TU Graz

SCIENCE
PASSION
TECHNOLOGY

Artifacts Evaluated
acm
Functional V1.1

DOPE: DOmain Protection Enforcement with PKS

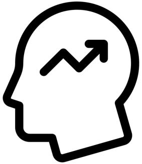
Lukas Maar, Martin Schwarzl, Fabian Rauscher, Daniel Gruss, Stefan Mangard

7 December 2023

› lukas.maar@iaik.tugraz.at

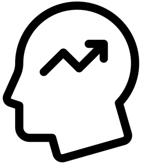
Motivation

Exploitation



- Goals of adversaries
  - Leaking sensitive informations, e.g., 🔒, 🔑, or 💳
  - Resource compromising
  - . . .
- Kernel security
  - Isolate different entities
- Kernel vulnerabilities
  - Exploitation to bypass isolation primitives

Exploitation



Goals of adversaries

- Leaking sensitive informations, e.g., 🔒, 🔑, or 💳
- Resource compromising
- ...

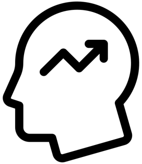
Kernel security

- Isolate different entities

Kernel vulnerabilities

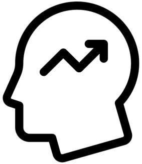
- Exploitation to bypass isolation primitives

Exploitation



- Goals of adversaries
  - Leaking sensitive informations, e.g., 🔒, 🔑, or 💳
  - Resource compromising
  - . . .
- Kernel security
  - Isolate different entities
- Kernel vulnerabilities
  - Exploitation to bypass isolation primitives

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

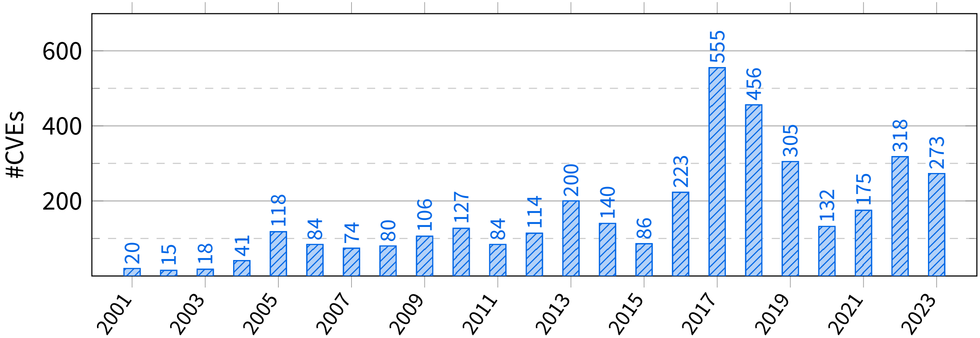


Figure: Found Linux kernel CVEs from NIST NVD.

Kernel Attacks

- Control-flow hijacking attacks
  - Corrupt control data to redirect control flow
  - ROP or JOP chain
  - Code execution → escalate privileges
- Kernel Control-Flow Integrity (CFI) [CDA14, Edg20, ABEL05] prevents control-flow hijacking attacks
- What about corrupting non-control data?

Kernel Attacks

Control-flow hijacking attacks

- Corrupt control data to redirect control flow
- ROP or JOP chain
- Code execution → escalate privileges

Kernel Control-Flow Integrity (CFI) [CDA14, Edg20, ABEL05] prevents control-flow hijacking attacks

What about corrupting non-control data?

Kernel Attacks

Control-flow hijacking attacks

- Corrupt control data to redirect control flow
- ROP or JOP chain
- Code execution → escalate privileges

Kernel Control-Flow Integrity (CFI) [CDA14, Edg20, ABEL05] prevents control-flow hijacking attacks

What about corrupting non-control data?

Kernel Attacks

- Control-flow hijacking attacks
  - Corrupt control data to redirect control flow
  - ROP or JOP chain
  - Code execution → escalate privileges
- Kernel Control-Flow Integrity (CFI) [CDA14, Edg20, ABEL05] prevents control-flow hijacking attacks
- What about corrupting non-control data?

Data-Oriented Attacks

Overview

Goal of adversaries to overwrite sensitive non-control data

Does not violate control flow's integrity

Sensitive data objects in the kernel

- Credentials
- Inode
- Page tables
- ...

Overview

Goal of adversaries to overwrite sensitive non-control data

Does not violate control flow's integrity

Sensitive data objects in the kernel

- Credentials
- Inode
- Page tables
- ...

Overview

- Goal of adversaries to overwrite sensitive non-control data
- Does not violate control flow's integrity
- Sensitive data objects in the kernel
  - Credentials
  - Inode
  - Page tables
  - . . .

