

Beyond the Edges of Kernel Control-Flow Hijacking Protection with HEK-CFI

Lukas Maar, Pascal Nasahl, Stefan Mangard

5 Juli 2024

Contributions



- HEK-CFI: protection against kernel control-flow hijacking
- Proof-of-concept implementation
- Performance evaluation on Ubuntu 22.04 with an 1.85 % geomean overhead
- Security evaluation and comparison to other solutions

Motivation




Exploitation



Exploitation

🚩 Goals of adversaries






- Leaking sensitive informations, e.g., , , or 
- Resource compromising
- ...

Exploitation



🚩 Goals of adversaries

- Leaking sensitive informations, e.g., , , or 
- Resource compromising
- ...






Kernel security

- Isolate different entities

Exploitation



Goals of adversaries

- Leaking sensitive informations, e.g., , , or 
- Resource compromising
- ...

Kernel security

- Isolate different entities

Kernel vulnerabilities

- Exploitation to bypass isolation primitives

CVEs in the Linux Kernel

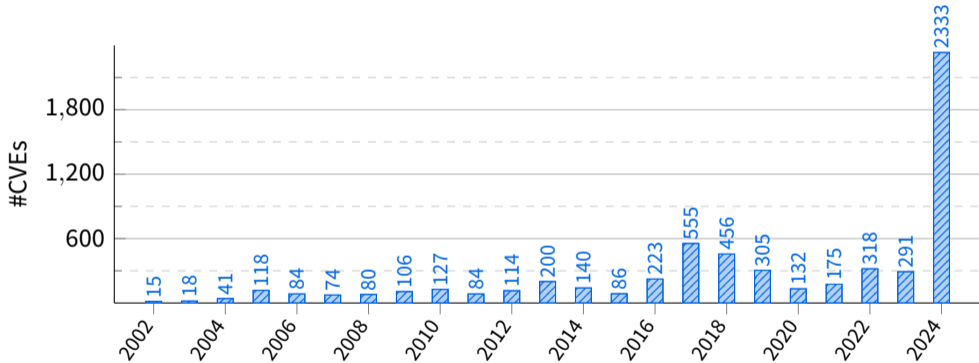


Figure: Found Linux kernel CVEs from NIST NVD.

Kernel Attacks



Kernel Attacks



Control-flow hijacking attacks

- Corrupt control data to redirect control flow
- Code execution → escalate privileges
- Popular, i.e., 15 out of 16 kernel exploits reported to Google's kernel bug bounty program [Goo22]

Kernel Attacks



Control-flow hijacking attacks

- Corrupt control data to redirect control flow
- Code execution → escalate privileges
- Popular, i.e., 15 out of 16 kernel exploits reported to Google's kernel bug bounty program [Goo22]



Kernel Control-Flow Integrity (CFI) [CDA14, And22, ABEL05]

- Restricts the control flow to the Control-Flow Graph (CFG)
- E.g., Android ensures with function-signature granularity [And22]

Motivational Example: CVE-2022-42703

- CVE-2022-42703 [Set22] presents novel exploitation technique
 - Manipulates thread state for redirecting control flow



Motivational Example: CVE-2022-42703

- CVE-2022-42703 [Set22] presents novel exploitation technique
 - Manipulates thread state for redirecting control flow



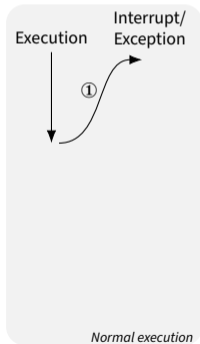
Execution



Normal execution

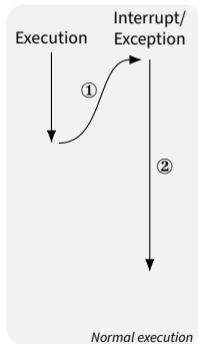
Motivational Example: CVE-2022-42703

- CVE-2022-42703 [Set22] presents novel exploitation technique
 - Manipulates thread state for redirecting control flow



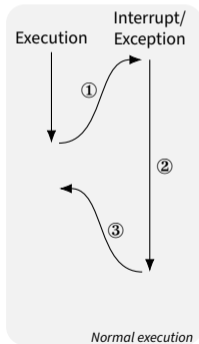
Motivational Example: CVE-2022-42703

- CVE-2022-42703 [Set22] presents novel exploitation technique
 - Manipulates thread state for redirecting control flow



