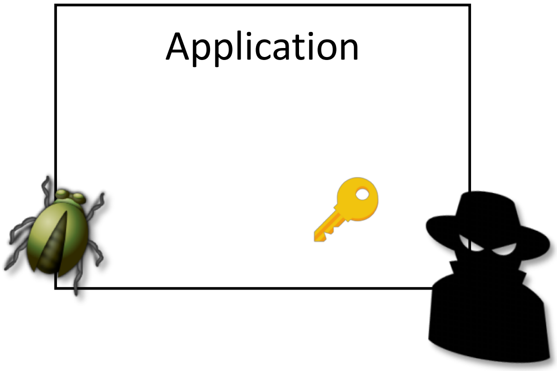
# ERIM: Secure, Efficient in-process Isolation with Memory Protection Keys

**Anjo Vahldiek-Oberwagner**, Eslam Elnikety, Nuno O. Duarte,
Michael Sammler, Peter Druschel, Deepak Garg

MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

# Applications in the **Absence of Isolation**

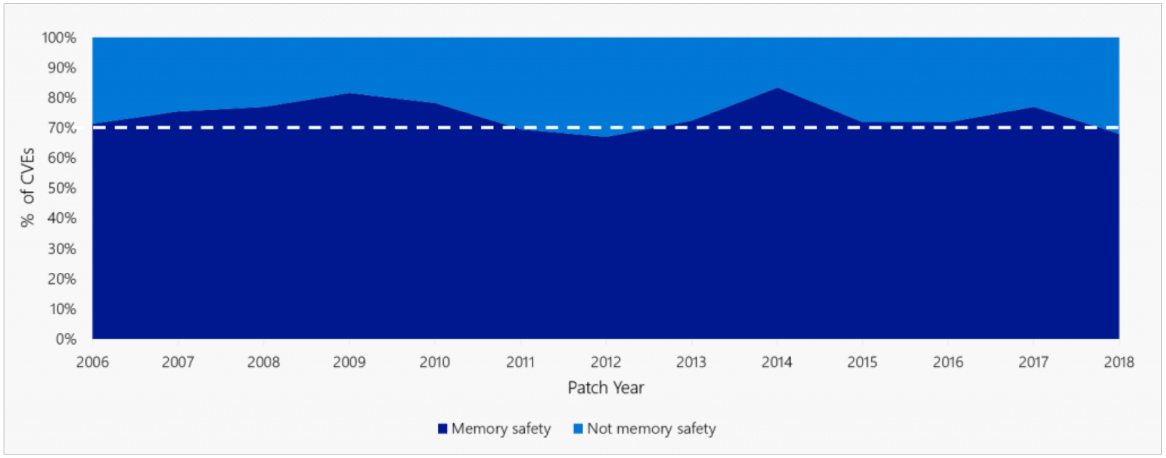
- All state accessible at **all times** to
  - Bugs
  - Security vulnerabilities



# Applications in the **Absence of Isolation**

**Heartbleed Bug**

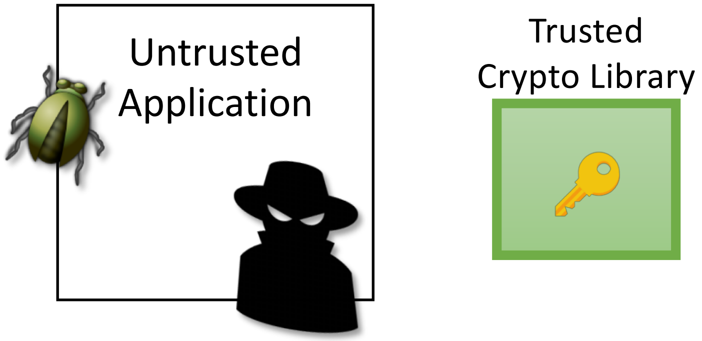
**~70% of CVE assigned by Microsoft are memory safety issues.**



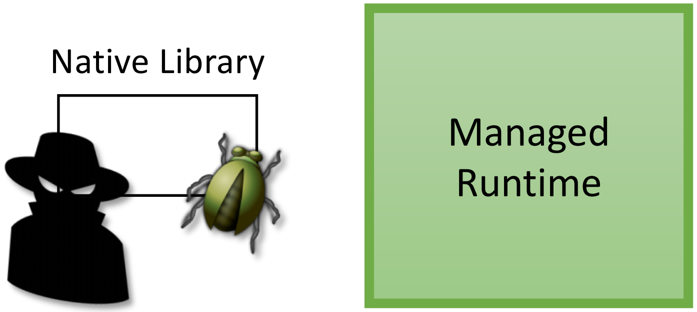
Microsoft Security Response Center: “A proactive approach to more secure code”, 2019

# Example In-Process Isolation Use Cases

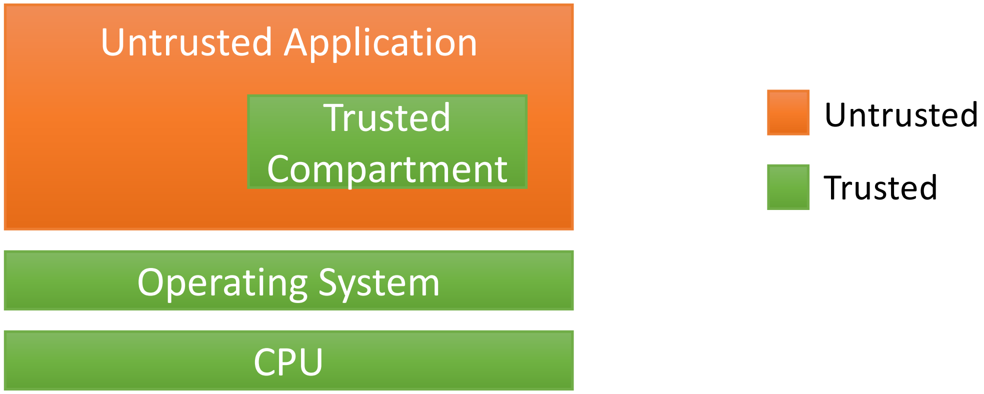
## Cryptographic Secrets



## Managed runtimes from native libraries



# User-space Threat Model



Untrusted Application

Trusted Compartment

Operating System

CPU

🟧 Untrusted

🟩 Trusted

Attacker’s Capabilities include, but not limited to

- Control-flow hijacks
- Memory corruption (i.e., out-of-bounds accesses)

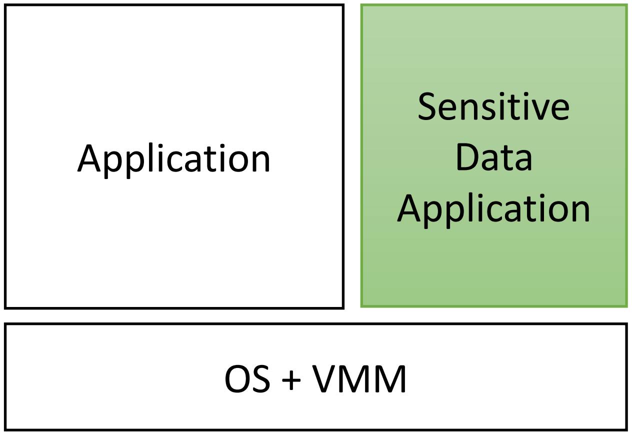
Out of scope:

- Side-channel, row hammer or microarchitectural attacks

# State of In-Application Isolation Techniques

| | Execution overhead | | Switch overhead |
|---|---|---|---|
| | Untrusted | Trusted | |
| **OS/VMM -based[2]** | **Low** | **Low** | **Medium** |
| Lang. & RT[3] | Medium – High | None | None |
| ERIM | Low | None | Low |

### OS/VMM Technique



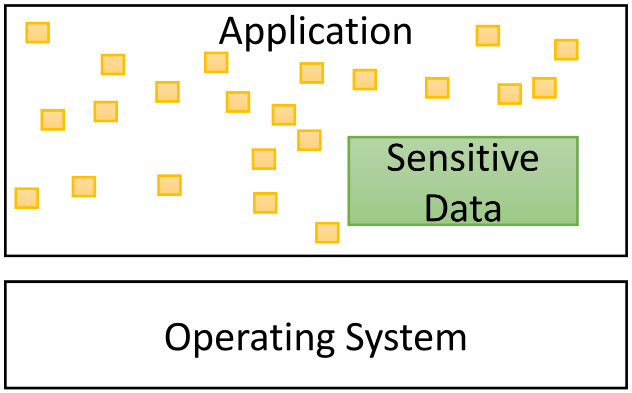
[1] LwC, SMVs, Shreds, Wedge, Nexen, Dune, SeCage, TrustVisor

[2] SFI

# State of In-Application Isolation Techniques

| | Execution overhead | | Switch overhead |
|---|---|---|---|
| | Untrusted | Trusted | |
| OS/VMM-based[2] | Low | Low | Medium |
| **Lang. & RT[3]** | **Medium – High** | **None** | **None** |
| ERIM | Low | None | Low |

## Language and Runtime Techniques

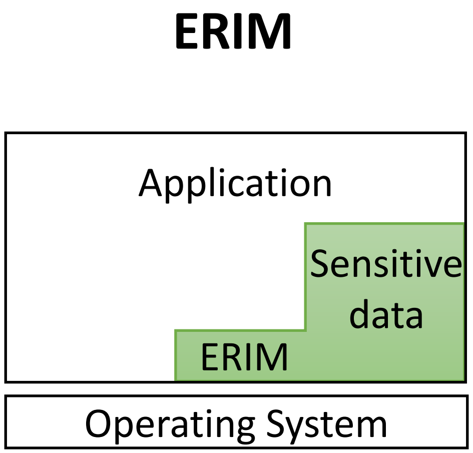


[1] LwC, SMVs, Shreds, Wedge, Nexen, Dune, SeCage, TrustVisor

[2] SFI

# State of In-Application Isolation Techniques

| | Execution overhead | | Switch overhead |
|---|---|---|---|
| | Untrusted | Trusted | |
| OS/VMM-based[2] | Low | Low | Medium |
| Lang. & RT[3] | Medium – High | None | None |
| **ERIM** | **Low** | **None** | **Low** |