Overview

- Goal of adversaries to overwrite sensitive non-control data
- Does not violate control flow’s integrity
- Sensitive data objects in the kernel
  - Credentials
  - Inode
  - Page tables
  - …

Overview

- Goal of adversaries to overwrite sensitive non-control data
- Does not violate control flow's integrity
- Sensitive data objects in the kernel
  - Credentials
  - Inode
  - Page tables
  - ...

```
struct cred {
  kuid_t uid;
  kgid_t gid;
  ...
  kernel_cap_t cap_permitted;
  kernel_cap_t cap_effective;
  ...
  struct key *thread_keyring;
  ...
  struct user_namespace *user_ns;
  ...
} __randomize_layout;
```

Overview

- Goal of adversaries to overwrite sensitive non-control data
- Does not violate control flow's integrity
- Sensitive data objects in the kernel
  - Credentials
  - Inode
  - Page tables
  - ...

```
struct inode {
  umode_t i_mode;
  kuid_t i_uid;
  kgid_t i_gid;
  unsigned int i_flags;
  ...
} __randomize_layout;
```

Overview

- Goal of adversaries to overwrite sensitive non-control data
- Does not violate control flow's integrity
- Sensitive data objects in the kernel
  - Credentials
  - Inode
  - Page tables
  - ...

```
#define _PAGE_BIT_PRESENT 0
#define _PAGE_BIT_RW 1
#define _PAGE_BIT_USER 2
...
#define _PAGE_BIT_PAT_LARGE 12
...
#define _PAGE_BIT_NX 63
```

Data-Oriented Attacks in the Wild

- Data-oriented attacks are very common
  - DirtyCred [LWX22], Dirty PageTable [Nic23], …
  - Numerous public exploits and one-day attacks [Goo19, Goo21, Ale21]
  - Enormous threat to system security

RQ1: How can we enhance kernel security to provide effective protection against data-oriented attacks with reasonable performance overhead for multiple sensitive data objects?

RQ2: How does our solution scale and perform when compared to state-of-the-art solutions?

Data-Oriented Attacks in the Wild

Data-oriented attacks are very common

- DirtyCred [LWX22], Dirty PageTable [Nic23], . . .
- Numerous public exploits and one-day attacks [Goo19, Goo21, Ale21]
- Enormous threat to system security

RQ1: How can we enhance kernel security to provide effective protection against data-oriented attacks with reasonable performance overhead for multiple sensitive data objects?

RQ2: How does our solution scale and perform when compared to state-of-the-art solutions?

Data-Oriented Attacks in the Wild

Data-oriented attacks are very common

- DirtyCred [LWX22], Dirty PageTable [Nic23], . . .
- Numerous public exploits and one-day attacks [Goo19, Goo21, Ale21]
- Enormous threat to system security

RQ1: *How can we enhance kernel security to provide effective protection against data-oriented attacks with reasonable performance overhead for multiple sensitive data objects?*

RQ2: *How does our solution scale and perform when compared to state-of-the-art solutions?*

Data-Oriented Attacks in the Wild

Data-oriented attacks are very common

- DirtyCred [LWX22], Dirty PageTable [Nic23], . . .
- Numerous public exploits and one-day attacks [Goo19, Goo21, Ale21]
- Enormous threat to system security

RQ1: *How can we enhance kernel security to provide effective protection against data-oriented attacks with reasonable performance overhead for multiple sensitive data objects?*

RQ2: *How does our solution scale and perform when compared to state-of-the-art solutions?*

Sensitive Data Protection

DOPE: DOmain Protection Enforcement with PKS

★ Novel kernel mitigation to protect sensitive data objects

Enforces domain protection leveraging Intel PKS [Int16]

- Moves sensitive data to distinct security domains
- Restricts memory access to these domains
- Based on the principle of least privilege

Protects 8 sensitive data objects with an average runtime overhead of ≈2.3 %

Systematically analyze 11 state-of-the-art data protection schemes

… DOPE significantly improves in terms of security to performance

DOPE: DOmain Protection Enforcement with PKS

★ Novel kernel mitigation to protect sensitive data objects

Enforces domain protection leveraging Intel PKS [Int16]

- Moves sensitive data to distinct security domains
- Restricts memory access to these domains
- Based on the principle of least privilege

Protects 8 sensitive data objects with an average runtime overhead of ≈2.3 %

Systematically analyze 11 state-of-the-art data protection schemes

… DOPE significantly improves in terms of security to performance

DOPE: DOmain Protection Enforcement with PKS

★ Novel kernel mitigation to protect sensitive data objects

🛡 Enforces domain protection leveraging Intel PKS [Int16]

- Moves sensitive data to distinct security domains
- Restricts memory access to these domains
- Based on the principle of least privilege

🏷 Protects 8 sensitive data objects with an average runtime overhead of $\approx$2.3 %

✎ Systematically analyze 11 state-of-the-art data protection schemes

… DOPE significantly improves in terms of security to performance

DOPE: DOmain Protection Enforcement with PKS

- Novel kernel mitigation to protect sensitive data objects
- Enforces domain protection leveraging Intel PKS [Int16]
  - Moves sensitive data to distinct security domains
  - Restricts memory access to these domains
  - Based on the principle of least privilege
- Protects 8 sensitive data objects with an average runtime overhead of $\approx$2.3 %
- Systematically analyze 11 state-of-the-art data protection schemes
  - … DOPE significantly improves in terms of security to performance

