdf, 2018

[KHGG19] P. Kocher, J. Horn, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre Attacks: Exploiting Speculative Execution. In Proc. S&P'19.

[KKSG18] E. M. Koruyeh, K. N. Khasawneh, C. Song, and N. Abu-Ghazaleh. Spectre returns! speculation attacks using the return stack buffer. In Proc. WOOT'18.

[MR18] G. Maisuradze, and C. Rossow. ret2spec: Speculative execution using return stack buffers. In Proc. CCS 2018.

[ACSAC19] A. Mambretti, M. Neugschwandtner, A. Sorniotti, E. Kirda, W. Robertson, and A. Kurmus, Speculator: A tool to analyze speculative execution attacks and mitigations. In Proc. ACSAC'19.

[WOOT19] A. Mambretti, A. Sandulescu, M. Neugschwandtner, A. Sorniotti, and A. Kurmus, Two methods for exploiting speculative control flow hijacks. In Proc. WOOT'19.

[CCS19] A. Bhattacharyya, A. Sandulescu, M. Neugschwandtner, A. Sorniotti, B. Falsafi, M. Payer, and A. Kurmus, SMoTherSpectre: Exploiting speculative execution through port contention. In Proc. CCS 2019.

[ES&P21] A. Mambretti, A. Sandulescu, A. Sorniotti, E. Kirda, W. Robertson, and A. Kurmus, Bypassing memory safety mechanisms through speculative control flow hijacks. To appear at Euro S&P'21. Available through https://arxiv.org/abs/2003.05503, 2020.

[SANER21] A. Mambretti, P. Convertini, A. Sorniotti, A. Sandulescu, E. Kirda, and A. Kurmus. GhostBuster: understanding and overcoming the pitfalls of transient execution vulnerability checkers. In Proc. IEEE SANER 2021

References

[PJ0] J. Horn Tech. Report https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html, 2018

[KW18] V. Kiriansky and C. Waldspurger, Speculative Buffer Overflows: Attacks and Defenses, Tech. Report https://people.csail.mit.edu/vlk/spectre11.pdf, 2018

[KHGG19] P. Kocher, J. Horn, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre Attacks: Exploiting Speculative Execution. In Proc. S&P'19.

[KKSG18] E. M. Koruyeh, K. N. Khasawneh, C. Song, and N. Abu-Ghazaleh. Spectre returns! speculation attacks using the return stack buffer. In Proc. WOOT'18.

[MR18] G. Maisuradze, and C. Rossow. ret2spec: Speculative execution using return stack buffers. In Proc. CCS 2018.

[ACSAC19] A. Mambretti, M. Neugschwandtner, A. Sorniotti, E. Kirda, W. Robertson, and A. Kurmus, Speculator: A tool to analyze speculative execution attacks and mitigations. In Proc. ACSAC'19.

[WOOT19] A. Mambretti, A. Sandulescu, M. Neugschwandtner, A. Sorniotti, and A. Kurmus, Two methods for exploiting speculative control flow hijacks. In Proc. WOOT'19.

[CCS19] A. Bhattacharyya, A. Sandulescu, M. Neugschwandtner, A. Sorniotti, B. Falsafi, M. Payer, and A. Kurmus, SMoTherSpectre: Exploiting speculative execution through port contention. In Proc. CCS 2019.

[ES&P21] A. Mambretti, A. Sandulescu, A. Sorniotti, E. Kirda, W. Robertson, and A. Kurmus, Bypassing memory safety mechanisms through speculative control flow hijacks. To appear at Euro S&P'21. Available through https://arxiv.org/abs/2003.05503, 2020.

[SANER21] A. Mambretti, P. Convertini, A. Sorniotti, A. Sandulescu, E. Kirda, and A. Kurmus. GhostBuster: understanding and overcoming the pitfalls of transient execution vulnerability checkers. In Proc. IEEE SANER 2021