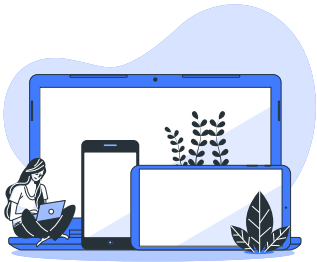
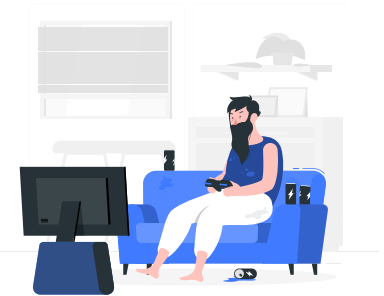
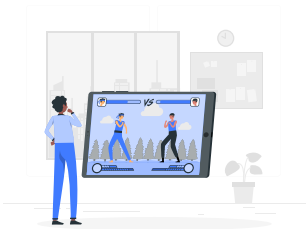
Hardening the Kernel Against Unprivileged Attacks

Claudio Canella
Graz University of Technology

November 9, 2022

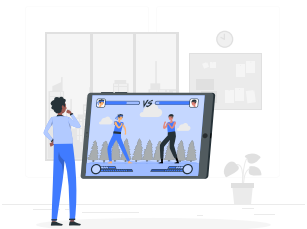





VS

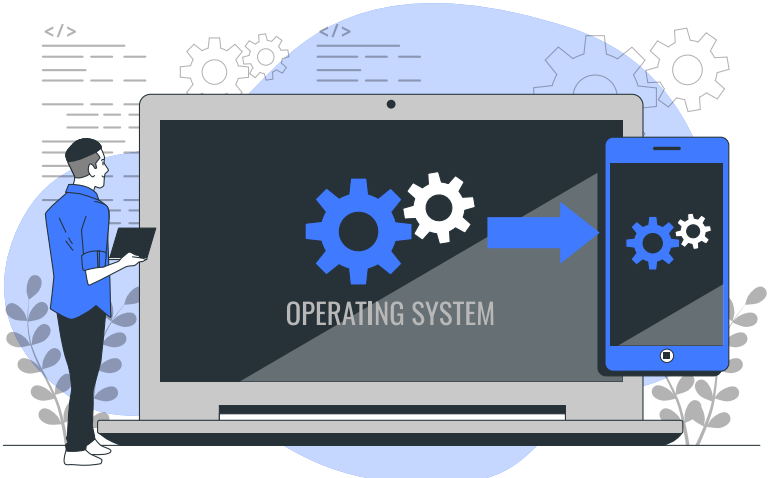

VS


Be


VS


Be


My Wallet
BANK
PAY


OPERATING SYSTEM

A Systematic Evaluation of Transient Execution Attacks and Defenses

Claudio Canella
Graz University of Technology

Jo Van Bulck
imec-DistriNet, KU Leuven

Michael Schwarz
Graz University of Technology

Moritz Lipp
Graz University of Technology

Benjamin von Berg
Graz University of Technology

Philipp Ortner
Graz University of Technology

Frank Piessens
imec-DistriNet, KU Leuven

Dmitry Evtyushkin
College of William and Mary

Daniel Gruss
Graz University of Technology

Transient Execution

CPUs optimize performance through

Transient Execution

CPUs optimize performance through

- out-of-order execution

Transient Execution

CPUs optimize performance through

- out-of-order execution
- speculative execution

Transient Execution

CPUs optimize performance through

- out-of-order execution
- speculative execution

What if CPU was wrong?

Transient Execution

CPUs optimize performance through

- out-of-order execution
- speculative execution

What if CPU was wrong?

$\rightarrow$ Roll back to mistake

Transient Execution

CPUs optimize performance through

- out-of-order execution
- speculative execution

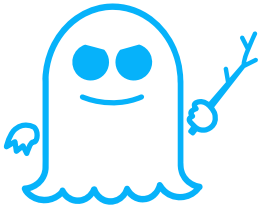
What if CPU was wrong?

$\rightarrow$ Roll back to mistake

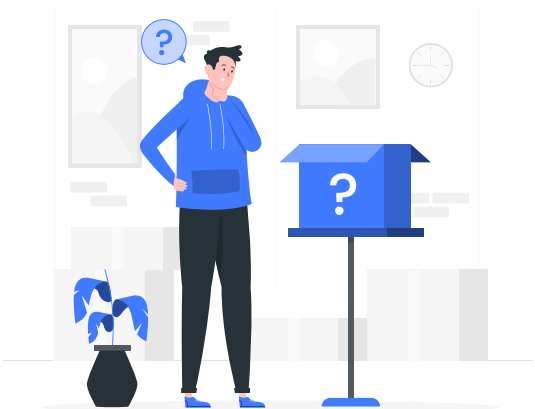
Squashed instructions are called transient