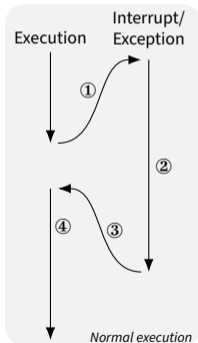
Motivational Example: CVE-2022-42703

- CVE-2022-42703 [Set22] presents novel exploitation technique
 - Manipulates thread state for redirecting control flow



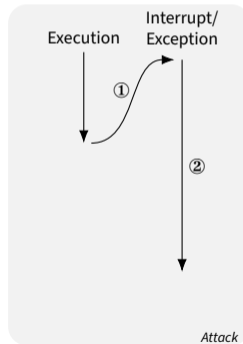
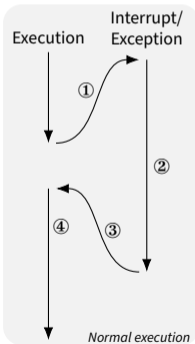
Motivational Example: CVE-2022-42703

- CVE-2022-42703 [Set22] presents novel exploitation technique
 - Manipulates thread state for redirecting control flow



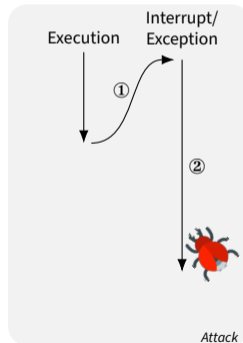
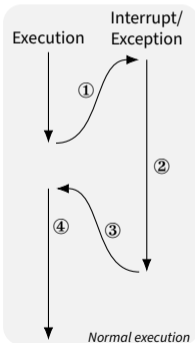
Motivational Example: CVE-2022-42703

- **CVE-2022-42703** [Set22] presents novel exploitation technique
 - Manipulates thread state for redirecting control flow



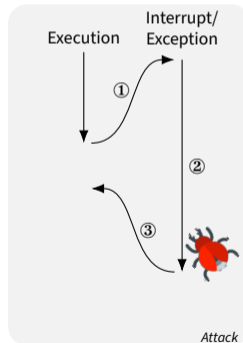
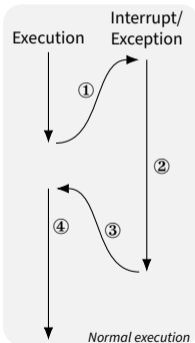
Motivational Example: CVE-2022-42703

- CVE-2022-42703 [Set22] presents novel exploitation technique
 - Manipulates thread state for redirecting control flow



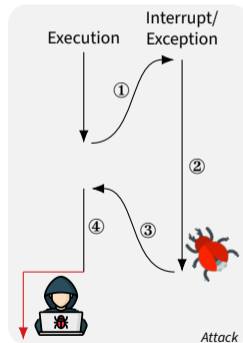
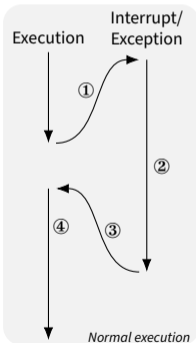
Motivational Example: CVE-2022-42703

- CVE-2022-42703 [Set22] presents novel exploitation technique
 - Manipulates thread state for redirecting control flow



Motivational Example: CVE-2022-42703

- CVE-2022-42703 [Set22] presents novel exploitation technique
 - Manipulates thread state for redirecting control flow



Protecting Control-Flow Related Data

Control-Flow Hijacking Attacks in the Kernel



- 🚩 Various control-flow related data allow to hijacking the control-flow

Control-Flow Hijacking Attacks in the Kernel



Various control-flow related data allow to hijacking the control-flow

- Function pointers

```
1 struct timerfd_ctx {  
2     ...  
3     enum hrtimer_restart (*function)(struct hrtimer *);  
4     struct hrtimer_clock_base *base;  
5     ...  
6 }
```