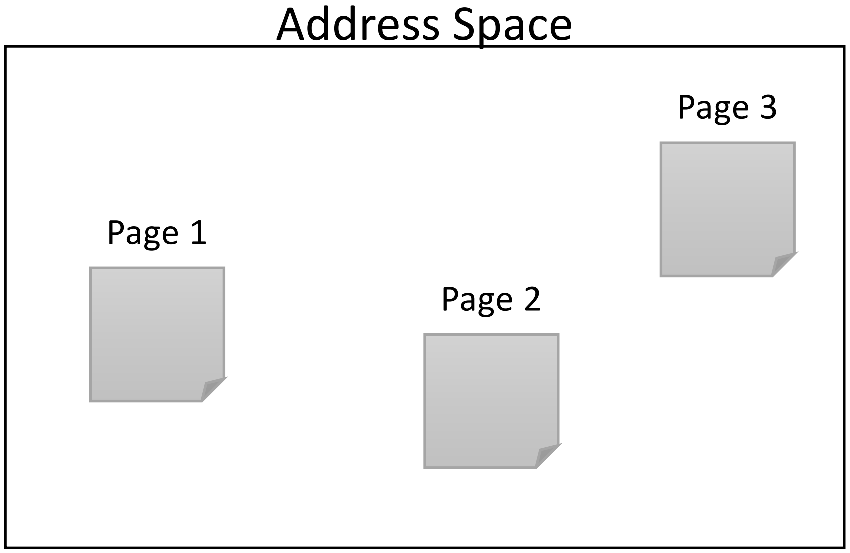


[1] LwC, SMVs, Shreds, Wedge, Nexen, Dune, SeCage, TrustVisor

[2] SFI, Native Client, Memsentry-MPX

# Memory Protection Keys (MPK)

- Available in Skylake server CPUs
- Tag memory pages with PKEY

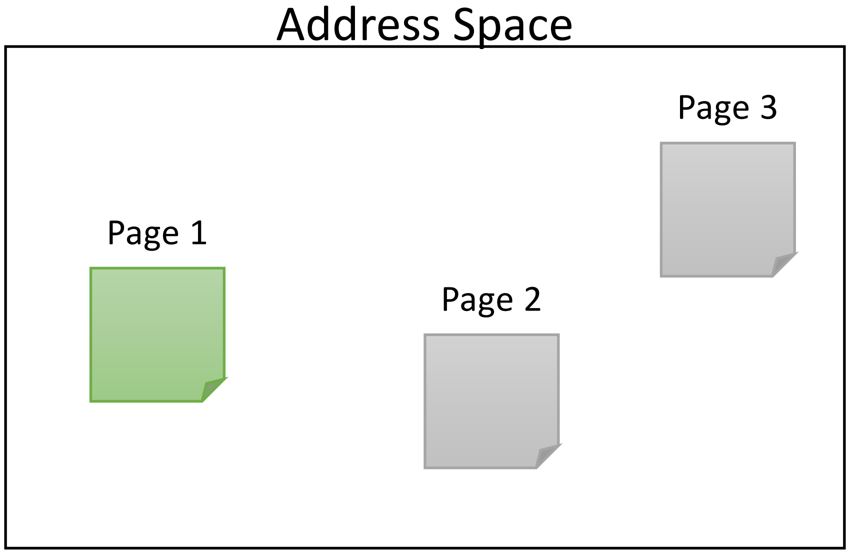


Page Table Entry (PTE)

|  |  |  |  |  |
|---|---|---|---|---|
| … | **PKEY**<br>**0** | … | Page 1 | … |

# Intel Memory Protection Keys (MPK)

- Available in Skylake server CPUs
- Tag memory pages with PKEY

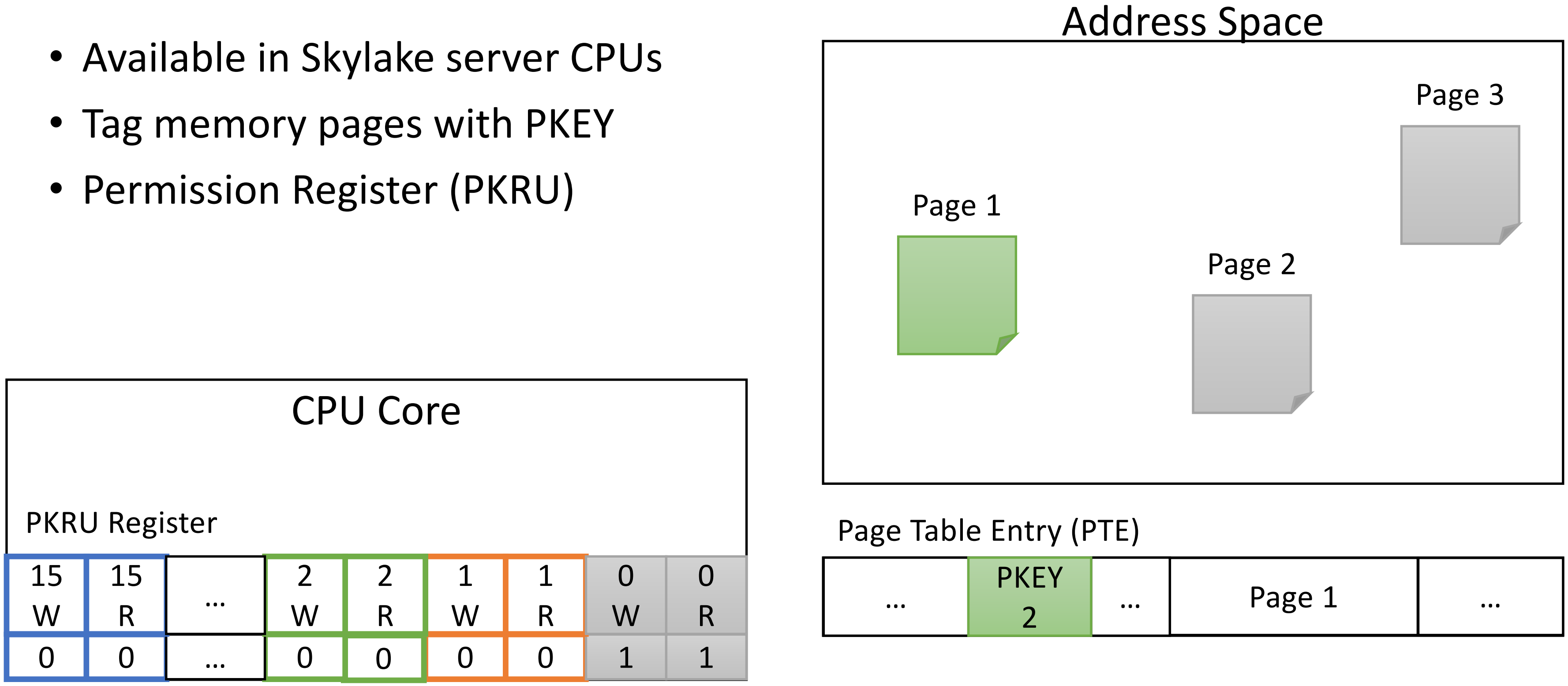


Page Table Entry (PTE)

| | | | | |
|---|---|---|---|---|
| … | PKEY<br>2 | … | Page 1 | … |

# Intel Memory Protection Keys (MPK)

- Available in Skylake server CPUs
- Tag memory pages with PKEY
- Permission Register (PKRU)



PKRU Register

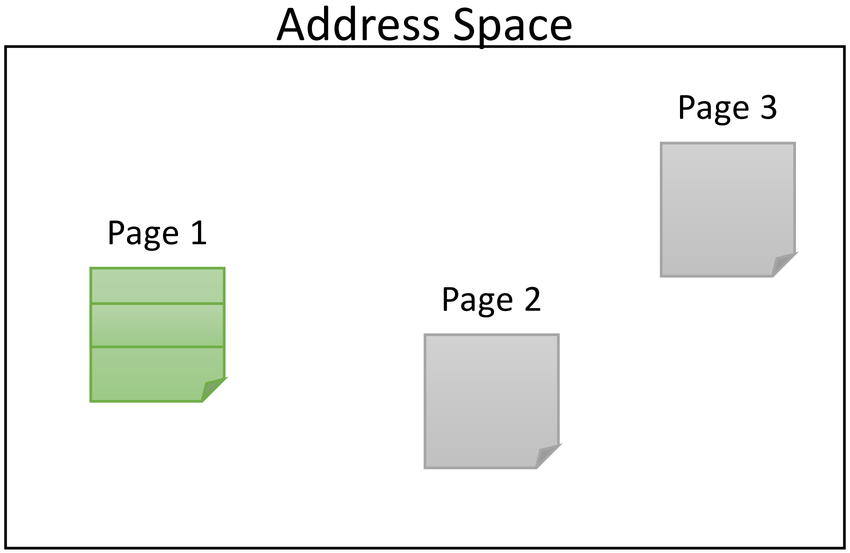
| 15<br>W | 15<br>R | … | 2<br>W | 2<br>R | 1<br>W | 1<br>R | 0<br>W | 0<br>R |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | … | 0 | 0 | 0 | 0 | 1 | 1 |

Page Table Entry (PTE)

| | | | | |
|---|---|---|---|---|
| … | PKEY<br>2 | … | Page 1 | … |

# Intel Memory Protection Keys (MPK)

- Available in Skylake server CPUs
- Tag memory pages with PKEY
- Permission Register (PKRU)
- Userspace instruction to update PKRU
  - Fast switch between 11 – 260 cycles/switch



CPU Core

PKRU Register

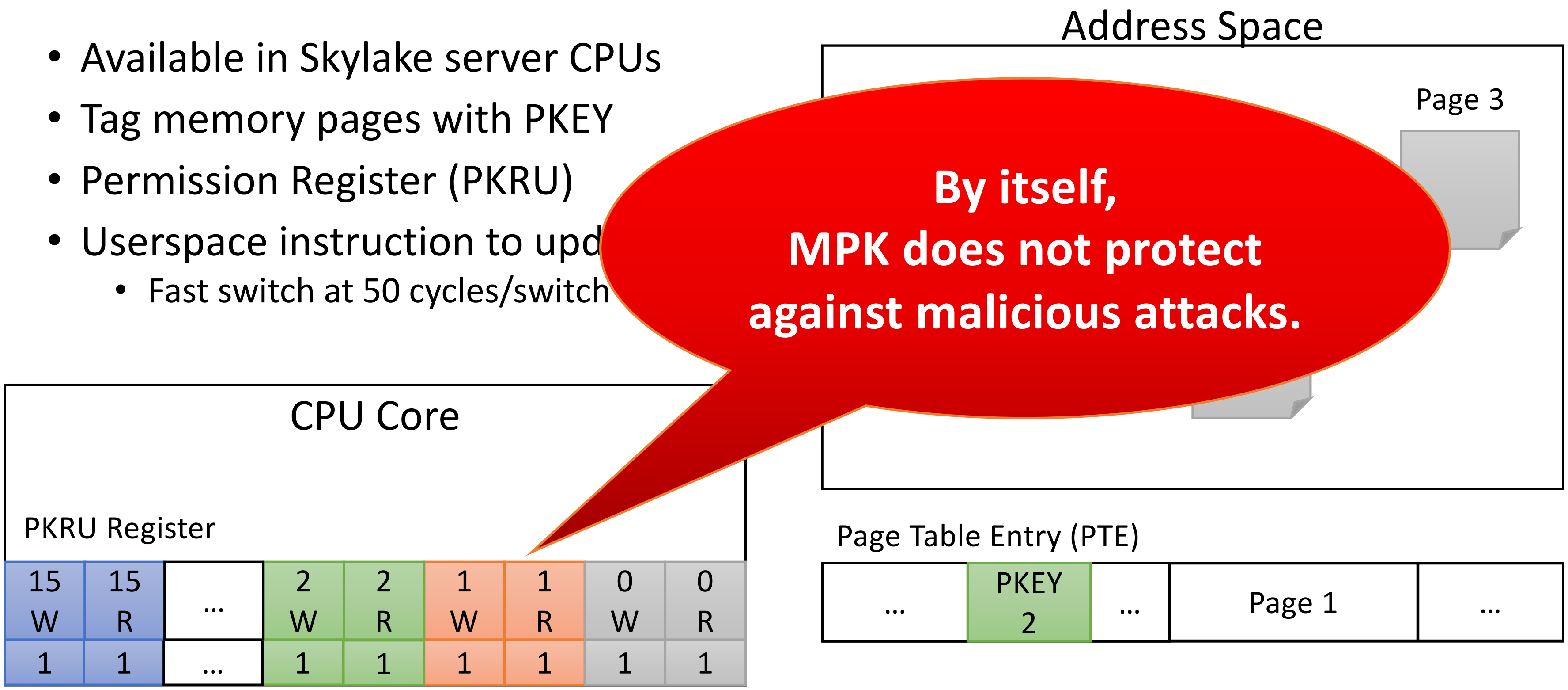
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 15<br>W | 15<br>R | … | 2<br>W | 2<br>R | 1<br>W | 1<br>R | 0<br>W | 0<br>R |
| 0 | 0 | … | 1 | 1 | 0 | 0 | 1 | 1 |

Page Table Entry (PTE)

| | | | | |
|---|---|---|---|---|
| … | PKEY<br>2 | … | Page 1 | … |

# Intel Memory Protection Keys (MPK)

- Available in Skylake server CPUs
- Tag memory pages with PKEY
- Permission Register (PKRU)
- Userspace instruction to upd
  - Fast switch at 50 cycles/switch



By itself,
MPK does not protect
against malicious attacks.