Transient-Execution Attacks

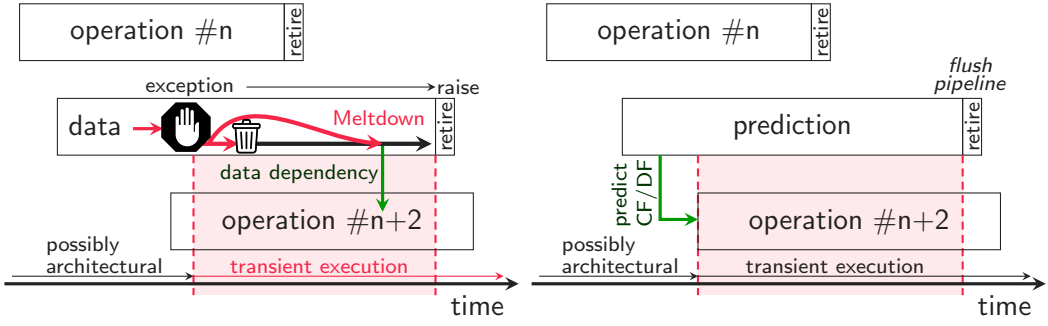
MELTDOWN



SPECTRE

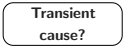


Root Cause Analysis

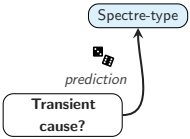


Meltdown **Spectre**

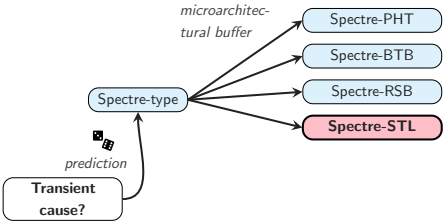
Spectre Tree



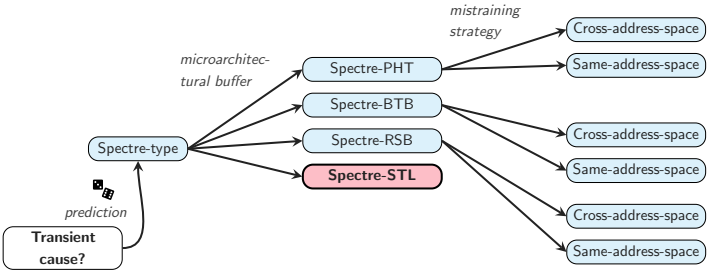
Spectre Tree



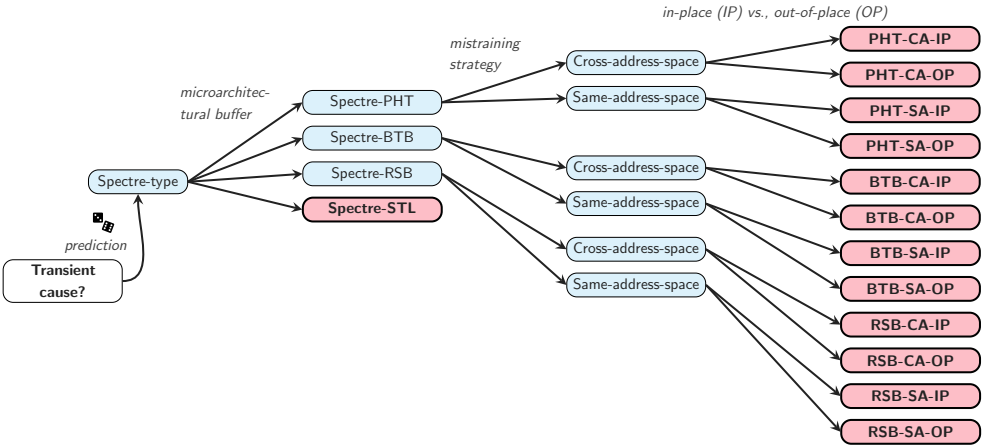
Spectre Tree



Spectre Tree



Spectre Tree



Spectre: Defense Analysis

Attack		InvisiSpec	SafeSpec	DAWG	RSB Stuffing	Retpoline	Poison Value	Index Masking	Site Isolation	SLH	YSNB	IBRS	STIPB	IBPB	Serialization	Taint Tracking	Timer Reduction	Sloth	SSBD/SSBB
Intel	Spectre-PHT	□	□	□	◇	◇	●	◐	◐	●	○	◇	◇	◇	◐	■	◐	◧	◇
	Spectre-BTB	□	□	□	◇	●	◇	◇	◐	◇	◇	●	◐	◐	◇	■	◐	◇	◇
	Spectre-RSB	□	□	□	◐	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	◇	◇
	Spectre-STL	□	□	□	◇	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	■	●
ARM	Spectre-PHT	□	□	□	◇	◇	●	◐	◐	●	○	◇	◇	◇	◐	■	◐	◧	◇
	Spectre-BTB	□	□	□	◇	●	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	◇	◇
	Spectre-RSB	□	□	□	◐	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	◇	◇
	Spectre-STL	□	□	□	◇	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	■	●
AMD	Spectre-PHT	□	□	□	◇	◇	●	◐	◐	●	○	◇	◇	◇	◐	■	◐	◧	◇
	Spectre-BTB	□	□	□	◇	●	◇	◇	◐	◇	◇	■	◧	◧	◇	■	◐	◇	◇
	Spectre-RSB	□	□	□	◐	◇	◇	◇	◐	◇	◇	◇	◇	◧	◇	■	◐	◇	◇
	Spectre-STL	□	□	□	◇	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	■	●

Symbols show if an attack is mitigated (●), partially mitigated (◐), not mitigated (○), theoretically mitigated (■), theoretically impeded (◧), not theoretically impeded (□), or out of scope (◇).

More Details

More details in the paper

- New Meltdown variants
- Spectre mistraining strategies
- Defense categorization and performance analysis
- Gadget classification and prevalence analysis
- ...

[Can+19b]

Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, Daniel Gruss.
A Systematic Evaluation of Transient Execution Attacks and Defenses.