Control-Flow Hijacking Attacks in the Kernel

Various control-flow related data allow to hijacking the control-flow

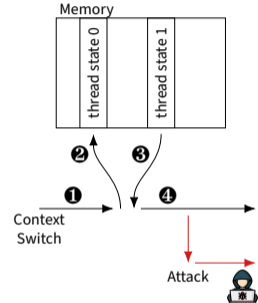
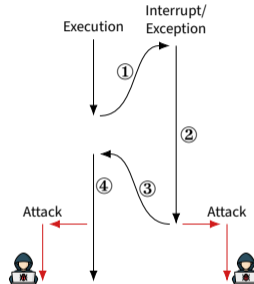
- Function pointers
- Operation table pointers

```
1 struct file_operations {
2     ...
3     ssize_t (*read)(struct file *, char *, size_t, loff_t *);
4     ssize_t (*write)(struct file *, const char *, size_t,
5                     loff_t *);
6     ssize_t (*read_iter)(struct kiocb *, struct iov_iter *);
7     ssize_t (*write_iter)(struct kiocb *, struct iov_iter *);
8     ...
9 };
10 struct file {
11     ...
12     const struct file_operations *f_op;
13     ...
14 };
```

Control-Flow Hijacking Attacks in the Kernel

Various control-flow related data allow to hijacking the control-flow

- Function pointers
- Operation table pointers
- Thread state



Control-Flow Hijacking Attacks in the Kernel

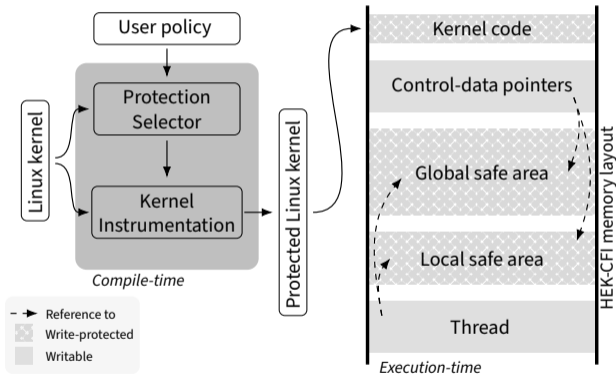


Various control-flow related data allow to hijacking the control-flow

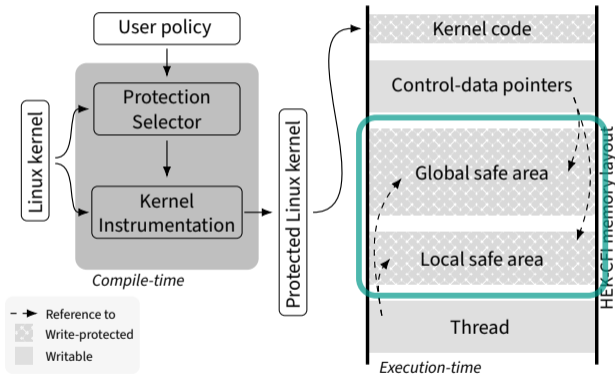
- Function pointers
- Operation table pointers
- Thread state
- Return addresses

```
1 dummy_fn:  
2 push %r14  
3 push %r13  
4 push %r12  
5 mov %rdi,%r12  
6 push %rbp  
7 sub $0x8,%rsp  
8 ...  
9 add $0x8,%rsp  
10 pop %rbp  
11 pop %r12  
12 pop %r13  
13 pop %r14  
14 ret
```

Hardware Enforced Kernel Control-Flow Integrity (HEK-CFI)



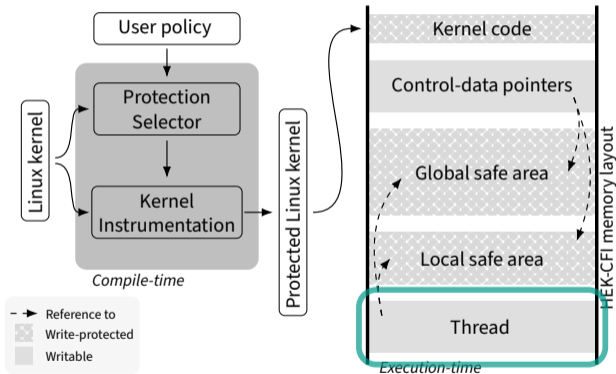
Hardware Enforced Kernel Control-Flow Integrity (HEK-CFI)