PKRU Register

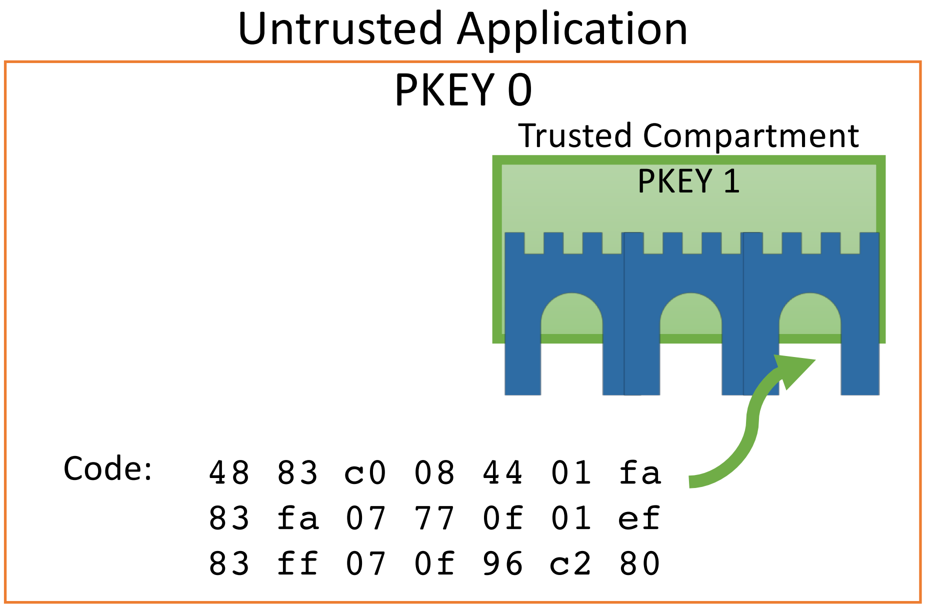
| 15<br>W | 15<br>R | … | 2<br>W | 2<br>R | 1<br>W | 1<br>R | 0<br>W | 0<br>R |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | … | 1 | 1 | 1 | 1 | 1 | 1 |

Page Table Entry (PTE)

| | | | | |
|---|---|---|---|---|
| … | PKEY<br>2 | … | Page 1 | … |

# Overview of ERIM

- Prevent MPK exploitation
  - Safe call gates
  - Prevent execution of permission register updates outside of call gates



Untrusted Application

PKEY 0

Trusted Compartment

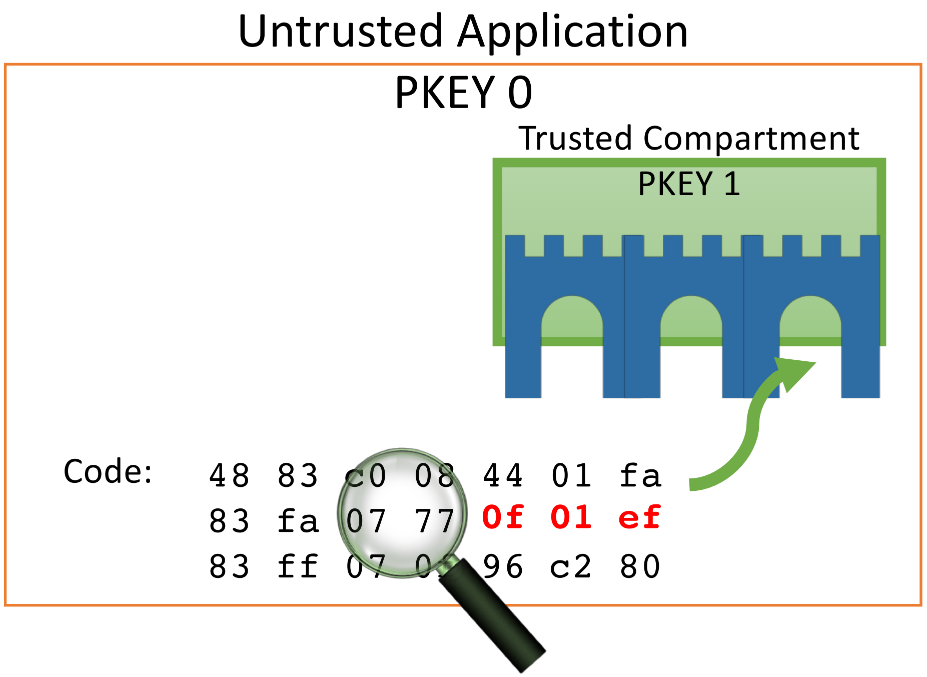
PKEY 1

Code:

```
48 83 c0 08 44 01 fa
83 fa 07 77 0f 01 ef
83 ff 07 0f 96 c2 80
```

# Overview of ERIM

- Prevent MPK exploitation
  - Safe call gates
  - Prevent execution of permission register updates outside of call gates



Untrusted Application

PKEY 0

Trusted Compartment

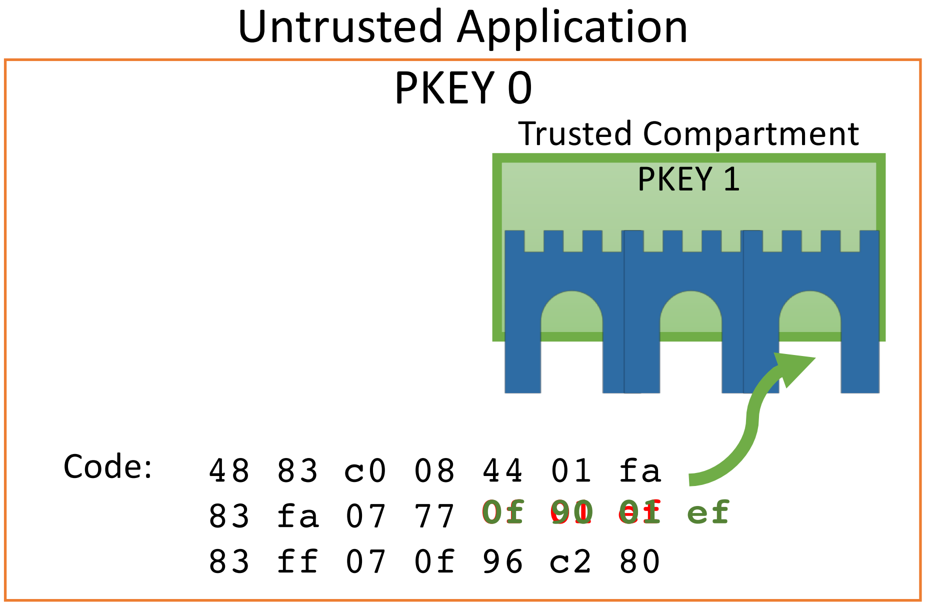
PKEY 1

Code:

```
48 83 c0 08 44 01 fa
83 fa 07 77 0f 01 ef
83 ff 07 02 96 c2 80
```

# Overview of ERIM

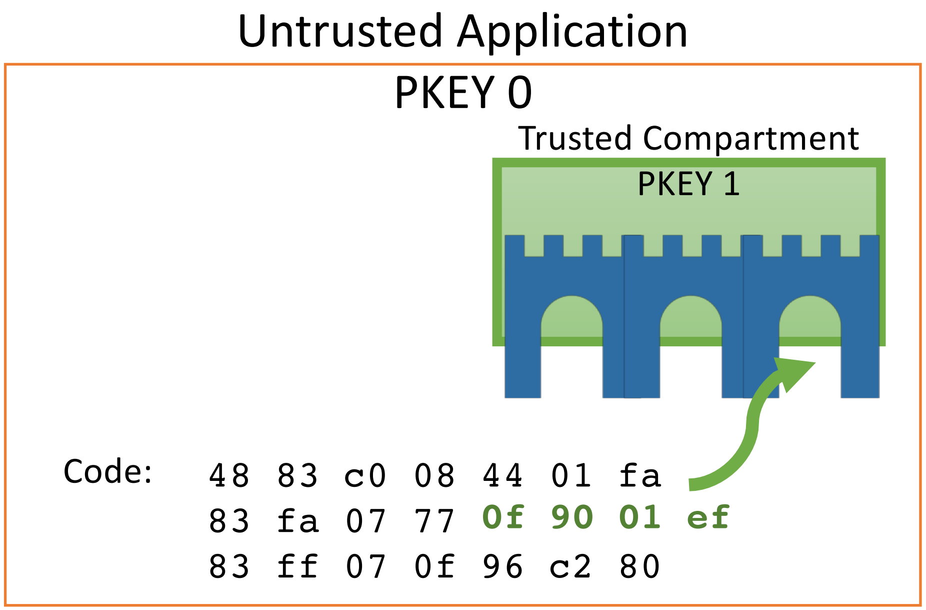
- Prevent MPK exploitation
  - Safe call gates
  - Prevent execution of permission register updates outside of call gates
- Creating usable binaries
  - Inadvertent PKRU update instruction
  - Rewrite strategy



```
48 83 c0 08 44 01 fa
83 fa 07 77 0f 90 01 ef
83 ff 07 0f 96 c2 80
```

# Overview of ERIM

- Prevent MPK exploitation
  - Safe call gates
  - Prevent execution of permission register updates outside of call gates
- Creating usable binaries
  - Inadvertent PKRU update instruction
  - Rewrite strategy
- Evaluation
  - Frequently-switching use cases
  - 10% higher throughput compared to best existing technique



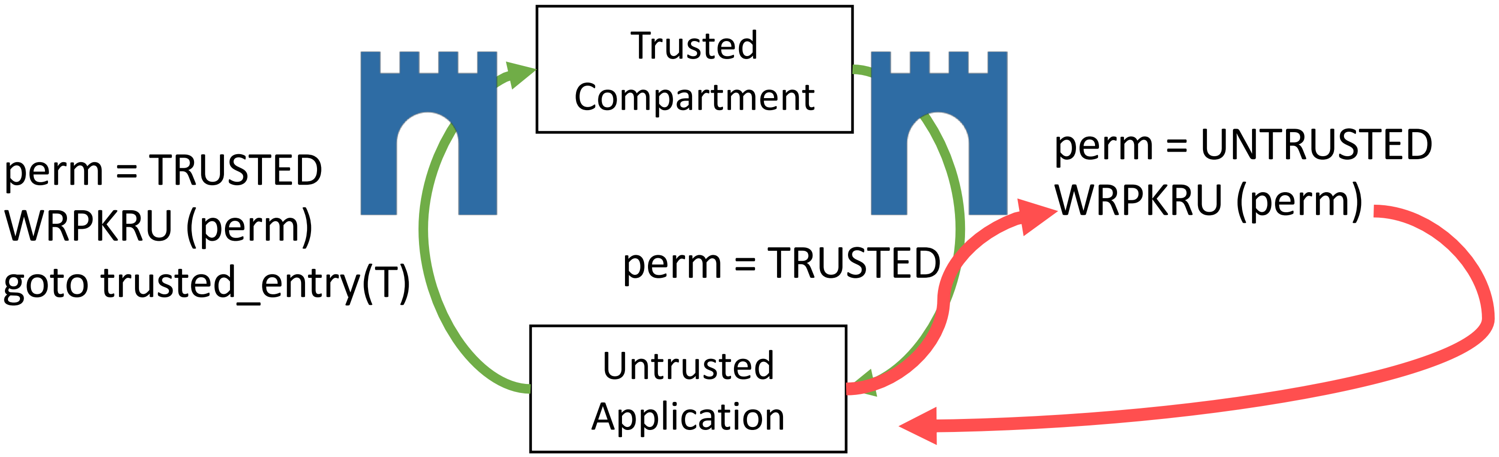
```
48 83 c0 08 44 01 fa
83 fa 07 77 0f 90 01 ef
83 ff 07 0f 96 c2 80
```