DOPE: DOmain Protection Enforcement with PKS

★ Novel kernel mitigation to protect sensitive data objects

Enforces domain protection leveraging Intel PKS [Int16]

- Moves sensitive data to distinct security domains
- Restricts memory access to these domains
- Based on the principle of least privilege

Protects 8 sensitive data objects with an average runtime overhead of ≈2.3 %

Systematically analyze 11 state-of-the-art data protection schemes

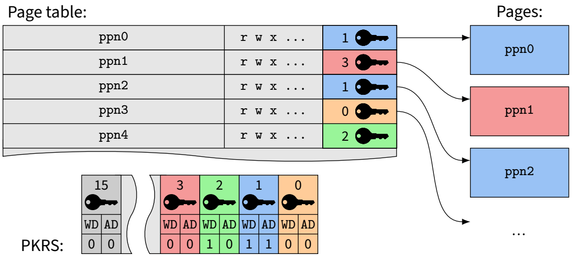
… DOPE significantly improves in terms of security to performance

DOPE: DOmain Protection Enforcement with PKS

- ★ Novel kernel mitigation to protect sensitive data objects
- Enforces domain protection leveraging Intel PKS [Int16]
  - Moves sensitive data to distinct security domains
  - Restricts memory access to these domains
  - Based on the principle of least privilege
- Protects 8 sensitive data objects with an average runtime overhead of $\approx 2.3\,\%$
- Systematically analyze 11 state-of-the-art data protection schemes

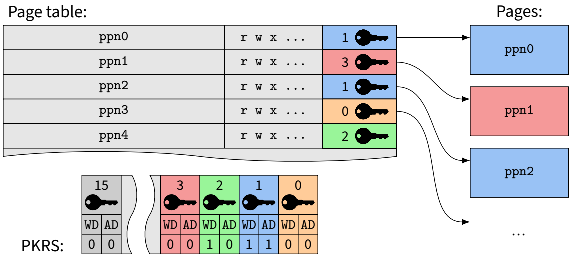
… DOPE significantly improves in terms of security to performance

Intel Protection Keys for Supervisor (PKS)



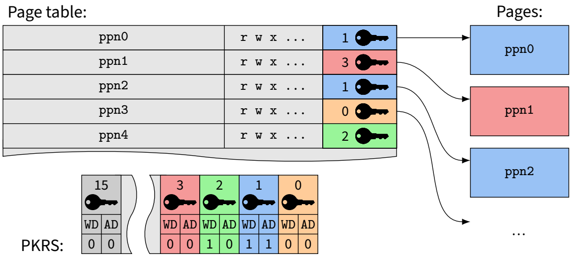
- Intel's implementation of MPK
- Tags page with key
- Access only allowed if permission is set in the PKRS
  - `WD` Write Disabled
  - `AD` Access Disabled
- Permission switch by re/setting `AD`/`WD` bits in the PKRS
- No TLB flush or page table walk

Intel Protection Keys for Supervisor (PKS)



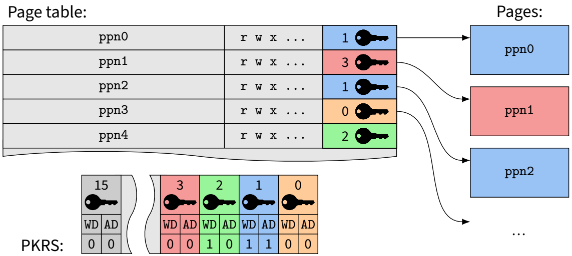
- Intel's implementation of MPK
- Tags page with key
- Access only allowed if permission is set in the PKRS
  - `WD` Write Disabled
  - `AD` Access Disabled
- Permission switch by re/setting `AD`/`WD` bits in the PKRS
- No TLB flush or page table walk

Intel Protection Keys for Supervisor (PKS)



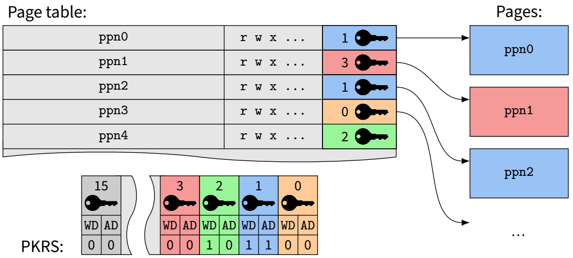
- Intel’s implementation of MPK
- Tags page with key
- Access only allowed if permission is set in the PKRS
  - WD Write Disabled
  - AD Access Disabled
- Permission switch by re/setting AD/WD bits in the PKRS
- No TLB flush or page table walk

Intel Protection Keys for Supervisor (PKS)