- Kernel control-data integrity

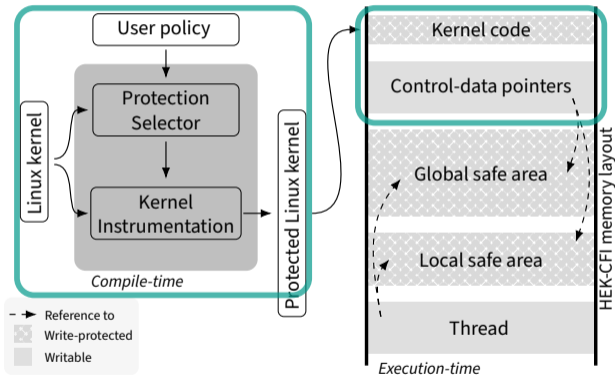
- Provides global/local safe areas
- Write-protected memory areas

Hardware Enforced Kernel Control-Flow Integrity (HEK-CFI)



- **Kernel control-data integrity**
 - Provides global/local safe areas
 - Write-protected memory areas
- **Thread state protection**
 - Protects thread state with control-data integrity
 - Protects return addresses

Hardware Enforced Kernel Control-Flow Integrity (HEK-CFI)

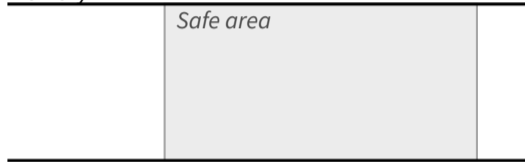


- **Kernel control-data integrity**
 - Provides global/local safe areas
 - Write-protected memory areas
- **Thread state protection**
 - Protects thread state with control-data integrity
 - Protects return addresses
- **Protection selector & instrumentation**
 - Protects valuable pointers with control-data integrity
 - Protects non-valuable pointers with signature-based CFI
 - Based on user policy

Write-Protected Memory

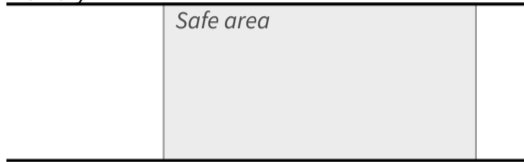


Memory



Write-Protected Memory

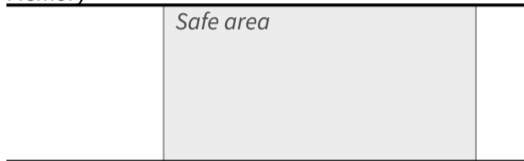
Memory



- Write-protected memory for safe areas

Write-Protected Memory

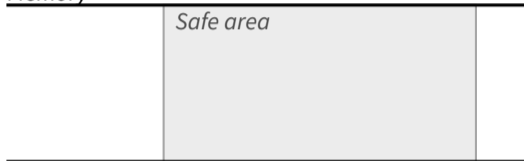
Memory



- Write-protected memory for safe areas
- **Primitive:** Intel CET SHSTK
 - Only certain instructions allowed to write to shadow pages [XWZ⁺22]
 - E.g., `call`, `ret`, `wrssl`, or `iret`

Write-Protected Memory

Memory




- Write-protected memory for safe areas
- **Primitive:** Intel CET SHSTK
 - Only certain instructions allowed to write to shadow pages [XWZ⁺22]
 - E.g., `call`, `ret`, `wrssl`, or `iret`
- **Idea:** Mark safe areas as shadow pages

Write-Protected Memory

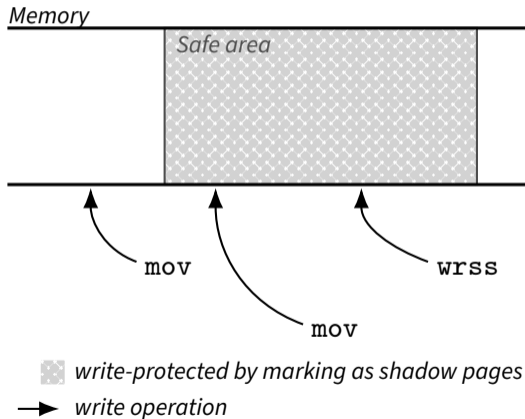
Memory



- Write-protected memory for safe areas
- **Primitive:** Intel CET SHSTK
 - Only certain instructions allowed to write to shadow pages [XWZ⁺22]
 - E.g., `call`, `ret`, `wrssl`, or `iret`
- **Idea:** Mark safe areas as shadow pages