# Updating the permission in PKRU register

- WRPKRU
  - Write EAX into PKRU

- XRSTOR
  - If **bit 9** of EAX is set
  - Load PKRU register from specified memory address

# Safe switching using **call gates**



Trusted Compartment

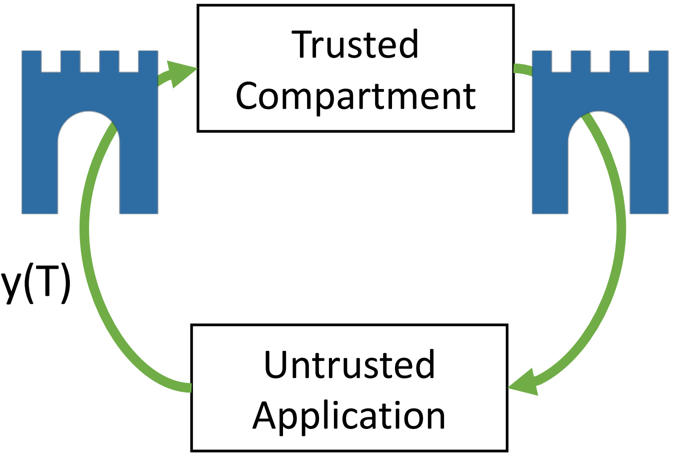
perm = TRUSTED
WRPKRU (perm)
goto trusted_entry(T)

perm = UNTRUSTED
WRPKRU (perm)

perm = TRUSTED

Untrusted Application

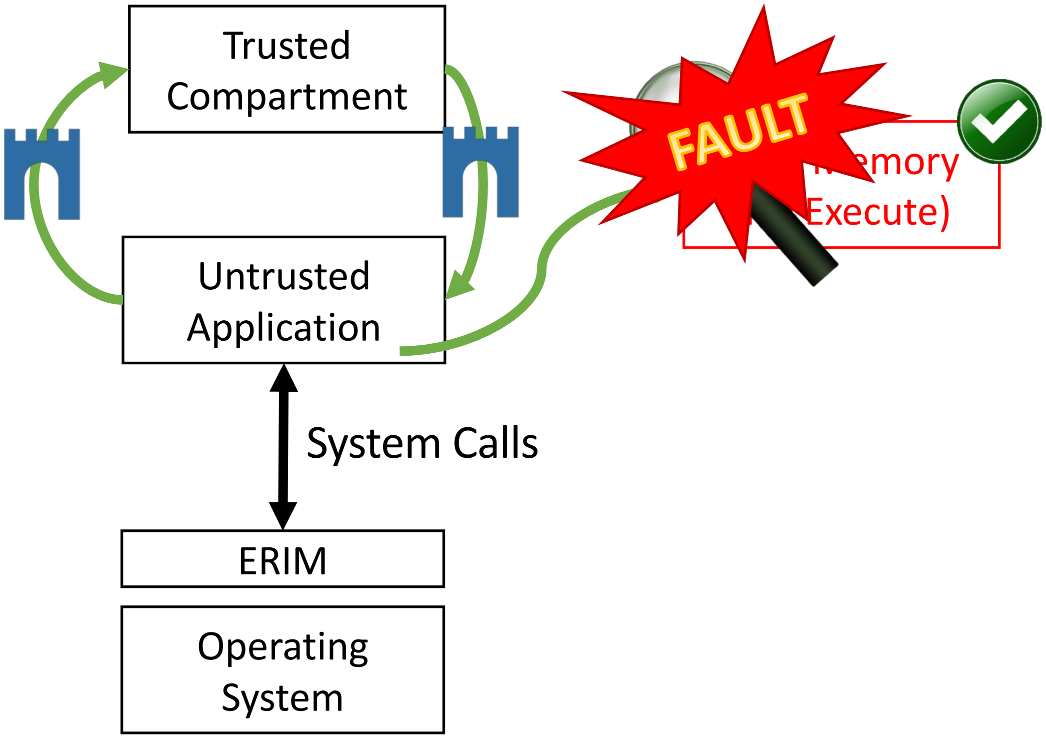
# Safe switching using **call gates**



```
perm = TRUSTED
WRPKRU (perm)
goto trusted_entry(T)
```

```
perm = UNTRUSTED
WRPKRU (perm)
if (perm != UNTRUSTED)
   exit;
```

# Prevent execution of WRPKRU/XRSTOR outside of call gates



Trusted Compartment

Untrusted Application

FAULT

Memory (Execute)

System Calls

ERIM

Operating System

Prevent execution of unvetted pages by

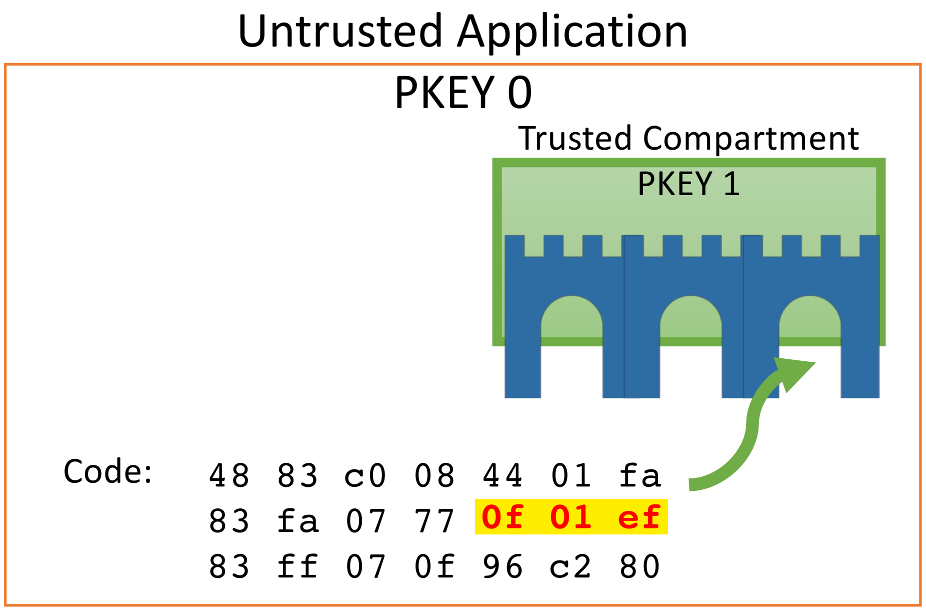
1) Monitoring system calls and removing the execute permission

2) ERIM's fault handler scans memory pages and ensures:
   - WRPKRU is part of a call gate
   - XRSTOR is followed by

```
if(eax | 0x100)
        exit();
```

# Overview of ERIM

- Prevent MPK exploitation
  - Safe call gates
  - Prevent execution of permission register updates outside of call gates
- Creating usable binaries
  - Inadvertent PKRU update instruction
  - Rewrite strategy
- Evaluation
  - Frequently-switching use cases
  - 10% higher throughput compared to best existing technique



```
48 83 c0 08 44 01 fa
83 fa 07 77 0f 01 ef
83 ff 07 0f 96 c2 80
```

# Creating usable binaries

- ERIM halts executables with inadvertent WRPKRUs/XRSTORs

**Inter-Instruction WRPKRU**

| Instruction 1 | Instruction 2 |
|---|---|
| …**0F** | **01EF**… |

**Intra-Instruction WRPKRU**

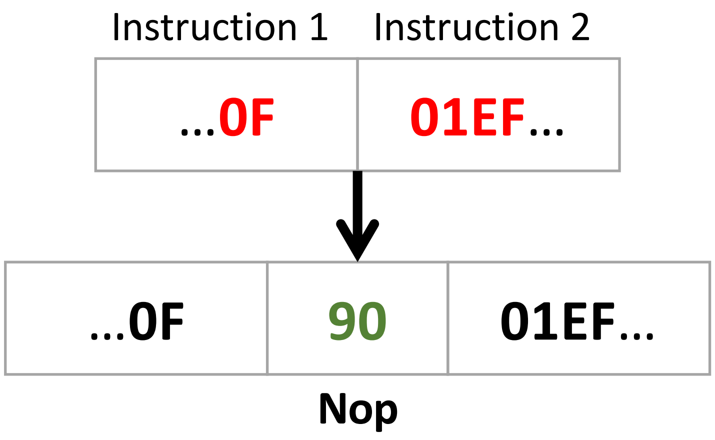
| Instruction 1 |
|---|
| 01**0F01EF**0000 |

→Eliminate inadvertent WRPKRU/XRSTOR by **binary rewriting** at
**compile** time,
**runtime** prior to enabling execute permission,
or via **static** binary rewriting for pre-compiled binaries

# Rewriting inadvertent WRPKRUs/XRSTORs

Devise rewrite rules for inadvertent WRPKRUs

**Inter-Instruction:**



Instruction 1

Instruction 2

...0F

01EF...

...0F

90

01EF...

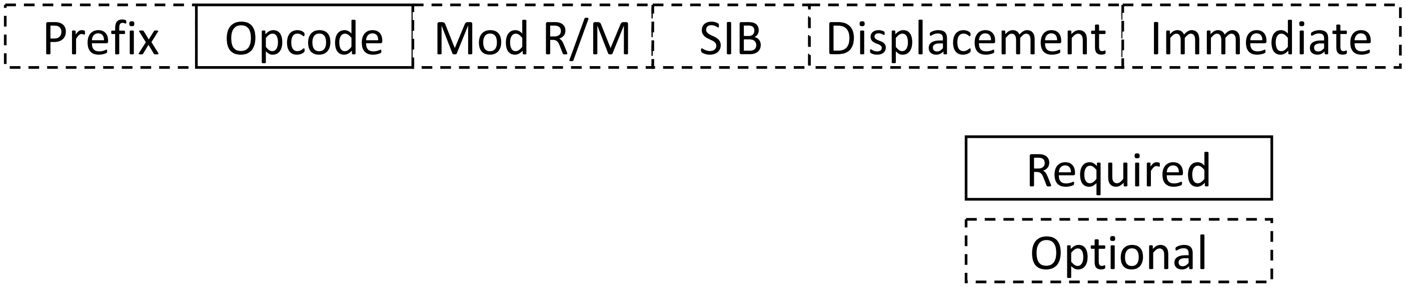
Nop

# Rewriting inadvertent WRPKRUs/XRSTORs

Devise rewrite rules for inadvertent WRPKRUs

**Intra-instruction WRPKRU**

Simplified x86 instruction format:



| Prefix | Opcode | Mod R/M | SIB | Displacement | Immediate |
|---|---|---|---|---|---|

Required

Optional

# Rewriting inadvertent WRPKRUs/XRSTORs

Devise rewrite rules for inadvertent WRPKRUs

Example rewrite rule:

add ecx, [**ebx** + **0x01EF0000**]

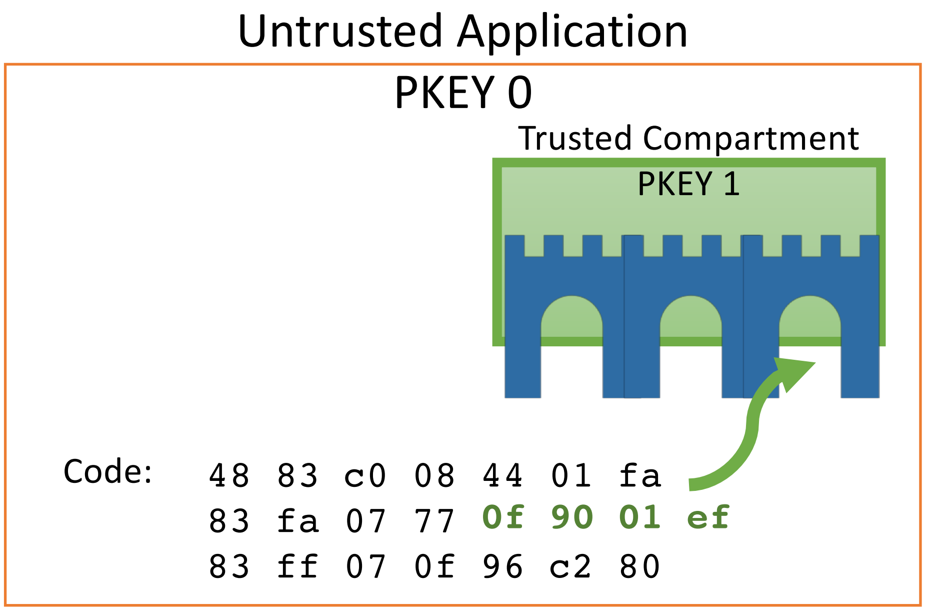
| Opcode | Mod R/M | Displacement |
|---|---|---|
| 0x01 | **0x0F** | 0x01EF0000 |