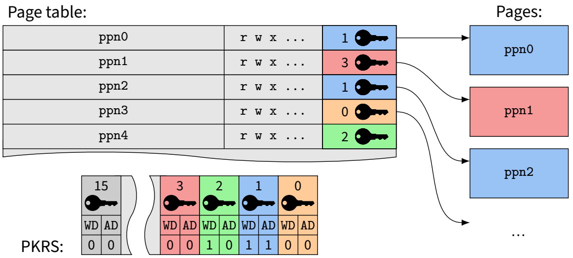
- Intel's implementation of MPK
- Tags page with key
- Access only allowed if permission is set in the PKRS
  - `WD` Write Disabled
  - `AD` Access Disabled
- Permission switch by re/setting `AD`/`WD` bits in the PKRS
- No TLB flush or page table walk

Intel Protection Keys for Supervisor (PKS)



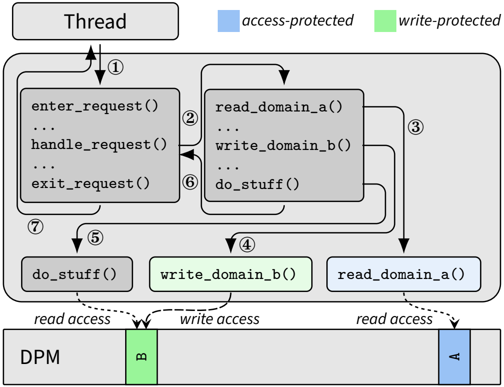
- Intel's implementation of MPK
- Tags page with key
- Access only allowed if permission is set in the PKRS
  - `WD` Write Disabled
  - `AD` Access Disabled
- Permission switch by re/setting `AD`/`WD` bits in the PKRS
- No TLB flush or page table walk

Intel Protection Keys for Supervisor (PKS)



- Intel’s implementation of MPK
- Tags page with key
- Access only allowed if permission is set in the PKRS
    - `WD` Write Disabled
    - `AD` Access Disabled
- Permission switch by re/setting `AD`/`WD` bits in the PKRS
- No TLB flush or page table walk