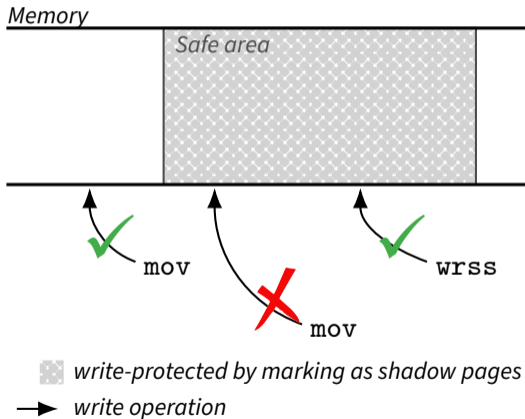
 *write-protected by marking as shadow pages*

Write-Protected Memory



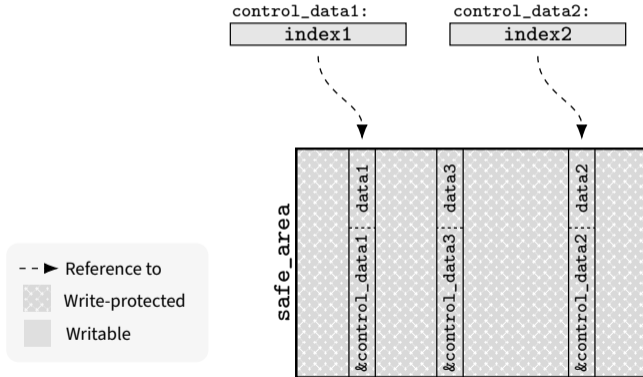
- Write-protected memory for safe areas
- **Primitive:** Intel CET SHSTK
 - Only certain instructions allowed to write to shadow pages [XWZ⁺22]
 - E.g., `call`, `ret`, `wrss`, or `iret`
- **Idea:** Mark safe areas as shadow pages
- Access write-protected memory
 - `wrss` for legal writes
 - Write operations (e.g., `mov`) causes exceptions

Write-Protected Memory

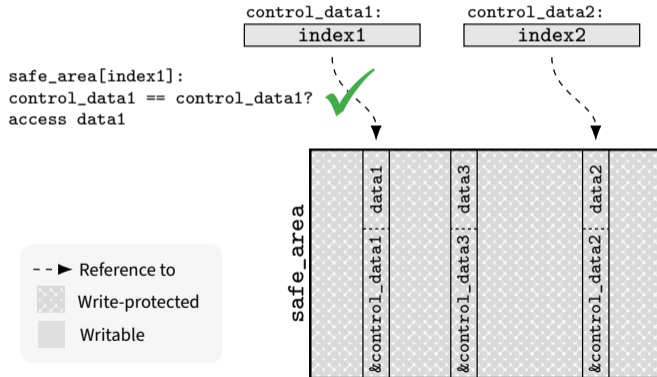


- Write-protected memory for safe areas
- **Primitive:** Intel CET SHSTK
 - Only certain instructions allowed to write to shadow pages [XWZ⁺22]
 - E.g., `call`, `ret`, `wrss`, or `iret`
- **Idea:** Mark safe areas as shadow pages
- Access write-protected memory
 - `wrss` for legal writes
 - Write operations (e.g., `mov`) causes exceptions