→ push eax;
mov eax, ebx;
add ecx, **[eax + 0x01EF0000]**;
pop eax;

| Opcode | Mod R/M | Displacement |
|---|---|---|
| 0x01 | **0x07** | 0x01EF0000 |

# Overview of ERIM

- Prevent MPK exploitation
  - Safe call gates
  - Prevent execution of permission register updates outside of call gates
- Creating usable binaries
  - Inadvertent PKRU update instruction
  - Rewrite strategy
- Evaluation
  - Frequently-switching use cases
  - 10% higher throughput compared to best existing technique



Untrusted Application

PKEY 0

Trusted Compartment

PKEY 1

Code:

```
48 83 c0 08 44 01 fa
83 fa 07 77 0f 90 01 ef
83 ff 07 0f 96 c2 80
```

# Prototype implementation

- ERIM userspace library
  - Call gates
  - Memory allocator for trusted component overloading malloc-like functions
  - Memory inspection (exclude unsafe WRPKRU/XRSTOR)
- Prevent execution on pages with unsafe WRPKRUs/XRSTOR
  a) P-Trace and seccomp BPF userspace monitor
  b) Linux Security Module
- Remove inadvertent WRPKRUs/XRSTORs
  - Static binary rewrite tool based on DynInst

# Evaluation

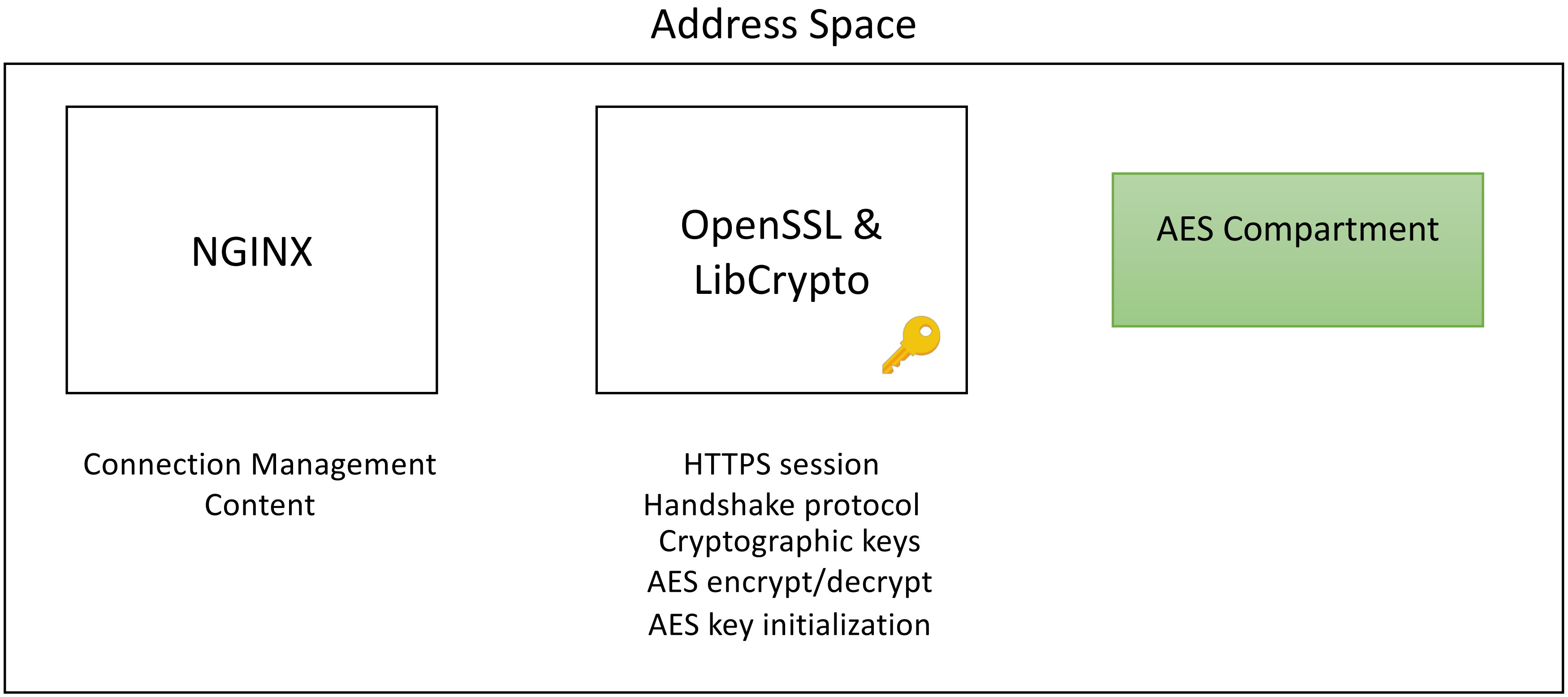
How frequent are inadvertent WRPKRUs/XRSTORs?

- Inspected about 200,000 executable files of 5 Linux distributions
- Found 1213 **inadvertent** WRPKRU/XRSTOR  in binary code
- DynInst disassembled 1,023
- 100% rewrite success

What is ERIM’s overhead in frequently-switching use cases?

- Isolating **session keys** in Nginx
- Isolating a **managed runtime** (node.js) from native libraries
- Isolating **in-memory state** of reference monitors (CPI/CPS)

# Use case: Session Key Isolation



Address Space

NGINX

OpenSSL & LibCrypto

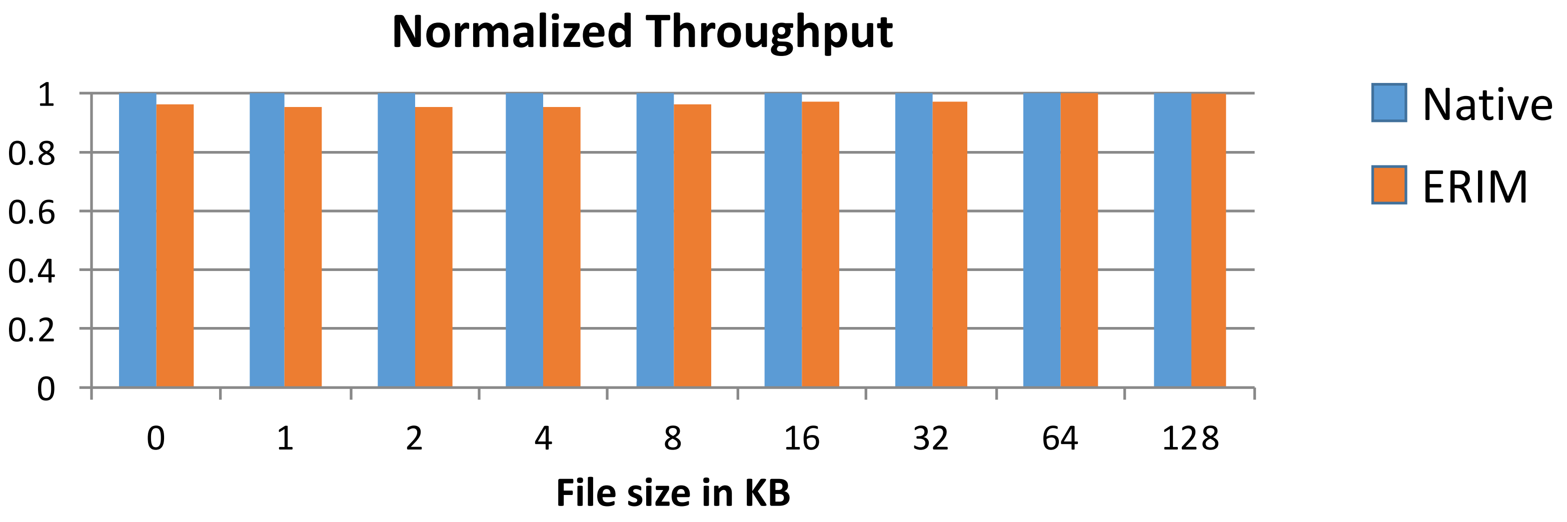
AES Compartment

Connection Management
Content

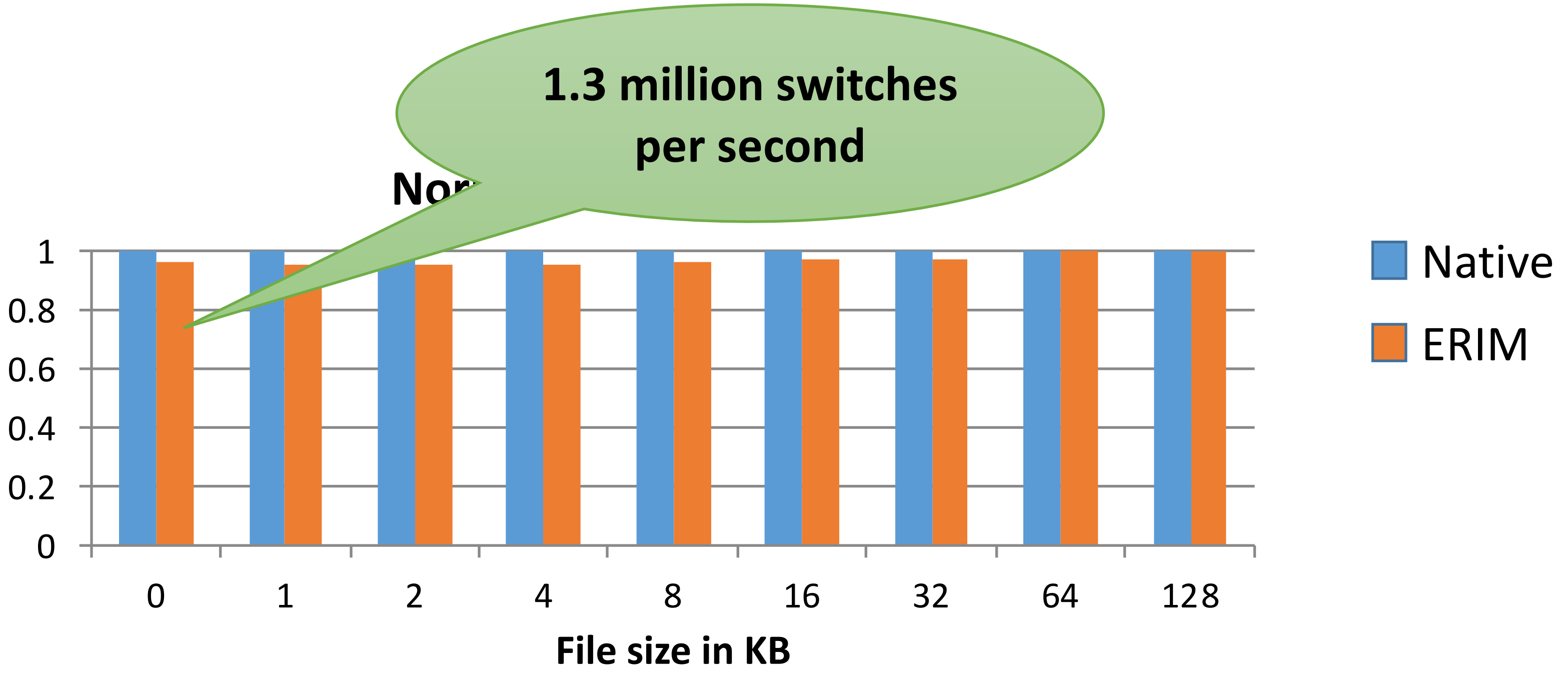
HTTPS session
Handshake protocol
Cryptographic keys
AES encrypt/decrypt
AES key initialization