DOPE



① Enters a kernel execution request and obtains restricted access permissions

② Handles the execution request

③ Legally reads access-protected data

- By switching to domain A

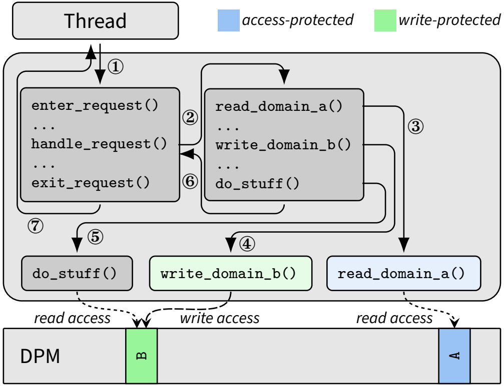
④ Legally writes write-protected data

- By switching to domain B

⑤ Reads write-protected data

⑥/⑦ Exits the execution request

DOPE



① Enters a kernel execution request and obtains restricted access permissions

② Handles the execution request

③ Legally reads access-protected data

- By switching to domain A

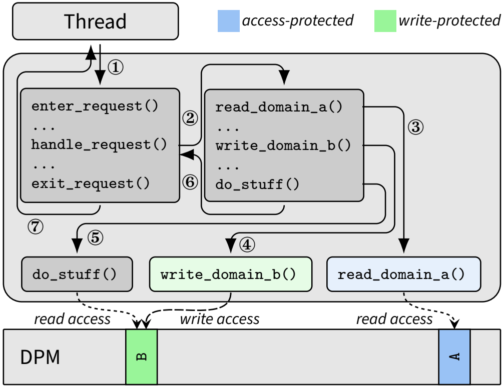
④ Legally writes write-protected data

- By switching to domain B

⑤ Reads write-protected data

⑥/⑦ Exits the execution request

DOPE



① Enters a kernel execution request and obtains restricted access permissions

② Handles the execution request

③ Legally reads access-protected data

- By switching to domain A

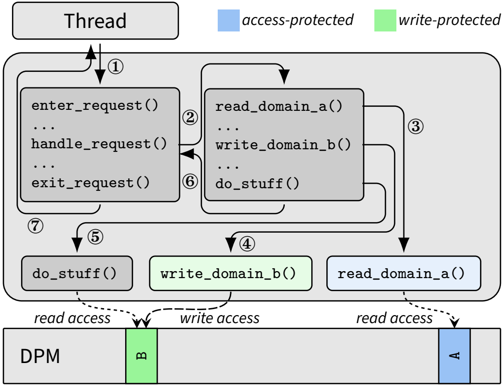
④ Legally writes write-protected data

- By switching to domain B

⑤ Reads write-protected data

⑥/⑦ Exits the execution request

DOPE



① Enters a kernel execution request and obtains restricted access permissions

② Handles the execution request

③ Legally reads access-protected data

- By switching to domain A

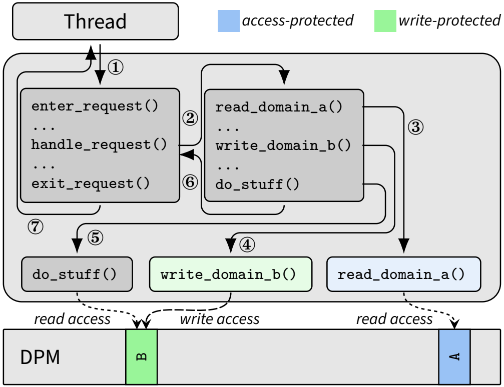
④ Legally writes write-protected data

- By switching to domain B

⑤ Reads write-protected data

⑥/⑦ Exits the execution request

DOPE



① Enters a kernel execution request and obtains restricted access permissions

② Handles the execution request

③ Legally reads access-protected data

- By switching to domain A

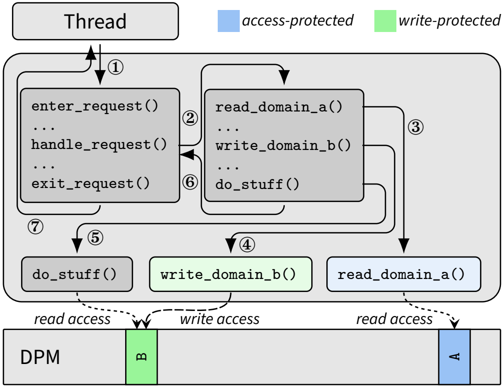
④ Legally writes write-protected data

- By switching to domain B

⑤ Reads write-protected data

⑥/⑦ Exits the execution request

DOPE



① Enters a kernel execution request and obtains restricted access permissions

② Handles the execution request

③ Legally reads access-protected data

- By switching to domain A

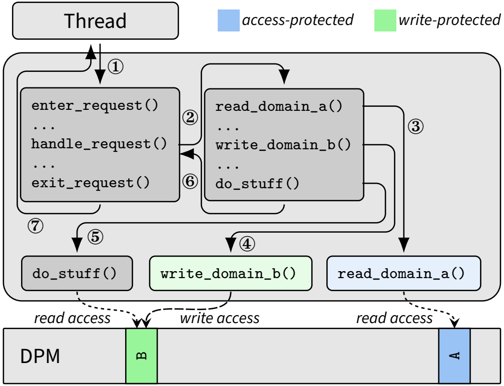
④ Legally writes write-protected data

- By switching to domain B

⑤ Reads write-protected data

⑥/⑦ Exits the execution request

DOPE



① Enters a kernel execution request and obtains restricted access permissions

② Handles the execution request

③ Legally reads access-protected data

- By switching to domain A

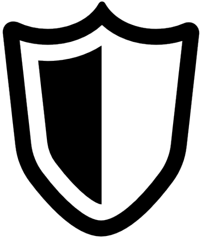
④ Legally writes write-protected data

- By switching to domain B

⑤ Reads write-protected data

⑥/⑦ Exits the execution request

DOPE contd

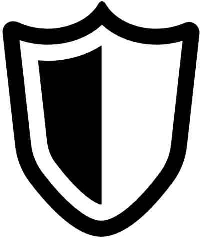


- Predefined restricted access permissions
- Sensitive data access in trusted code locations
    - Predefined before compile-time
    - Semi-automatic approach with compiler pass
- Three variants of enforcing domain protection with PKS
    - Entire data object protection
    - Shadow memory protection
    - Sensitive data protection
- Pointer integrity through ownership

DOPE contd

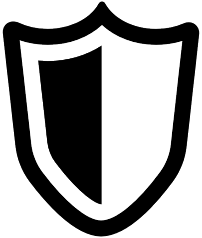
- Predefined restricted access permissions
- Sensitive data access in trusted code locations
    - Predefined before compile-time
    - Semi-automatic approach with compiler pass
- Three variants of enforcing domain protection with PKS
    - Entire data object protection
    - Shadow memory protection
    - Sensitive data protection
- Pointer integrity through ownership

DOPE contd



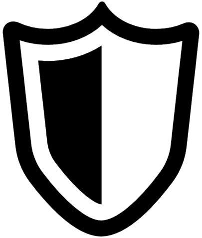
- Predefined restricted access permissions
- Sensitive data access in trusted code locations
  - Predefined before compile-time
  - Semi-automatic approach with compiler pass
- Three variants of enforcing domain protection with PKS
  - Entire data object protection
  - Shadow memory protection
  - Sensitive data protection
- Pointer integrity through ownership

DOPE contd



- Predefined restricted access permissions
- Sensitive data access in trusted code locations
  - Predefined before compile-time
  - Semi-automatic approach with compiler pass
- Three variants of enforcing domain protection with PKS
  - Entire data object protection
  - Shadow memory protection
  - Sensitive data protection
- Pointer integrity through ownership

DOPE contd



- Predefined restricted access permissions
- Sensitive data access in trusted code locations
  - Predefined before compile-time
  - Semi-automatic approach with compiler pass
- Three variants of enforcing domain protection with PKS
  - Entire data object protection
  - Shadow memory protection
  - Sensitive data protection
- Pointer integrity through ownership