KASLR: Break It, Fix It, Repeat

Claudio Canella
Graz University of Technology

Michael Schwarz
Graz University of Technology

Martin Haubenwallner
Graz University of Technology

Martin Schwarzl
Graz University of Technology

Daniel Gruss
Graz University of Technology

Hypothesis

Load is executed, returned value is zeroed out

Hypothesis

Load is executed, returned value is zeroed out

Meltdown Attack

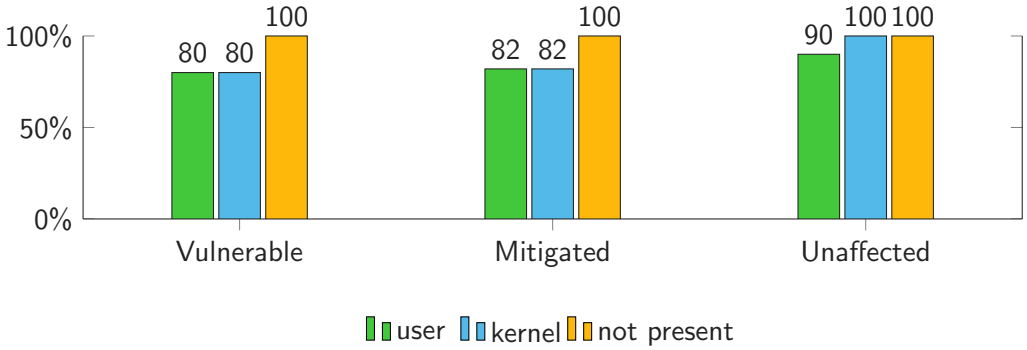
Hypothesis

Load is executed, returned value is zeroed out

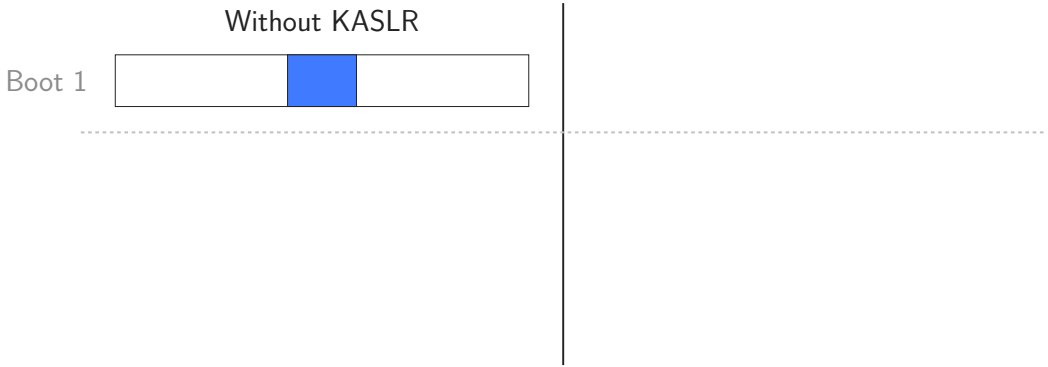
Meltdown Attack

Performance Counter

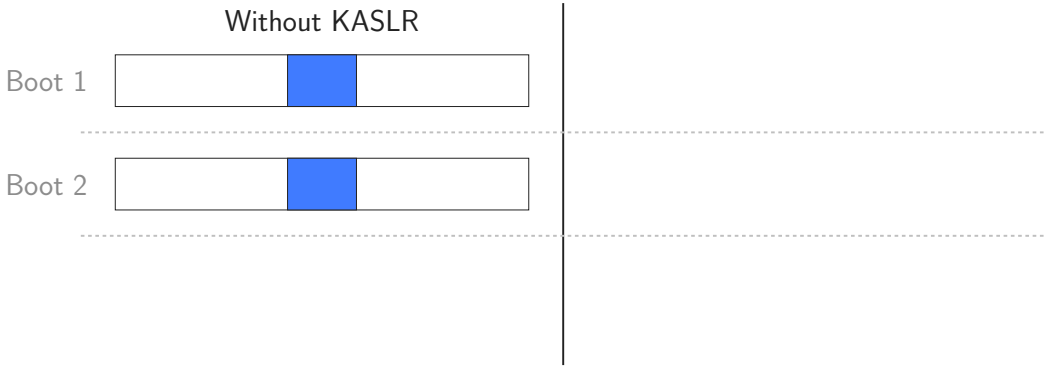
CPU Stalling Behavior



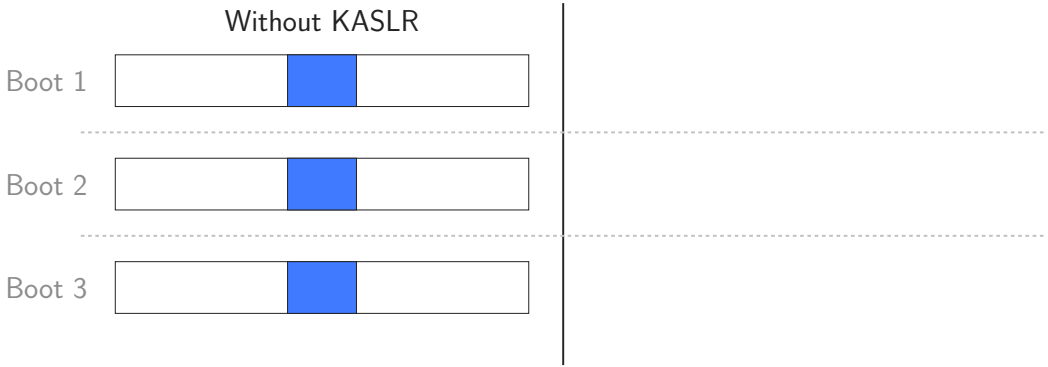
KASLR