# Nginx Throughput with protected session keys

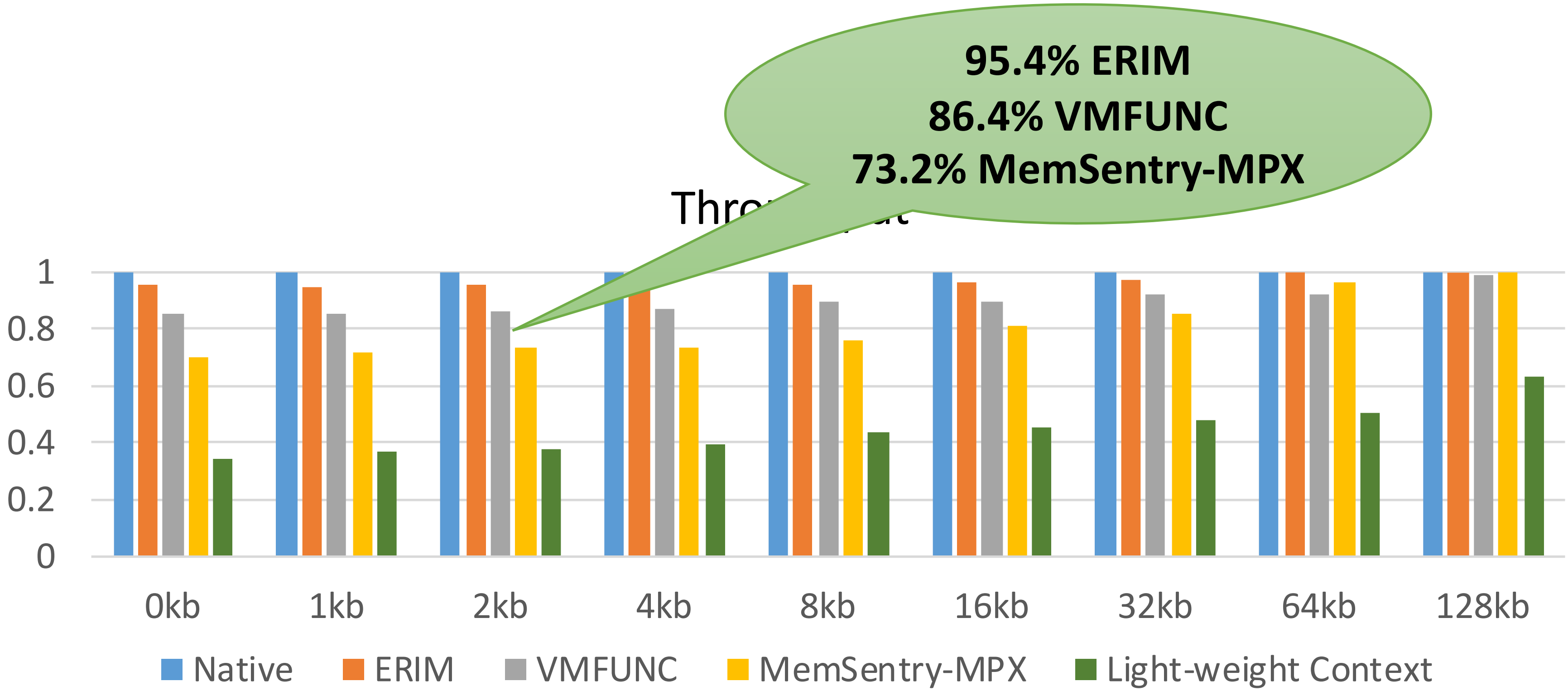
ERIM throughput within 5% of native.



# Nginx Throughput with protected session keys



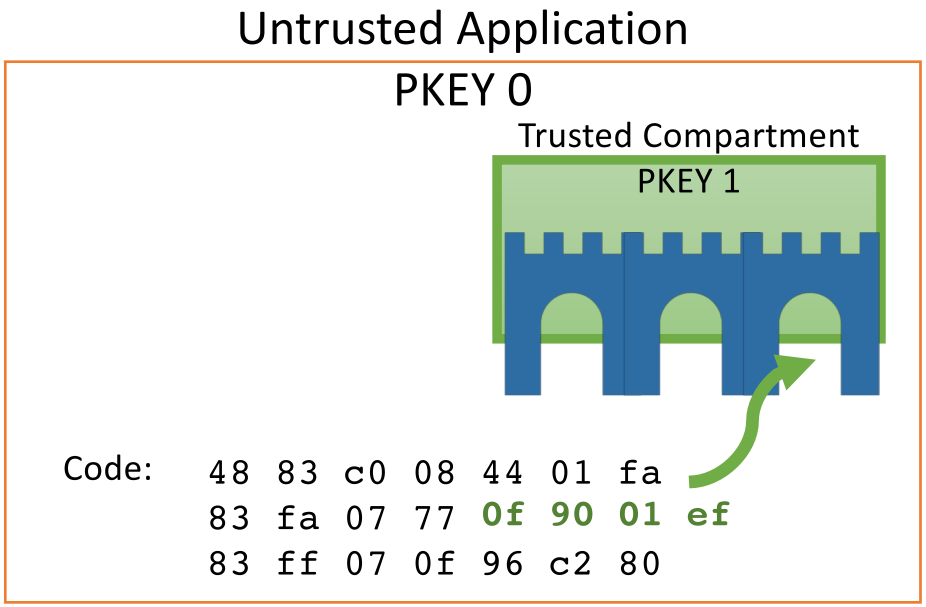
# Comparison to Prior Art



**95.4% ERIM**
**86.4% VMFUNC**
**73.2% MemSentry-MPX**

# Summary

- Prevent MPK exploitation
  - Safe call gates
  - Prevent execution of permission register updates outside of call gates
- Creating usable binaries
  - Inadvertent PKRU update instruction
  - Rewrite strategy
- Evaluation
  - Frequently-switching use cases
  - 10% higher throughput compared to best existing technique



Untrusted Application

PKEY 0

Trusted Compartment

PKEY 1

Code:

```
48 83 c0 08 44 01 fa
83 fa 07 77 0f 90 01 ef
83 ff 07 0f 96 c2 80
```

# Thank you!

MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

## ERIM: Secure, Efficient in-process Isolation with Memory Protection Keys

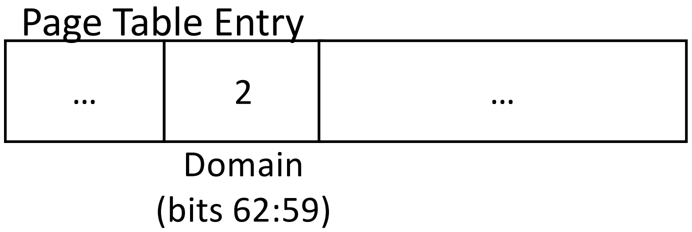
**Anjo Vahldiek-Oberwagner**, Eslam Elnikety, Nuno O. Duarte,
Michael Sammler, Peter Druschel, Deepak Garg

Code available at **https://gitlab.mpi-sws.org/vahldiek/erim**

# Backup

# Intel Memory Protection Keys (MPK)

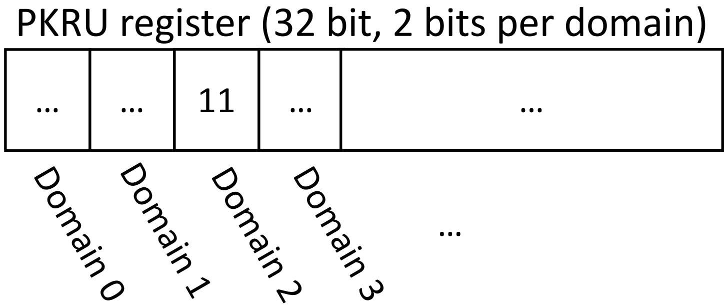
- Tag memory pages with a memory domains (bits 62:59 in page table)
- Permission register (PKRU) enables R/W to a domain
- Update accessible permissions from userspace
    - Fast switching, without context/PT switch
- By itself, protects against **bugs only**



Page Table Entry

| … | 2 | … |
|---|---|---|

Domain
(bits 62:59)



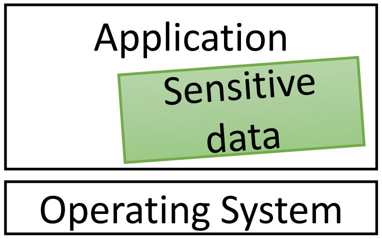
PKRU register (32 bit, 2 bits per domain)