Trusted Code

```
/* get ext4 inode */
struct inode *ext4_iget(){
  struct ext4_inode *ei;
  struct inode *ino;
  ...
  ino = dentry->inode;


  ino->i_uid = i_uid;
  ino->i_gid = i_gid;

  ei->i_data[blk] = data;
  ...
  return ino;
}
```

- `ext4_iget` function returns ext4 inode
  - Access `inode` from its owner `dentry`
  - Legally overwrites sensitive data i.e., `i_*id`
- Code analyzer detects accesses, i.e., owner and sensitive data
- Insert domain switches
- Insert owner validation

Trusted Code

```
/* get ext4 inode */
struct inode *ext4_iget(){
  struct ext4_inode *ei;
  struct inode *ino;
  ...
  ino = dentry->inode;


  ino->i_uid = i_uid;
  ino->i_gid = i_gid;

  ei->i_data[blk] = data;
  ...
  return ino;
}
```

- `ext4_iget` function returns ext4 inode
  - Access `inode` from its owner `dentry`
  - Legally overwrites sensitive data i.e., `i_*id`
- Code analyzer detects accesses, i.e., owner and sensitive data
- Insert domain switches
- Insert owner validation

Trusted Code

```
/* get ext4 inode */
struct inode *ext4_iget(){
  struct ext4_inode *ei;
  struct inode *ino;
  ...
  ino = dentry->inode;


  ino->i_uid = i_uid;
  ino->i_gid = i_gid;

  ei->i_data[blk] = data;
  ...
  return ino;
}
```

- `ext4_iget` function returns ext4 inode
  - Access `inode` from its owner `dentry`
  - Legally overwrites sensitive data i.e., `i_*id`
- Code analyzer detects accesses, i.e., owner and sensitive data
- Insert domain switches
- Insert owner validation

Trusted Code

```
/* get ext4 inode */
struct inode *ext4_iget(){
  struct ext4_inode *ei;
  struct inode *ino;
  ...
  ino = dentry->inode;

+ enter_inode_wr();
  ino->i_uid = i_uid;
  ino->i_gid = i_gid;
+ exit_inode_wr();
  ei->i_data[blk] = data;
  ...
  return ino;
}
```

- `ext4_iget` function returns ext4 inode
  - Access `inode` from its owner `dentry`
  - Legally overwrites sensitive data i.e., `i_*id`
- Code analyzer detects accesses, i.e., owner and sensitive data
- Insert domain switches
- Insert owner validation

Trusted Code

```
/* get ext4 inode */
struct inode *ext4_iget(){
  struct ext4_inode *ei;
  struct inode *ino;
  ...
  ino = dentry->inode;
+ owner_check(dentry, ino);
+ enter_inode_wr();
  ino->i_uid = i_uid;
  ino->i_gid = i_gid;
+ exit_inode_wr();
  ei->i_data[blk] = data;
  ...
  return ino;
}
```

- `ext4_iget` function returns ext4 inode
  - Access `inode` from its owner `dentry`
  - Legally overwrites sensitive data i.e., `i_*id`
- Code analyzer detects accesses, i.e., owner and sensitive data
- Insert domain switches
- Insert owner validation

Ownership

- Sensitive data object comprises the owner's address
- Validation check
  - Owner same
  - Sensitive data object correctly tagged
- Multiple ownership
  - Store both addresses in hashtable
  - Hashtable tagged same domain

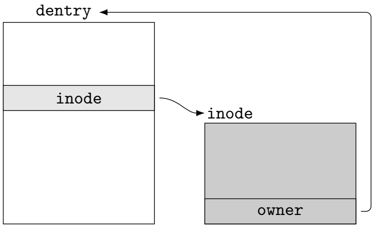


Figure: Ownership-based protection is employed to protect the sensitive pointer to `inode` within its owner `dentry`.

Ownership

- Sensitive data object comprises the owner's address
- Validation check
  - Owner same
  - Sensitive data object correctly tagged
- Multiple ownership
  - Store both addresses in hashtable
  - Hashtable tagged same domain

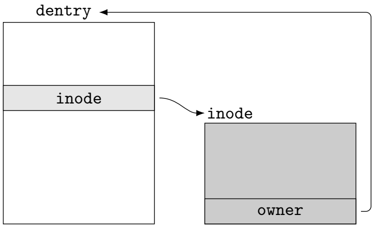


Figure: Ownership-based protection is employed to protect the sensitive pointer to `inode` within its owner `dentry`.

Ownership

- Sensitive data object comprises the owner's address
- Validation check
  - Owner same
  - Sensitive data object correctly tagged
- Multiple ownership
  - Store both addresses in hashtable
  - Hashtable tagged same domain

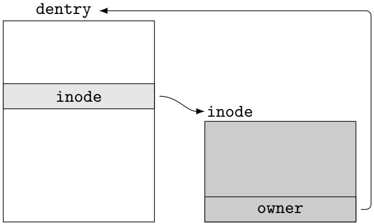


Figure: Ownership-based protection is employed to protect the sensitive pointer to `inode` within its owner `dentry`.