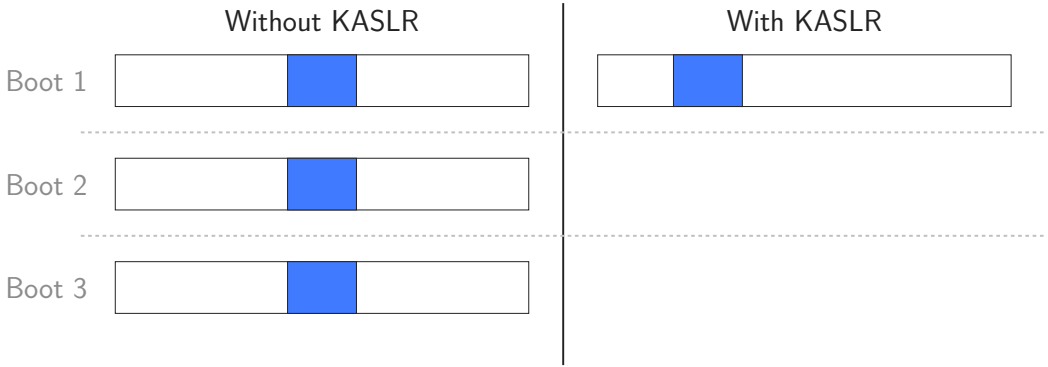
KASLR



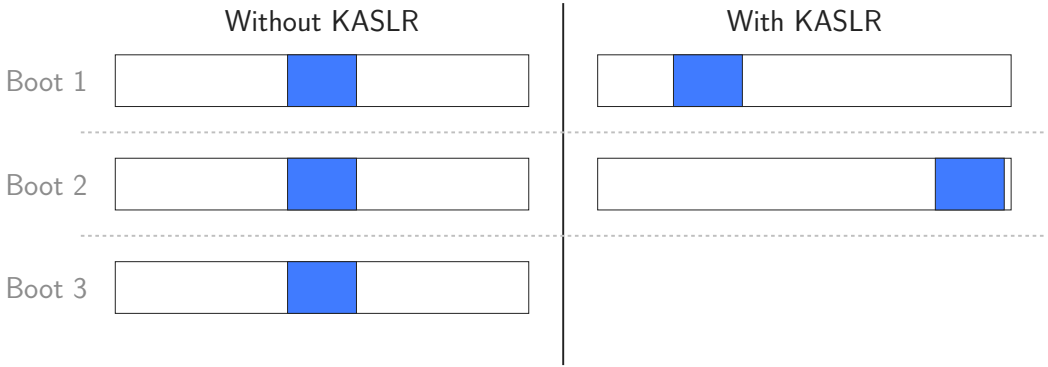
KASLR



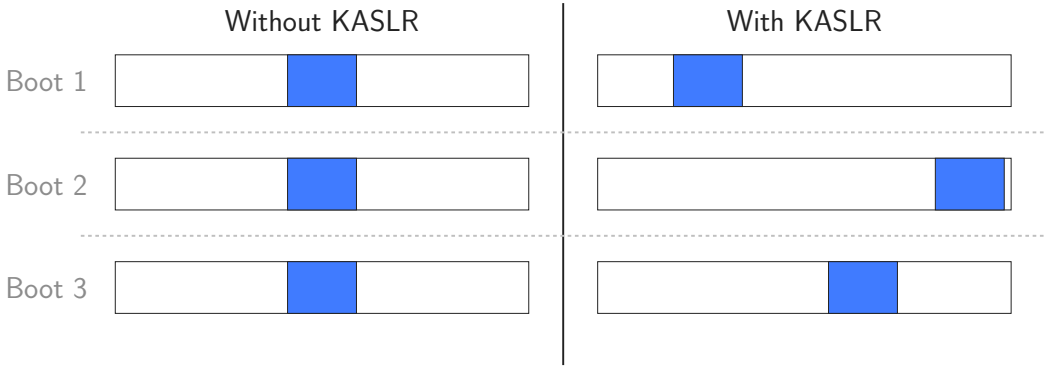
KASLR



KASLR



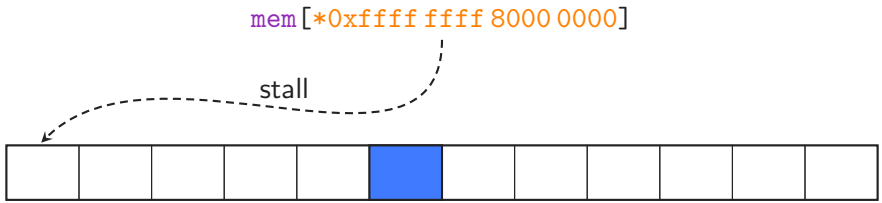
KASLR



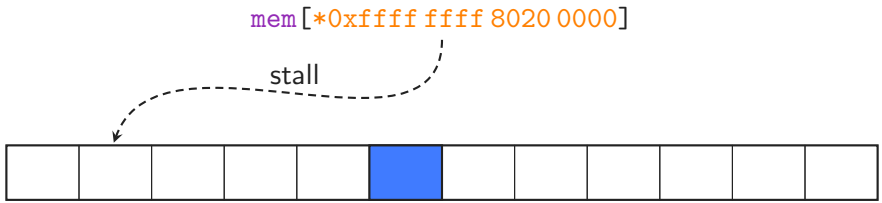
EchoLoad

`mem[*0xffff ffff 8000 0000]`

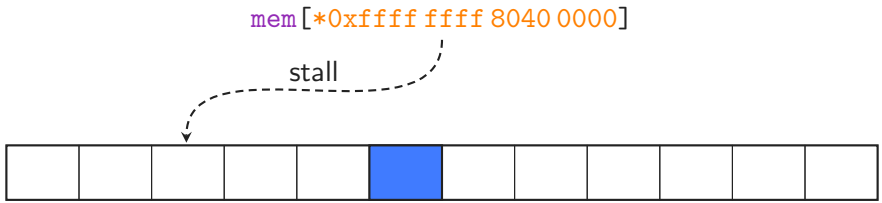
EchoLoad



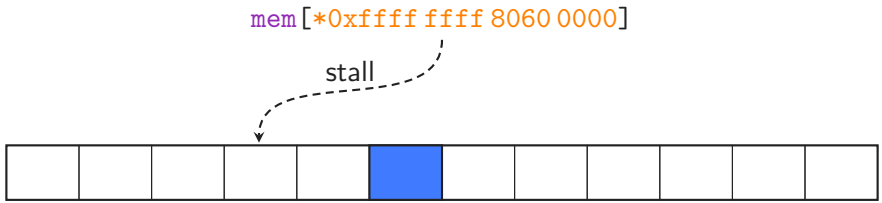
EchoLoad



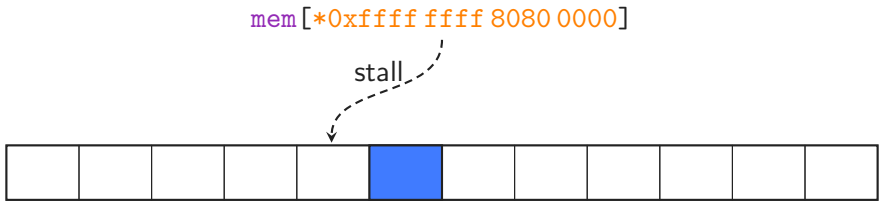
EchoLoad



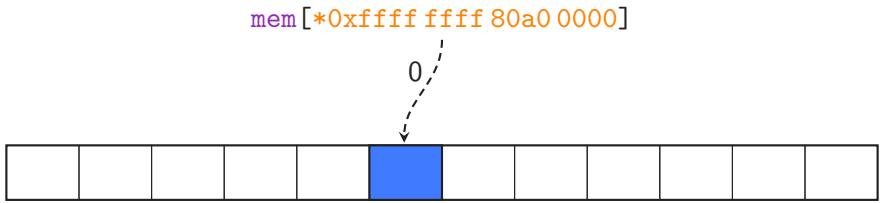
EchoLoad

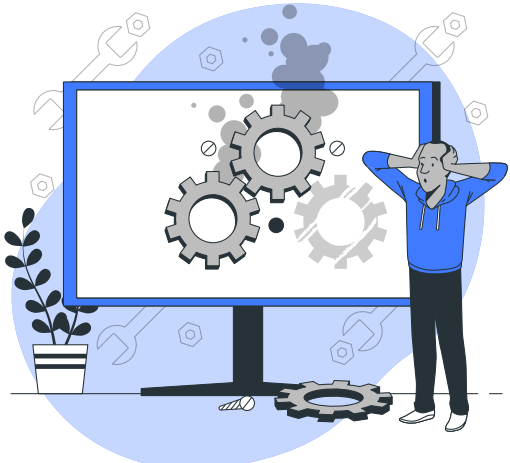


EchoLoad

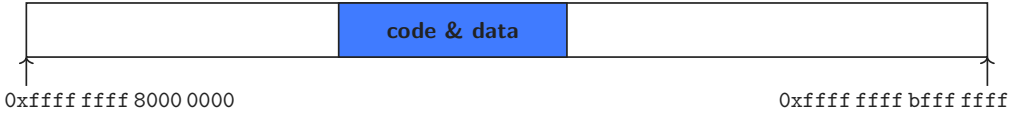


EchoLoad

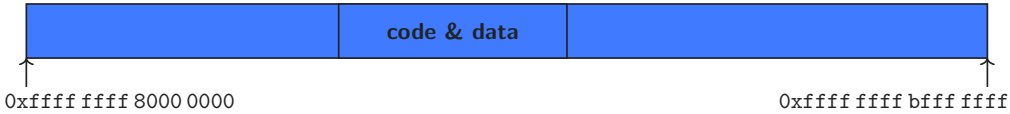




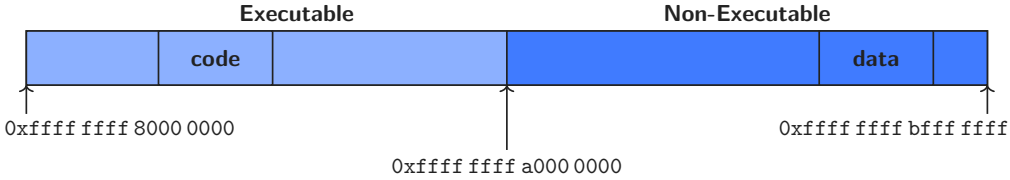
Fake Load Address REsponse (FLARE)