| … | … | 11 | … | … |
|---|---|---|---|---|

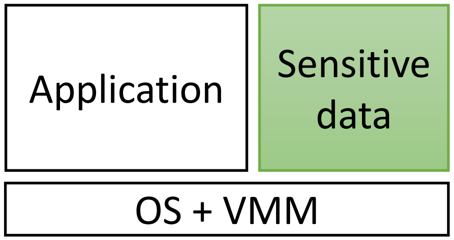
Domain 0
Domain 1
Domain 2
Domain 3
…

# State of the art: Isolating in-memory state

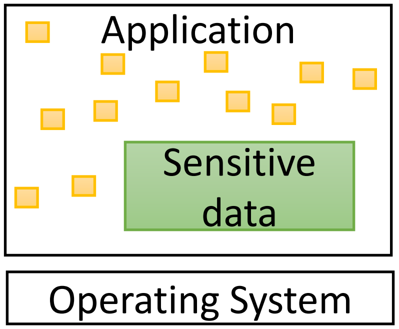
**ASLR-based Hiding**



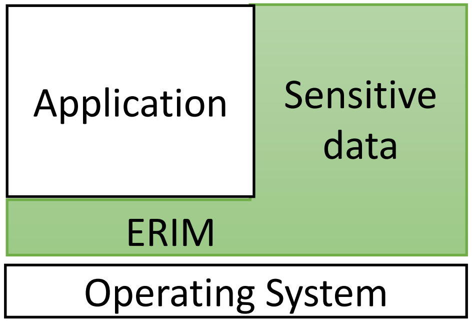
**OS/VMM-Based**



**Language and Runtime Techniques**



**ERIM: Memory Isolation using Intel MPK**



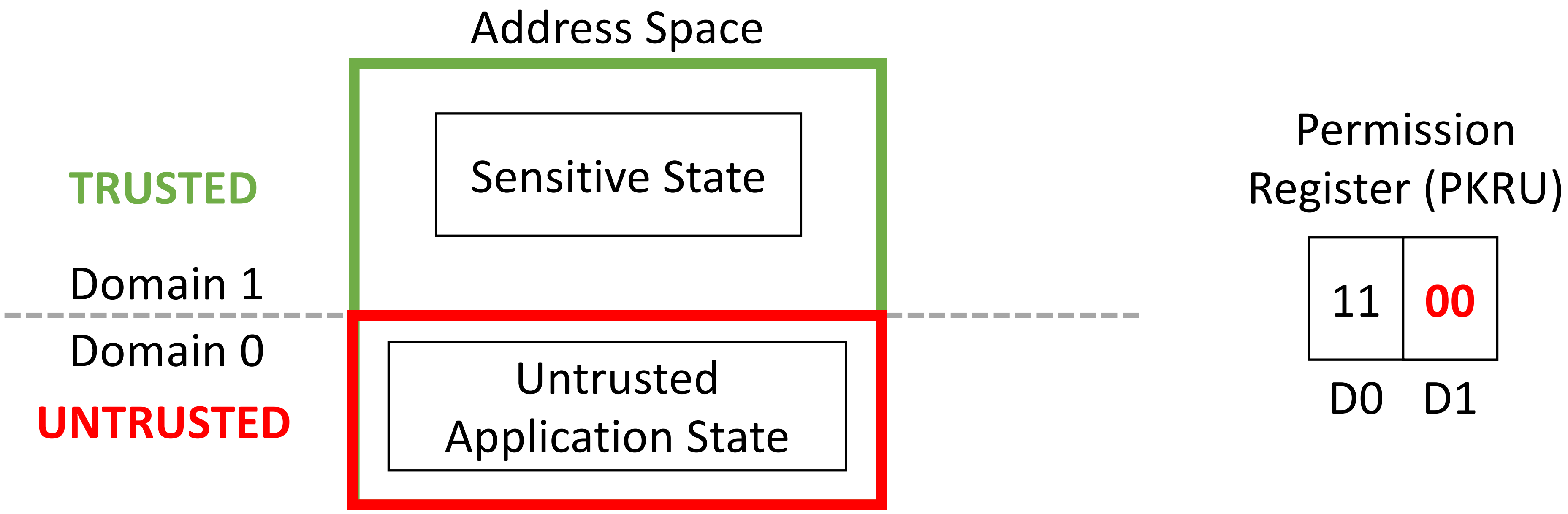
| | Execution overhead | | Switch overhead | Threat model |
|---|---|---|---|---|
| | Untrusted | Trusted | | |
| **ASLR[1]** | Low | None | None | Application bugs only |
| **OS/VMM -based[2]** | Low | Low | Medium | Any userspace |
| **Lang. & RT[3]** | Medium – High | None | None | Any userspace |
| **ERIM** | Low | None | Low | Any userspace |

[1] ASLR-Guard, Near, XnR

[2] LwC, SMVs, Shreds, Wedge, Nexen, Dune, SeCage, TrustVisor

[3] MemSentry, SFI

# Isolating sensitive state with Intel MPK



Address Space

TRUSTED

Domain 1

Sensitive State

Domain 0

UNTRUSTED

Untrusted Application State

Permission Register (PKRU)

| D0 | D1 |
|---|---|
| 11 | 00 |

Domain switch is a user-mode register write: efficient but vulnerable to attack.

# Using ERIM to isolate memory

**Inlined switches**

```
fct_A(…) {
….
switch(Trusted)

access sensitive data

switch(Untrusted)
…
}
```

**Function overwriting**

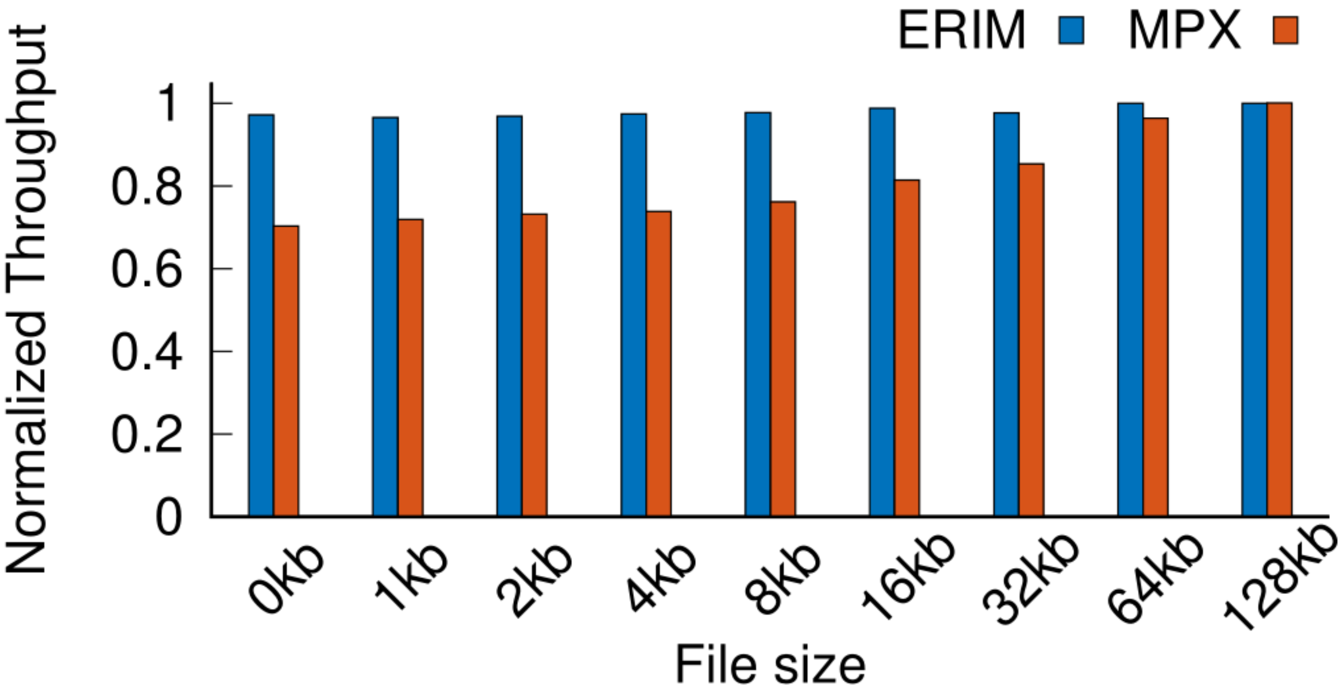
```
fct_A(…) {
….
}

BUILD_BRIDGE(fct_A);

fct_B(…) {
…
CALL_BRIDGE(fct_A, args);
…
}
```

Function **overloading** via LD_PRELOAD

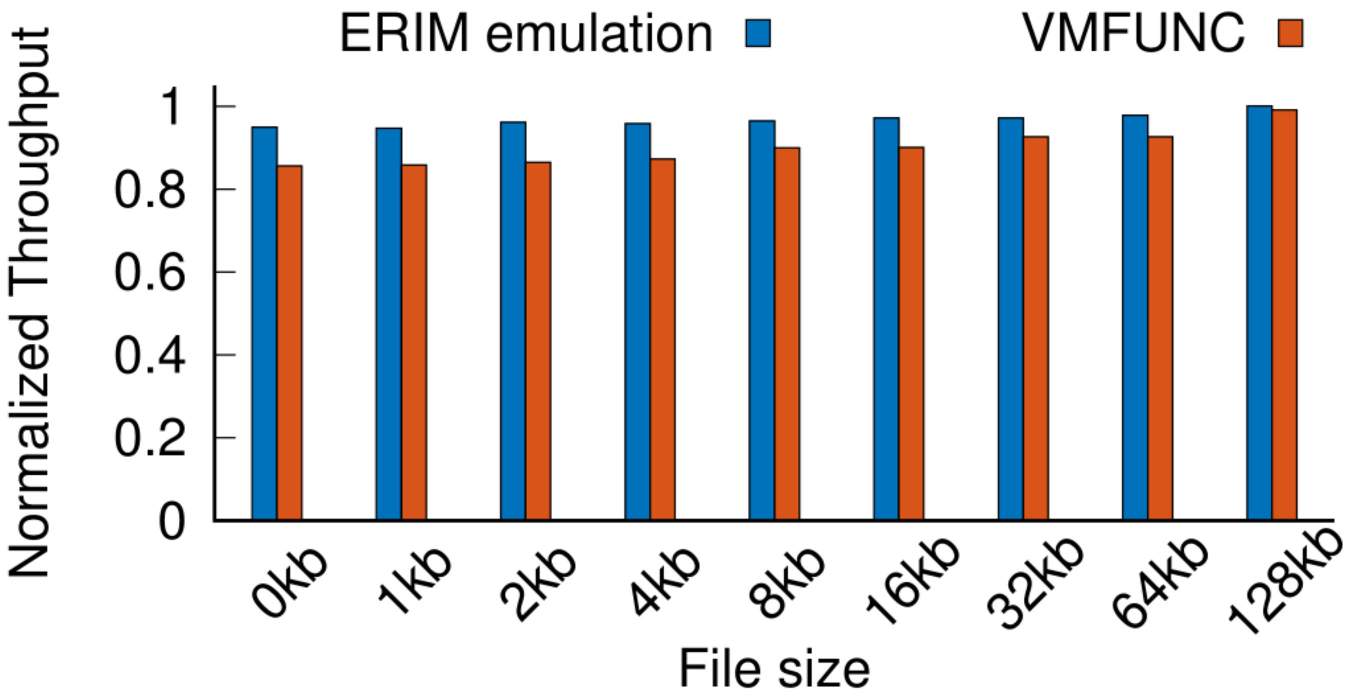
Shared library defines:

```
fct_A(…) {
f = dlsym(fct_A, …);
switch(Trusted);
ret = f(args);
switch(Untrusted);
return ret;
}
```

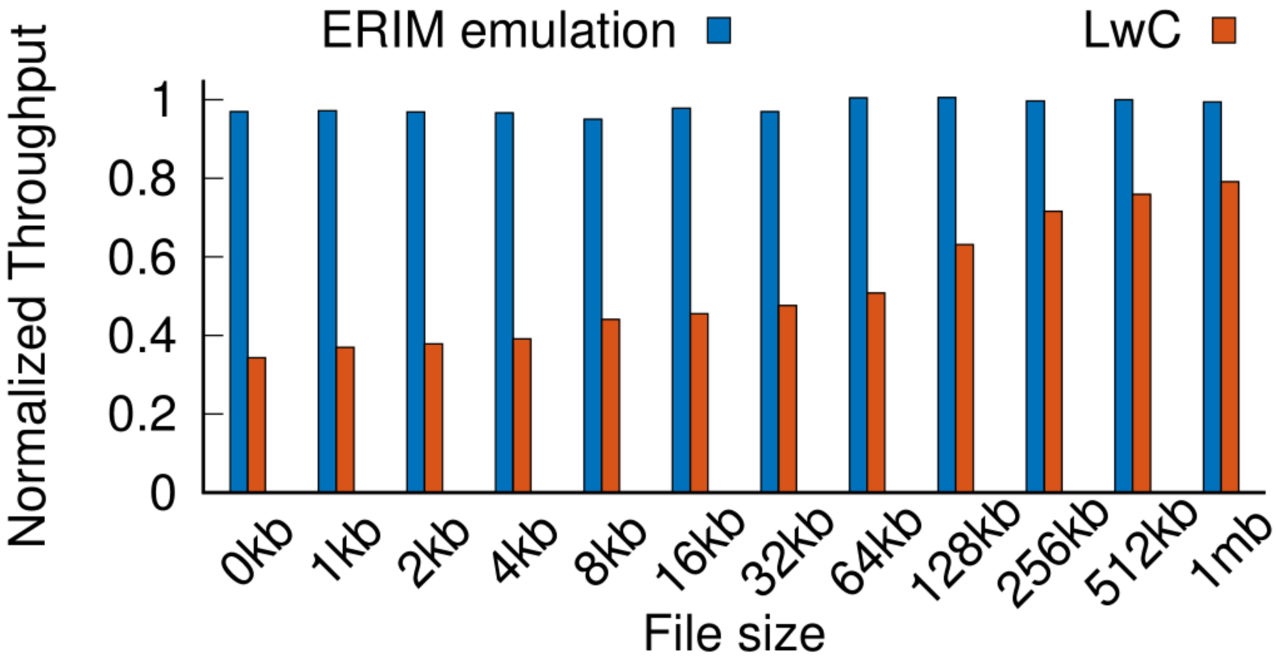
# Comparison to MPX



# Comparison to VMFUNC EPT switch



# Comparison to LwC



# How frequent are inadvertent WRPKRUs/XRSTORs?

| | Debian 8 | Ubuntu 14 | Ubuntu 16 | Gentoo | Gentoo Gold |
|---|---|---|---|---|---|
| **Elf files** | 56035 | 58548 | 69907 | 9940 | 9940 |
| **Elf files with WRPKRU/XRSTOR** | 665 | 603 | 720 | 73 | 34 |
| **Executable WRPKRU/XRSTOR** | 4244 | 1147 | 2105 | 124 | 46 |
| **WPKRU/XRSTOR in code** | 481 | 276 | 384 | 41 | 31 |
| **Disassembled by Dyninst** | 420 | 215 | 332 | 32 | 24 |
| **Inter-instruction** | 30 | 29 | 44 | 5 | 5 |
| **Intra-instruction** | 390 | 186 | 288 | 27 | 19 |