Ownership

- Sensitive data object comprises the owner's address
- Validation check
  - Owner same
  - Sensitive data object correctly tagged
- Multiple ownership
  - Store both addresses in hashtable
  - Hashtable tagged same domain

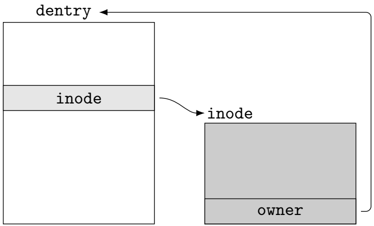


Figure: Ownership-based protection is employed to protect the sensitive pointer to `inode` within its owner `dentry`.

Enforcing Domain Protection with PKS

Table: Applied protection variant for our sensitive data objects.

Variant	Sensitive data objects								
	User-accessible pages	Credentials	Inodes	Page tables	Virtual memory areas	Virtual memory	Filesystem mount	Stored registers	Sensitive state
Entire date object protection	●	●	○	●	○	○	○	●	●
Shadow memory protection	○	○	○	○	●	●	●	○	○
Sensitive data protection	○	○	●	○	○	○	○	○	○

● Applied ○ Not applied

Performance Overhead

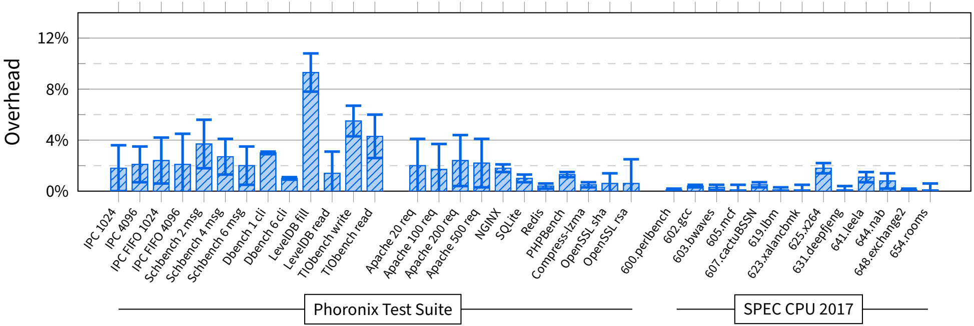


Figure: We implement our DOPE proof-of-concept in Linux kernel v5.19 and run it on Ubuntu 22.04.1 LTS with a recent Intel Alder Lake processor.

Systematic Analysis

Table: Systematic overview of mitigations against data-oriented attacks in the Linux kernel.

Mitigations	Sensitive Data Objects									Overhead
	Credentials	Virtual memory	Virtual memory areas	Inodes	Page tables	Filesystem mount	Other non-control data	User-accessible pages	Stored registers	
PrivGuard [QYJS18]	○	○	-	-	-	-	-	-	-	⧖
AKO [YAY$^+$21]	○	-	-	-	-	-	-	-	-	⧖
PrivWatcher [CAGN17]	●	●	-	-	-	-	-	-	-	⧗[1]
SALADS [CXL$^+$15]	●	-	-	●	-	-	●[2]	-	-	⧗
PT-Rand [DGLS17]	-	-	-	-	●	-	-	-	-	⧖
Mondrix [WRA05]	-	-	-	-	-	-	●	-	●	⧗[1]
HAKC [MGP$^+$22]	○	○	●	○	●	○	●	○	◐	⧗
KDPM [KY22a]	○	-	-	-	-	-	-	-	-	⧖[1]
KPRM [KY22b]	○	-	-	-	-	-	○	-	-	⧗
KENALI [SLL$^+$16]	◐	●	◐	◐	●	◐	●	-	●	⧗
xMP [PMG$^+$20]	◐	●	-	-	●	-	●[3]	-	-	⧗
DOPE [**our solution**]	●	●	●	●	●	●	-	●	●	⏳

● Strong protection ◐ Partial protection ○ Insufficient protection - Not protected

⧖ Low overhead ⏳ Reasonable overhead ⧗ High overhead

[1] Not tested on hardware [2] Non-sensitive data [3] User space data

Conclusion

- Presented DOPE, a novel kernel mitigation to protect sensitive data objects
- Implementation and case study to protect 8 sensitive data objects
  - Opensource: `https://extgit.iaik.tugraz.at/sesys/dope`
- Performance evaluation on real hardware shows an average runtime overhead of ≈2.3 %
- Systematically analyze 11 state-of-the-art data protection schemes

Conclusion

- Presented DOPE, a novel kernel mitigation to protect sensitive data objects
- Implementation and case study to protect 8 sensitive data objects
  - Opensource: `https://extgit.iaik.tugraz.at/sesys/dope`
- Performance evaluation on real hardware shows an average runtime overhead of ≈2.3 %
- Systematically analyze 11 state-of-the-art data protection schemes