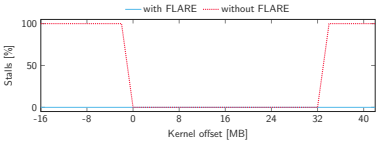
Fake Load Address REsponse (FLARE)



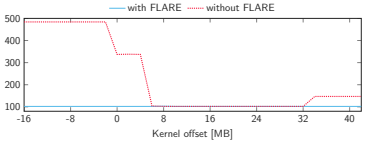
Fake Load Address REsponse (FLARE)



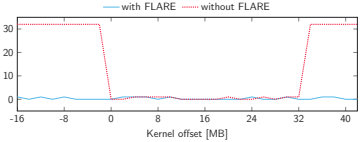
FLARE



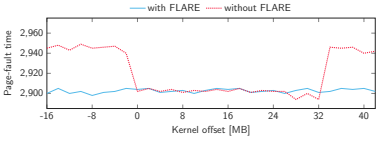
EchoLoad



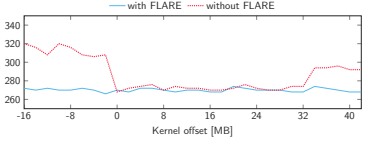
Prefetch [Gru+16]



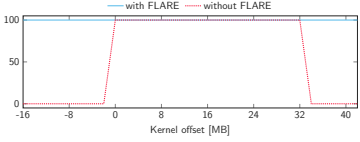
Data Bounce [Sch+19]



Double Page Fault [HWH13]



DrK [JLK16]



Fallout [Can+19a]

More Details

More details in the paper

- EchoLoad from and on SGX
- Meltdown in JavaScript on 32-bit Systems
- FLARE for Kernel modules, vmalloc, ...
- ...

[Can+20]

Claudio Canella, Michael Schwarz, Martin Haubenwallner, Martin Schwarzl, Daniel Gruss.
KASLR: Break It, Fix It, Repeat.

Linux Seccomp

Kernel Self-Protection Project

The “seccomp” system provides an opt-in feature made available to userspace

Kernel Self-Protection Project

The “seccomp” system provides an opt-in feature made available to userspace, which provides a way to reduce the number of kernel entry points available to a running process.

Kernel Self-Protection Project

The “seccomp” system provides an opt-in feature made available to userspace, which provides a way to reduce the number of kernel entry points available to a running process. This limits the breadth of kernel code that can be reached

Kernel Self-Protection Project

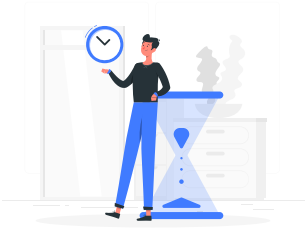
The "seccomp" system provides an opt-in feature made available to userspace, which provides a way to reduce the number of kernel entry points available to a running process. This limits the breadth of kernel code that can be reached, possibly reducing the availability of a given bug to an attack.

Seccomp Limitations

Effort

Seccomp Limitations

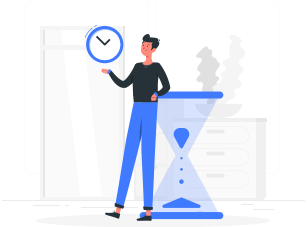
Effort



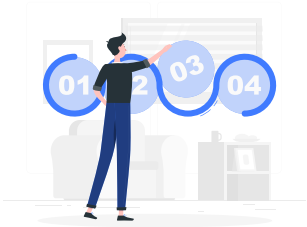
TOCTOU

Seccomp Limitations

Effort



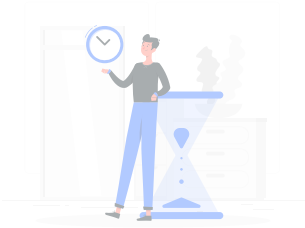
TOCTOU



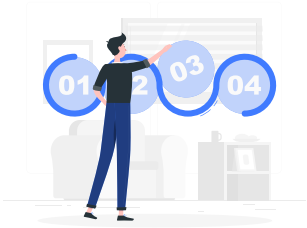
Stateless

Seccomp Limitations

Effort



TOCTOU



Stateless

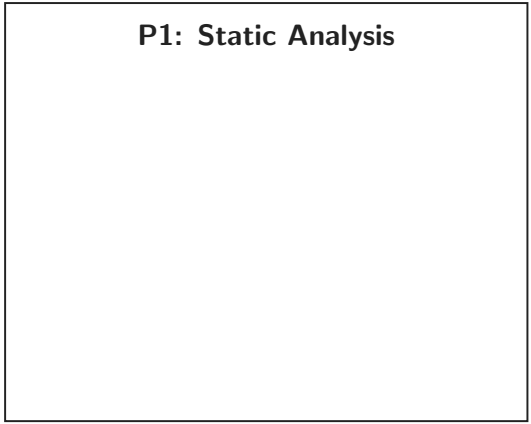
Automating Seccomp Filter Generation for Linux Applications

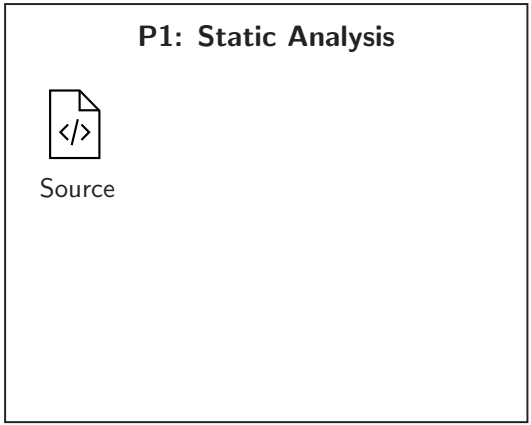
Claudio Canella
Graz University of Technology

