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
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
Motivation

Exploitation

Goals of adversaries

- Leaking sensitive informations, e.g., $\psi_2$, $\psi_1$, or $\psi$
- Resource compromising
- ...

Kernel security

- Isolate different entities

Kernel vulnerabilities

- Exploitation to bypass isolation primitives
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

Motivation

Lukas Maar, Martin Schwarzl, Fabian Rauscher, Daniel Gruss, Stefan Mangard
7 December 2023
Motivation

CVEs in the Linux Kernel

![Graph showing the number of CVEs in the Linux Kernel over the years from 2001 to 2023.](image)

**Figure:** Found Linux kernel CVEs from NIST NVD.

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

7 December 2023
Motivation

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 \(\rightarrow\) escalate privileges

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

What about corrupting non-control data?
Motivation

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?
Motivation

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
  - ...

Lukas Maar, Martin Schwarzl, Fabian Rauscher, Daniel Gruss, Stefan Mangard
7 December 2023
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
- ...

Lukas Maar, Martin Schwarzl, Fabian Rauscher, Daniel Gruss, Stefan Mangard
7 December 2023
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
- ...

Lukas Maar, Martin Schwarzl, Fabian Rauscher, Daniel Gruss, Stefan Mangard
7 December 2023
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
- ...

Lukas Maar, Martin Schwarzl, Fabian Rauscher, Daniel Gruss, Stefan Mangard
7 December 2023
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
  - ...

```
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;
```
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
  - ...

1 struct inode {
2   umode_t i_mode;
3   kuid_t i_uid;
4   kgid_t i_gid;
5   unsigned int i_flags;
6   ...
7 } __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

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

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

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 Portection
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 \( \approx 2.3\% \)

GetEnumerator

- 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

닫.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 ≈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’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**

#### Page table:

<table>
<thead>
<tr>
<th>ppn</th>
<th>r</th>
<th>w</th>
<th>x</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppn0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ppn1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ppn2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ppn3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ppn4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Pages:

<table>
<thead>
<tr>
<th>ppn</th>
<th>WD</th>
<th>AD</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppn0</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>ppn1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>ppn2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

#### PKRS:

<table>
<thead>
<tr>
<th>WD</th>
<th>AD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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
Sensitive Data Portection

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

Lukas Maar, Martin Schwarzl, Fabian Rauscher, Daniel Gruss, Stefan Mangard
7 December 2023
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
Sensitivl Data Portection

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

Lukas Maar, Martin Schwarzl, Fabian Rauscher, Daniel Gruss, Stefan Mangard
7 December 2023
Enter a kernel execution request and obtains restricted access permissions

Handles the execution request

Legally reads access-protected data
  - By switching to domain A

Legally writes write-protected data
  - By switching to domain B

Reads write-protected data

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
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**

```c
/* 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**
/* 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
```c
/* 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

Lukas Maar, Martin Schwarzl, Fabian Rauscher, Daniel Gruss, Stefan Mangard
7 December 2023
Trusted Code

```c
/* 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
  - HasTable 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.
**Table**: Applied protection variant for our sensitive data objects.

<table>
<thead>
<tr>
<th>Variant</th>
<th>Sensitive data objects</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>User-accessible pages</td>
</tr>
<tr>
<td>Entire date object protection</td>
<td>● ● ○ ● ● ○ ○ ○ ● ● ●</td>
</tr>
<tr>
<td>Shadow memory protection</td>
<td>○ ○ ○ ○ ● ● ● ● ○ ○ ○</td>
</tr>
<tr>
<td>Sensitive data protection</td>
<td>○ ○ ● ○ ○ ○ ○ ○ ○ ○</td>
</tr>
</tbody>
</table>

● Applied  ○ Not applied
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.

<table>
<thead>
<tr>
<th>Mitigations</th>
<th>Sensitive Data Objects</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Credentials</td>
<td>Virtual memory</td>
</tr>
<tr>
<td>PrivGuard [QYJS18]</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>AKO [YAY+21]</td>
<td>○</td>
<td>-</td>
</tr>
<tr>
<td>PrivWatcher [CAGN17]</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>SALADS [CXL+15]</td>
<td>●</td>
<td>-</td>
</tr>
<tr>
<td>PT-Rand [DGLS17]</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Mondrix [WRA05]</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>HAKC [MGP+22]</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>KDPM [KY22a]</td>
<td>○</td>
<td>-</td>
</tr>
<tr>
<td>KPRM [KY22b]</td>
<td>○</td>
<td>-</td>
</tr>
<tr>
<td>KENALI [SLL+16]</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>xMP [PMG+20]</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>DOPE [our solution]</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

- **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

---

Lukas Maar, Martin Schwarzl, Fabian Rauscher, Daniel Gruss, Stefan Mangard
7 December 2023
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 $\approx 2.3\%$
- 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
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
Thank you for your attention!
References I


References II


References IV