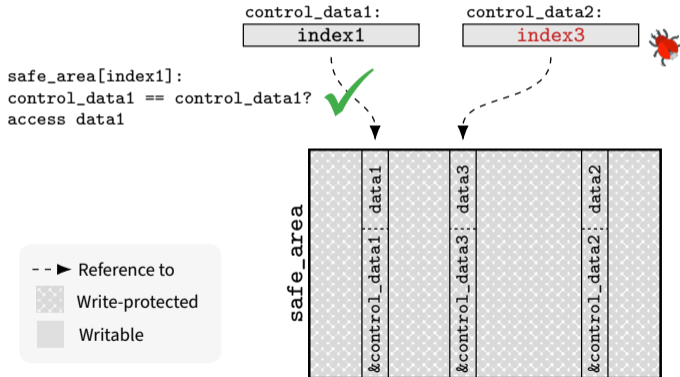
Global Safe Area



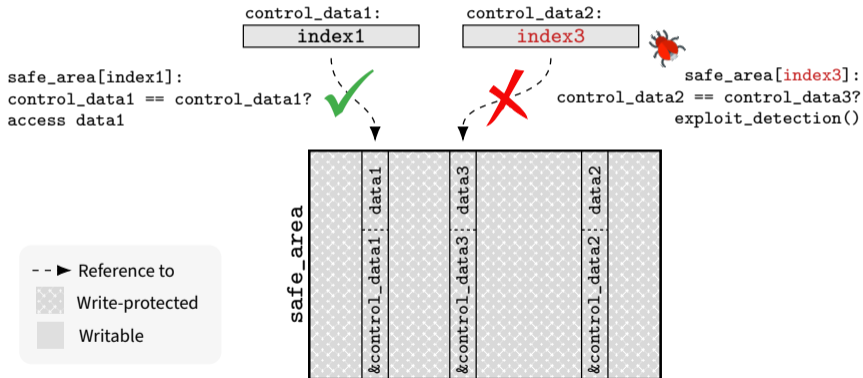
Global Safe Area



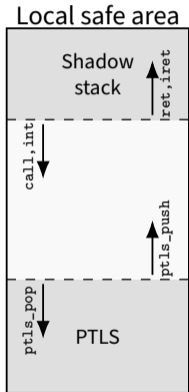
Global Safe Area



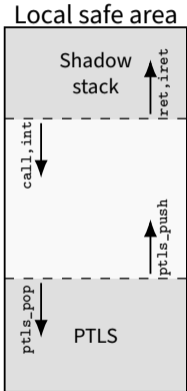
Global Safe Area



Local Safe Area

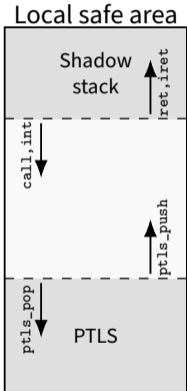


Local Safe Area



- Intel CET SHSTK
 - Use `ss` instructions to write to shadow stack
 - E.g., `call` pushes return addr
 - Does not provide `pushss` or `pullss`

Local Safe Area



- Intel CET SHSTK
 - Use `ss` instructions to write to shadow stack
 - E.g., `call` pushes return addr
 - Does not provide `pushss` or `pullss`
- Protected Thread Local Storage (PTLS)
 - Software solution using `wrss`
 - Provides `ptls_push/pull`
 - E.g., used to stored thread state during interrupts/exceptions

Implementation & Evaluation



Implementation & Evaluation



- We implemented Intel CET SHSTK for supervisor and a HEK-CFI proof-of-concept
 - As a compiler-assisted software framework
 - Linux kernel extension, a code analyzer and instrumentation

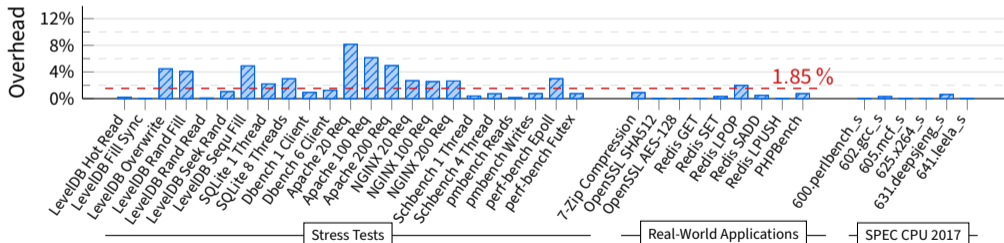
Implementation & Evaluation



- We implemented Intel CET SHSTK for supervisor and a HEK-CFI proof-of-concept
 - As a compiler-assisted software framework
 - Linux kernel extension, a code analyzer and instrumentation
- Evaluation on Ubuntu 22.04.1 LTS with a processor supporting Intel CET SHSTK

Implementation & Evaluation

- We implemented Intel CET SHSTK for supervisor and a HEK-CFI proof-of-concept
 - As a compiler-assisted software framework
 - Linux kernel extension, a code analyzer and instrumentation
- Evaluation on Ubuntu 22.04.1 LTS with a processor supporting Intel CET SHSTK