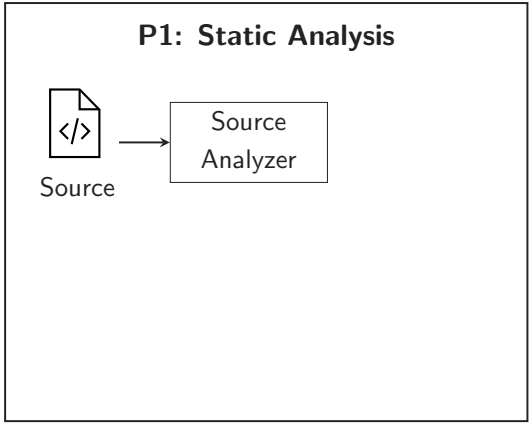
Mario Werner
Graz University of Technology

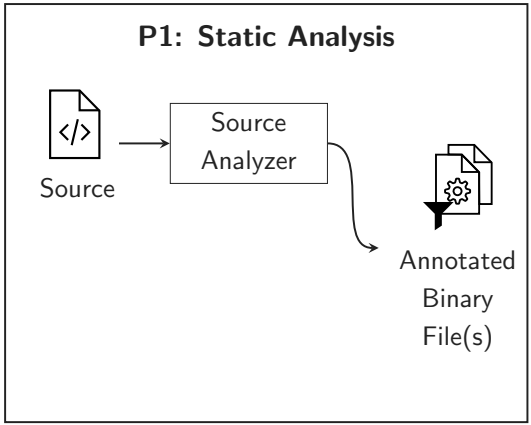
Daniel Gruss
Graz University of Technology

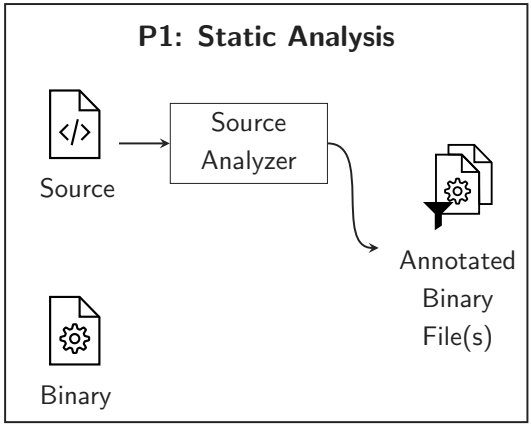
Michael Schwarz
CISPA Helmholtz Center for Information Security

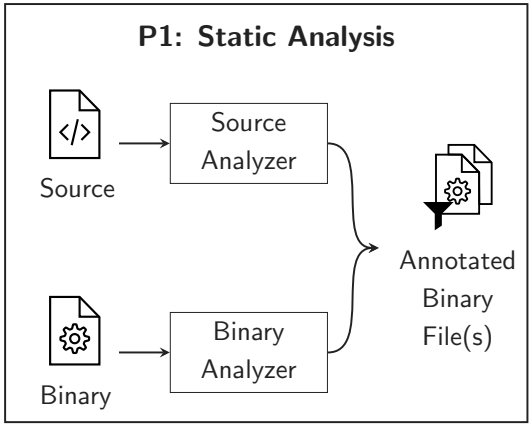

P1: Static Analysis

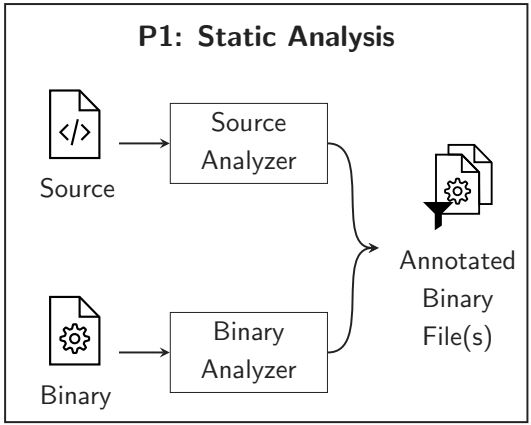

P1: Static Analysis
Source


P1: Static Analysis
Source
Source Analyzer

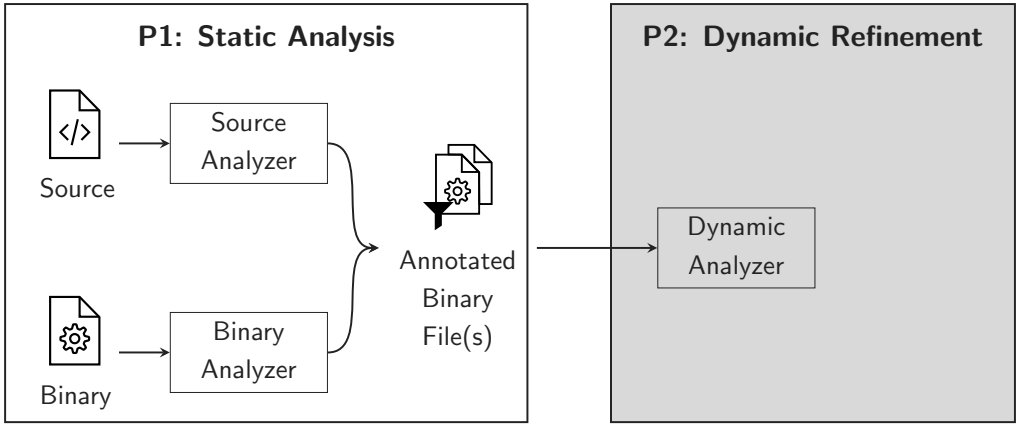

P1: Static Analysis
Source
Source Analyzer
Annotated Binary File(s)


P1: Static Analysis
Source
Source Analyzer
Annotated Binary File(s)
Binary

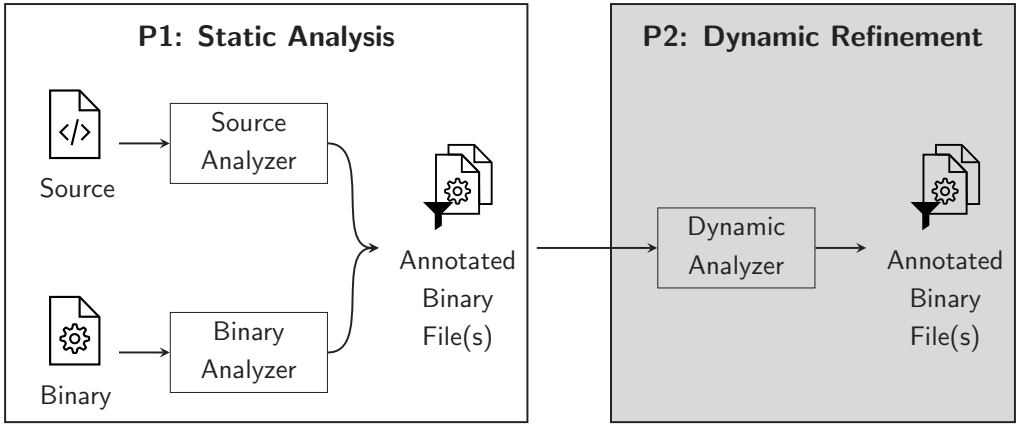

P1: Static Analysis
Source
Source Analyzer
Binary
Binary Analyzer
Annotated Binary File(s)