Conclusion

- Presented DOPE, a novel kernel mitigation to protect sensitive data objects
- Implementation and case study to protect 8 sensitive data objects
  - Opensource: `https://extgit.iaik.tugraz.at/sesys/dope`
- Performance evaluation on real hardware shows an average runtime overhead of ≈2.3 %
- Systematically analyze 11 state-of-the-art data protection schemes

Conclusion

- Presented DOPE, a novel kernel mitigation to protect sensitive data objects
- Implementation and case study to protect 8 sensitive data objects
  - Opensource: `https://extgit.iaik.tugraz.at/sesys/dope`
- Performance evaluation on real hardware shows an average runtime overhead of $\approx$2.3 %
- Systematically analyze 11 state-of-the-art data protection schemes

Conclusion

- Presented DOPE, a novel kernel mitigation to protect sensitive data objects
- Implementation and case study to protect 8 sensitive data objects
  - Opensource: `https://extgit.iaik.tugraz.at/sesys/dope`
- Performance evaluation on real hardware shows an average runtime overhead of ≈2.3 %
- Systematically analyze 11 state-of-the-art data protection schemes

Thank you for your attention!

References I

[ABEL05] Martín Abadi, Mihai Budiu, Ulfar Erlingsson, and Jay Ligatti, *Control-Flow Integrity*, CCS, 2005.

[Ale21] Alexander Popov, *Four Bytes of Power: Exploiting CVE-2021-26708 in the Linux kernel*, 2021.

[CAGN17] Quan Chen, Ahmed M. Azab, Guruprasad Ganesh, and Peng Ning, *PrivWatcher: Non-Bypassable Monitoring and Protection of Process Credentials from Memory Corruption Attacks*, AsiaCCS, 2017.

[CDA14] John Criswell, Nathan Dautenhahn, and Vikram Adve, *KCoFI: Complete Control-Flow Integrity for Commodity Operating System Kernels*, S&P, 2014.

References II

[CXL+15] Ping Chen, Jun Xu, Zhiqiang Lin, Dongyan Xu, Bing Mao, and Peng Liu, *A Practical Approach for Adaptive Data Structure Layout Randomization*, European Symposium on Research in Computer Security, 2015.

[DGLS17] Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi, *PT-Rand: Practical Mitigation of Data-only Attacks against Page Tables*, NDSS, 2017.

[Edg20] Jake Edge, *Control-flow integrity for the kernel*, 2020.

[Goo19] Google Project Zero, *CVE-2019-2215: Android use-after-free in Binder*, 2019.

[Goo21] ________ , *CVE-2021-0920: Android sk_buff use-after-free in Linux*, 2021.

References III

[Int16] Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture*, 2016.

[KY22a] Hiroki Kuzuno and Toshihiro Yamauchi, *KDPM: Kernel Data Protection Mechanism Using a Memory Protection Key*, International Workshop on Security (2022), 66–85.

[KY22b] ______, *Prevention of Kernel Memory Corruption Using Kernel Page Restriction Mechanism*, Journal of Information Processing **30** (2022), 563–576.

[LWX22] Zhenpeng Lin, Yuhang Wu, and Xinyu Xing, *DirtyCred: Escalating Privilege in Linux Kernel*, ACM, 2022.

References IV

[MGP+22] Derrick McKee, Yianni Giannaris, Carolina Ortega Perez, Howard Shrobe, Mathias Payer, Hamed Okhravi, and Nathan Burow, *Preventing Kernel Hacks with HAKC*, NDSS, 2022.

[Nic23] Nicolas Wu, *Dirty Pagetable: A Novel Exploitation Technique To Rule Linux Kernel*, 2023.

[PMG+20] Sergej Proskurin, Marius Momeu, Seyedhamed Ghavamnia, Vasileios P. Kemerlis, and Michalis Polychronakis, *xMP: Selective Memory Protection for Kernel and User Space*, S&P, 2020.

References V

[QYJS18] Weizhong Qiang, Jiawei Yang, Hai Jin, and Xuanhua Shi, *PrivGuard: Protecting Sensitive Kernel Data From Privilege Escalation Attacks*, IEEE Access **6** (2018), 46584–46594.

[SLL⁺16] Chengyu Song, Byoungyoung Lee, Kangjie Lu, William R. Harris, Taesoo Kim, and Wenke Lee, *Enforcing Kernel Security Invariants with Data Flow Integrity*, NDSS, 2016.

[WRA05] Emmett Witchel, Junghwan Rhee, and Krste Asanović, *Mondrix: Memory Isolation for Linux Using Mondriaan Memory Protection*, ACM SIGOPS Operating Systems Review, 2005.

References VI

[YAY$^+$21] Toshihiro Yamauchi, Yohei Akao, Ryota Yoshitani, Yuichi Nakamura, and Masaki Hashimoto, *Additional kernel observer: privilege escalation attack prevention mechanism focusing on system call privilege changes*, International Journal of Information Security **20** (2021).