Comparison

Mitigations	Attack Vector			
	Thread state	Return addresses	Operation table pointers	Function pointers
Ge et al. [GTPJ16]	○	○	■	●
kCFI [And22]	○	○	□	○
Fine-CFI [LTZM18]	○	○	■	●
PATTER [YZS ⁺ 19]	○	●	○	●
Camouflage [DCLCE20]	○	●	●	○
PAL [SJS ⁺ 22]	○	●	○	●
FineIBT [Mor22]	○	○	□	○
KCoFI [CDA14]	●	○	□	○
Intel CET SHSTK [Int16]	○	●	○	○
CPI [KSP ⁺ 14] + CETIS [XWZ ⁺ 22]	○	●	○	●
HEK-CFI	●	●	●	●

- Protection ○ Insufficient protection
 ■ Implicit protection □ Implicit insufficient protection
 ○ Does not protect but can be extended.

Conclusion



Conclusion



- Kernel control-data integrity, including a secure approach to protect system events and return addresses

Conclusion



- Kernel control-data integrity, including a secure approach to protect system events and return addresses
- HEK-CFI that combines our kernel control-data integrity with signature-based CFI

Conclusion



- Kernel control-data integrity, including a secure approach to protect system events and return addresses
- HEK-CFI that combines our kernel control-data integrity with signature-based CFI
- Implemented Intel CET SHSTK and a HEK-CFI proof-of-concept

Conclusion



- Kernel control-data integrity, including a secure approach to protect system events and return addresses
- HEK-CFI that combines our kernel control-data integrity with signature-based CFI
- Implemented Intel CET SHSTK and a HEK-CFI proof-of-concept
- Performed a security and performance evaluation of HEK-CFI.

References I



- [ABEL05] Martín Abadi, Mihai Budiu, Ulfar Erlingsson, and Jay Ligatti, *Control-Flow Integrity*, CCS, 2005.
- [And22] Android, *Kernel Control Flow Integrity*, 2022.
- [CDA14] John Criswell, Nathan Dautenhahn, and Vikram Adve, *KCoFI: Complete Control-Flow Integrity for Commodity Operating System Kernels*, S&P, 2014.
- [DCLCE20] Rémi Denis-Courmont, Hans Liljestrand, Carlos Chinaea, and Jan-Erik Ekberg, *Camouflage: Hardware-assisted CFI for the ARM Linux kernel*, DAC, 2020.
- [Goo22] Google, *Kernel Exploits Recipes Notebook*, 2022.

References II



- [GTPJ16] Xinyang Ge, Nirupama Talele, Mathias Payer, and Trent Jaeger, *Fine-Grained Control-Flow Integrity for Kernel Software*, Euro S&P, 2016.
- [Int16] Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture*, 2016.
- [KSP⁺14] Volodymyr Kuznetsov, László Szekeres, Mathias Payer, George Candea, R Sekar, and Dawn Song, *Code-Pointer Integrity*, OSDI, 2014.
- [LTZM18] Jinku Li, Xiaomeng Tong, Fengwei Zhang, and Jianfeng Ma, *Fine-CFI: Fine-Grained Control-Flow Integrity for Operating System Kernels*, IEEE Transactions on Information Forensics and Security (2018).

References III



- [Mor22] Joao Moreira, *Kernel FineIBT Support*, 4 2022.
- [Set22] Seth Jenkins, *Exploiting CVE-2022-42703 - Bringing back the stack attack*, 2022.
- [SJS⁺22] Yoo Sungbae, Park Jinbum, Kim Seolheui, Kim Yeji, and Kim Taesoo, *In-Kernel Control-Flow Integrity on Commodity OSes using ARM Pointer Authentication*, USENIX Security, 2022.
- [XWZ⁺22] Mengyao Xie, Chenggang Wu, Yinqian Zhang, Jiali Xu, Yuanming Lai, Yan Kang, Wei Wang, and Zhe Wang, *CETIS: Retrofitting Intel CET for Generic and Efficient Intra-Process Memory Isolation*, CCS, 2022.

References IV



- [YZS⁺19] Yutian Yang, Songbo Zhu, Wenbo Shen, Yajin Zhou, Jiadong Sun, and Kui Ren, *ARM Pointer Authentication based Forward-Edge and Backward-Edge Control Flow Integrity for Kernels*, arXiv:1912.10666 (2019).