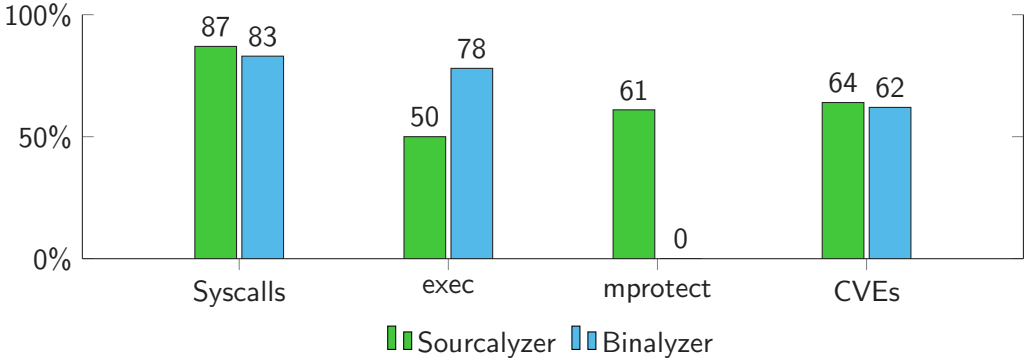
P2: Dynamic Refinement


P1: Static Analysis
Source
Source Analyzer
Binary
Binary Analyzer
Annotated Binary File(s)
P2: Dynamic Refinement
Dynamic Analyzer

Chestnut


P1: Static Analysis
Source
Source Analyzer
Binary
Binary Analyzer
Annotated Binary File(s)
P2: Dynamic Refinement
Dynamic Analyzer
Annotated Binary File(s)

Security Evaluation



More Details

More details in the paper

- Implementation details
- Information on overapproximation
- Detailed evaluation
- ...

[Can+21b]

Claudio Canella, Mario Werner, Daniel Gruss, Michael Schwarz.
Automating Seccomp Filter Generation for Linux Applications.

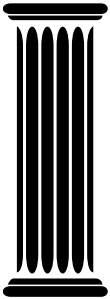
SFIP: Coarse-Grained Syscall-Flow-Integrity Protection in Modern Systems

Claudio Canella
Graz University of Technology

Sebastian Dorn
Graz University of Technology

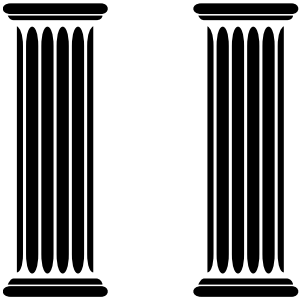
Daniel Gruss
Graz University of Technology

Michael Schwarz
CISPA Helmholtz Center for Information Security



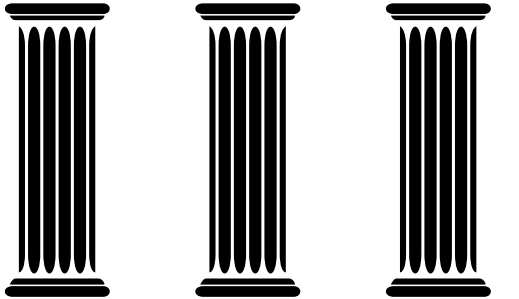
State Machine

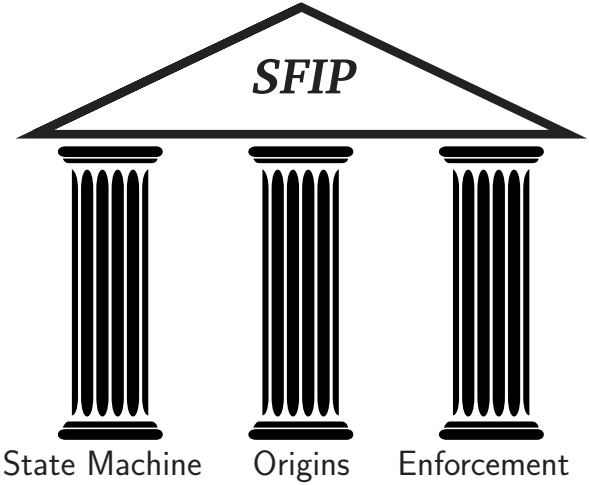
Syscall-Flow-Integrity Protection



State Machine Origins

Syscall-Flow-Integrity Protection




SFIP
State Machine
Origins
Enforcement

State Machine Analysis

Application	#States	Average Transitions
busybox	23.52	15.99
coreutils	26.64	16.66
pwgen	18	13.56
muraster	29	18.89
nginx	107	74.05
ffmpeg	55	49.07
memcached	86	43.16
mutool	53	32.26

State Machine Analysis

Application	#States	Average Transitions
busybox	23.52	15.99
coreutils	26.64	16.66
pwgen	18	13.56
muraster	29	18.89
nginx	107	74.05
ffmpeg	55	49.07
memcached	86	43.16
mutool	53	32.26

Origin Analysis

Application	Total #Offsets	Avg #Offsets
busybox	102.64	3.75
coreutils	116.71	4.42
pwgen	84	4.42
muraster	193	4.6
nginx	318	3.0
ffmpeg	279	4.98
memcached	317	3.69
mutool	278	4.15

Origin Analysis

Application	Total #Offsets	Avg #Offsets
busybox	102.64	3.75
coreutils	116.71	4.42
pwgen	84	4.42
muraster	193	4.6
nginx	318	3.0
ffmpeg	279	4.98
memcached	317	3.69
mutool	278	4.15

Return-Oriented Programming

Return-Oriented Programming

- uses exisiting code to exploit a program

Return-Oriented Programming

Return-Oriented Programming

- uses exisiting code to exploit a program
- jumps to parts of functions (gadgets)

Return-Oriented Programming

- uses exisiting code to exploit a program
- jumps to parts of functions (gadgets)
- chains gadgets together for an exploit

Return-Oriented Programming

Return-Oriented Programming

- uses exisiting code to exploit a program
- jumps to parts of functions (gadgets)
- chains gadgets together for an exploit

SFIP restricts ROP chains via

Return-Oriented Programming

Return-Oriented Programming

- uses exisiting code to exploit a program
- jumps to parts of functions (gadgets)
- chains gadgets together for an exploit

SFIP restricts ROP chains via

- syscall transitions $\rightarrow$ not every sequence is possible

Return-Oriented Programming

Return-Oriented Programming

- uses exisiting code to exploit a program
- jumps to parts of functions (gadgets)
- chains gadgets together for an exploit

SFIP restricts ROP chains via

- syscall transitions $\rightarrow$ not every sequence is possible
- syscall origins $\rightarrow$ not every location is valid

Return-Oriented Programming

Return-Oriented Programming

- uses exisiting code to exploit a program
- jumps to parts of functions (gadgets)
- chains gadgets together for an exploit

SFIP restricts ROP chains via

- syscall transitions $\rightarrow$ not every sequence is possible
- syscall origins $\rightarrow$ not every location is valid

Conclusion

SFIP imposes more significant constraints on control-flow-hijacking attacks than seccomp

More Details

More details in the paper

- Implementation details
- Extensive security discussion
- Mimicry attacks
- ...

[Can+22]

Claudio Canella, Sebastian Dorn, Daniel Gruss, Michael Schwarz.
SFIP: Coarse-Grained Syscall-Flow-Integrity Protection in Modern Systems.

Contributions

- Deepened understanding of transient-execution attacks

Contributions

- Deepened understanding of transient-execution attacks
  - $\rightarrow$ Found new attack variants

Contributions

- Deepened understanding of transient-execution attacks
  - → Found new attack variants
  - → Proposed widely-adopted naming scheme

Contributions

- Deepened understanding of transient-execution attacks
  - → Found new attack variants
  - → Proposed widely-adopted naming scheme
  - → Highlighted insufficient defenses

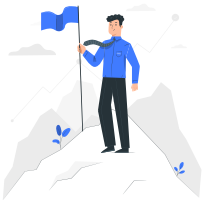
Contributions

- Deepened understanding of transient-execution attacks
  - → Found new attack variants
  - → Proposed widely-adopted naming scheme
  - → Highlighted insufficient defenses
- Hardened kernel against microarchitectural KASLR breaks

Contributions

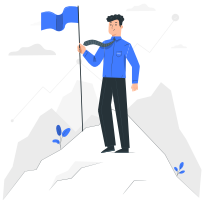
- Deepened understanding of transient-execution attacks
  - → Found new attack variants
  - → Proposed widely-adopted naming scheme
  - → Highlighted insufficient defenses
- Hardened kernel against microarchitectural KASLR breaks
- Reduced the exposed attack surface of the kernel

Contributions



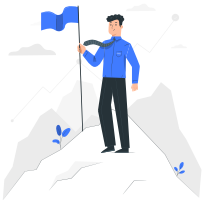
- Deepened understanding of transient-execution attacks
  - → Found new attack variants
  - → Proposed widely-adopted naming scheme
  - → Highlighted insufficient defenses
- Hardened kernel against microarchitectural KASLR breaks
- Reduced the exposed attack surface of the kernel
  - → Automated Seccomp

Contributions



- Deepened understanding of transient-execution attacks
  - → Found new attack variants
  - → Proposed widely-adopted naming scheme
  - → Highlighted insufficient defenses
- Hardened kernel against microarchitectural KASLR breaks
- Reduced the exposed attack surface of the kernel
  - → Automated Seccomp
  - → Syscall-Flow-Integrity Protection

Contributions



- Deepened understanding of transient-execution attacks
  - → Found new attack variants
  - → Proposed widely-adopted naming scheme
  - → Highlighted insufficient defenses
- Hardened kernel against microarchitectural KASLR breaks
- Reduced the exposed attack surface of the kernel
  - → Automated Seccomp
  - → Syscall-Flow-Integrity Protection
  - → Enabled complex argument checks

My PhD in Numbers

16

16

1

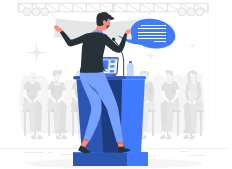
16

1

13

16

1

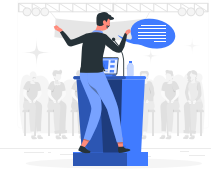


13

1

16

1



13

1

14

Hardening the Kernel Against Unprivileged Attacks

Claudio Canella
Graz University of Technology

November 9, 2022

References

[Can+19a] C. Canella, D. Genkin, L. Giner, D. Gruss, M. Lipp, M. Minkin, D. Moghimi, F. Piessens, M. Schwarz, B. Sunar, J. Van Bulck, and Y. Yarom. Fallout: Leaking Data on Meltdown-resistant CPUs. In: CCS. 2019.

[Can+19b] C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss. A Systematic Evaluation of Transient Execution Attacks and Defenses. In: USENIX Security Symposium. Extended classification tree and PoCs at https://transient.fail/. 2019.

[Can+20] C. Canella, M. Schwarz, M. Haubenwallner, M. Schwarzl, and D. Gruss. KASLR: Break It, Fix It, Repeat. In: AsiaCCS. 2020.

[Can+21a] C. Canella, A. Kogler, L. Giner, D. Gruss, and M. Schwarz. Domain Page-Table Isolation. In: arXiv:2111.10876 (2021).

[Can+21b] C. Canella, M. Werner, D. Gruss, and M. Schwarz. Automating Seccomp Filter Generation for Linux Applications. In: CCSW. 2021.

[Can+22] C. Canella, S. Dorn, D. Gruss, and M. Schwarz. SFIP: Coarse-Grained Syscall-Flow-Integrity Protection in Modern Systems. In: arXiv:2202.13716 (2022).

[Gru+16] D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard. Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR. In: CCS. 2016.

[HWH13] R. Hund, C. Willems, and T. Holz. Practical Timing Side Channel Attacks against Kernel Space ASLR. In: S&P. 2013.

[JLK16] Y. Jang, S. Lee, and T. Kim. Breaking Kernel Address Space Layout Randomization with Intel TSX. In: CCS. 2016.

[Sch+19] M. Schwarz, C. Canella, L. Giner, and D. Gruss. Store-to-Leak Forwarding: Leaking Data on Meltdown-resistant CPUs. In: arXiv:1905.05725 (2019).