# How frequent are inadvertent WRPKRUs?

| | | Debian 8 | | | Ubuntu 14 | | | Ubuntu 16 | | | Gentoo | | | Gentoo Gold | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Elf files | | 56035 | | | 58548 | | | 69907 | | | 9940 | | | 9940 | | |
| | | All | WRPKRU | XRSTOR | All | WRPKRU | XRSTOR | All | WRPKRU | XRSTOR | All | WRPKRU | XRSTOR | All | WRPKRU | XRSTOR |
| Elf files w/ WRPKRU/XRSTOR | | 665 | 174 | 541 | 603 | 215 | 435 | 720 | 189 | 580 | 73 | 22 | 59 | 34 | 17 | 20 |
| Executable WRPKRUXRSTOR | | 4244 | 288 | 3956 | 1147 | 442 | 705 | 205 | 235 | 1870 | 124 | 26 | 98 | 46 | 18 | 28 |
| WPKRU/XRSTOR in code | | 481 | 63 | 418 | 276 | 66 | 210 | 384 | 83 | 301 | 41 | 9 | 32 | 31 | 14 | 17 |
| Disassembled by Dyninst | | 420 | 52 | 368 | 215 | 55 | 160 | 332 | 73 | 259 | 32 | 9 | 23 | 24 | 14 | 10 |
| Inter-instruction | Number | 30 | 30 | 0 | 29 | 29 | 0 | 44 | 41 | 3 | 5 | 5 | 0 | 5 | 5 | 0 |
| | Rewritable by NOP | 30 | 30 | 0 | 29 | 29 | 0 | 44 | 41 | 3 | 5 | 5 | 0 | 5 | 5 | 0 |
| Intra-instruction | Number | 390 | 22 | 368 | 186 | 26 | 160 | 288 | 32 | 256 | 27 | 4 | 23 | 19 | 9 | 10 |
| | Rewritable by rule 5 | 199 | 22 | 177 | 181 | 26 | 155 | 246 | 32 | 214 | 27 | 4 | 23 | 19 | 9 | 10 |
| | Rewritable by rule 4/6 | 191 | 0 | 194 | 5 | 0 | 5 | 42 | 0 | 42 | 0 | 0 | 0 | 0 | 0 | 0 |

# ERIM Related Work

**Hardware-based Isolation:**

- Trusted Execution Engines (TEE) [SGX, TrustZone]
- Reducing TCB of TEE [Flicker]
- Sandbox applications in TEE [Haven, Scone]

**Hypervisor/OS-based:**

- Reference monitors [Dune, Wedge, LwC]
- Sandboxing Applications [Capsicum]
- Privilege Separation [PrivTrans]
- Hiding secrets in execute-only code [Redactor, Near]

# ERIM Related Work

**Software-fault isolation:**

- Compilation-based [NativeClient]
- Emulation [Vx32]
- Just-in-time compiled languages [NativeClient++]

**Inlined Reference Monitoring:**

- Control-Flow Integrity [CPI]
- Sandboxing annotated code [Shreds]
- Intercepting Android framework [Aurasium]

# Call Gates

```
  WRPKRU (RW_TRUSTED)

// entry point to trusted

  WRPKRU (DIS_TRUSTED)
  cmp DIS_TRUSTED, EAX
  je continue
  exit
continue:
```

Elevate privileges and transfer to trusted entry point

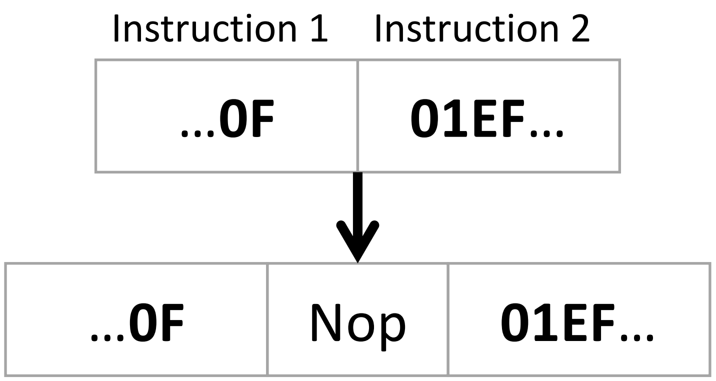
Remove privileges, check for reduced privileges and return from trusted component

# Creating safe binaries

Devise rewrite rules for WRPKRU in code segment

**Inter-instruction WRPKRU (0x0F01EF)**

Example rewrite rule:



Instruction 1 Instruction 2

**…0F** **01EF…**

**…0F** Nop **01EF…**

# Creating safe binaries

**Intra-instruction WRPKRU**

Simplified x86 instruction format:

| Prefix | Opcode | Mod R/M | SIB | Displacement | Immediate |
|---|---|---|---|---|---|

Example rewrite rule:

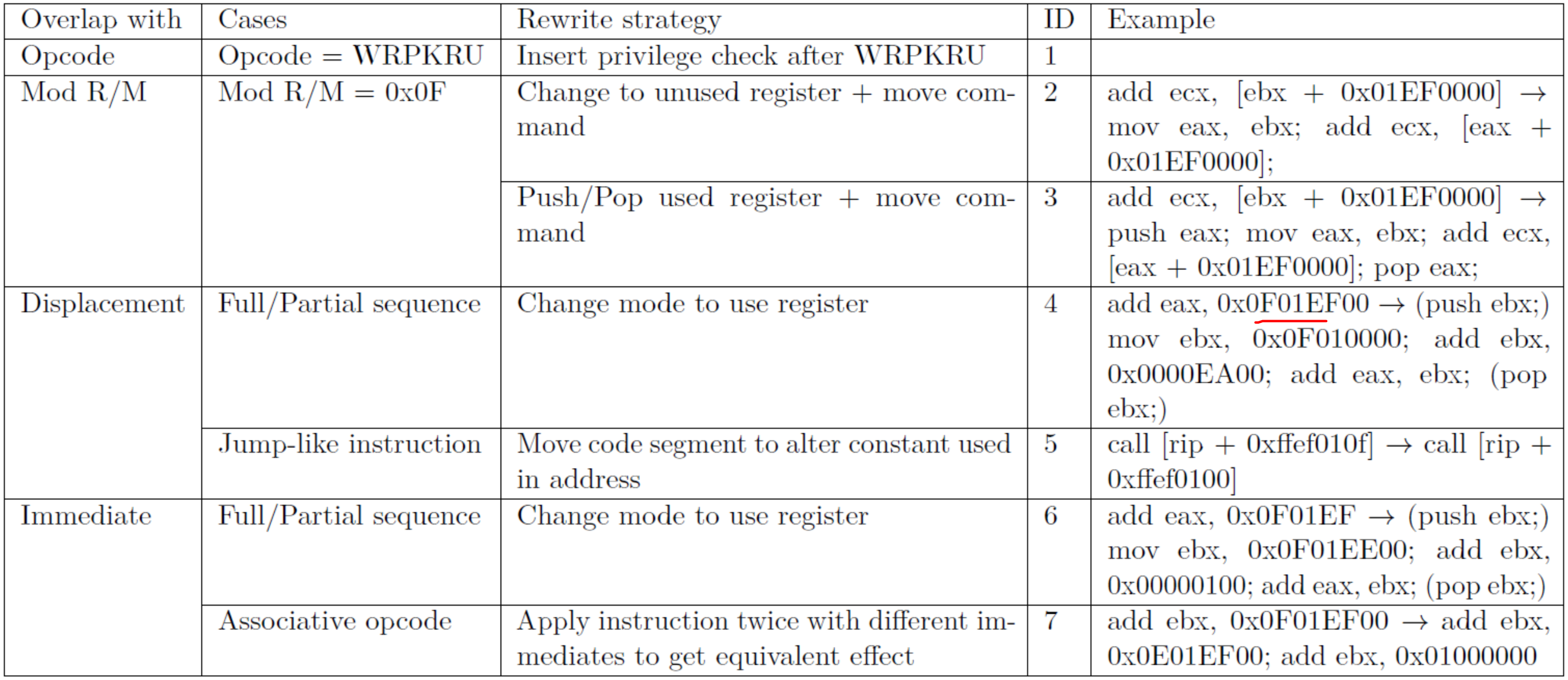
add ecx, [**ebx** + **0x01EF0000**]

| Opcode | Mod R/M | Displacement |
|---|---|---|
| 0x01 | **0x0F** | 0x01EF0000 |

→ push eax; mov eax, ebx; add ecx, **[eax + 0x01EF0000]**; pop eax;

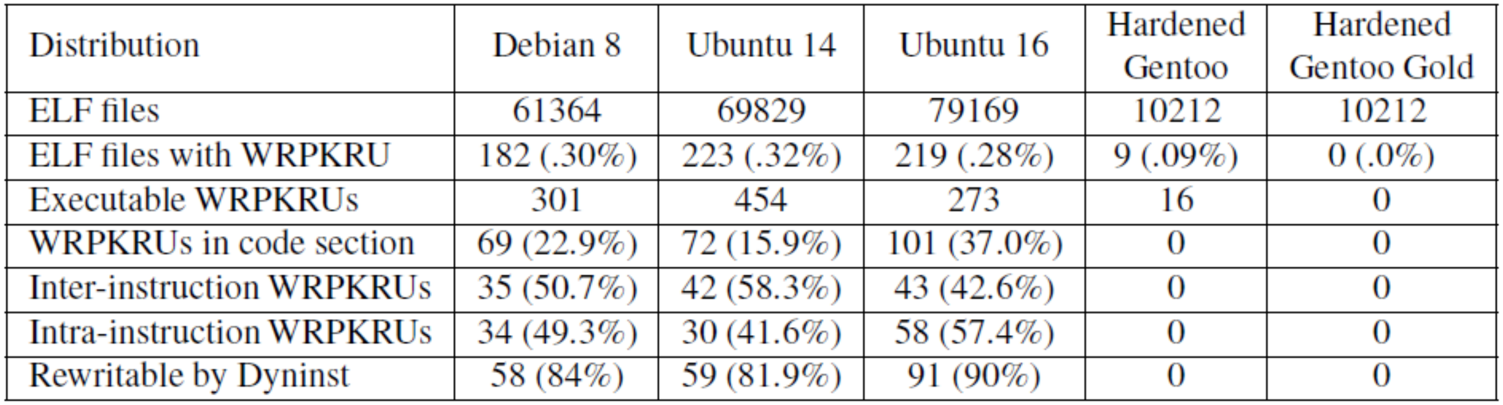
| Opcode | Mod R/M | Displacement |
|---|---|---|
| 0x01 | **0x07** | 0x01EF0000 |

# Creating safe binaries: Rewrite Rules



| Overlap with | Cases | Rewrite strategy | ID | Example |
|---|---|---|---|---|
| Opcode | Opcode = WRPKRU | Insert privilege check after WRPKRU | 1 | |
| Mod R/M | Mod R/M = 0x0F | Change to unused register + move command | 2 | add ecx, [ebx + 0x01EF0000] → mov eax, ebx; add ecx, [eax + 0x01EF0000]; |
| | | Push/Pop used register + move command | 3 | add ecx, [ebx + 0x01EF0000] → push eax; mov eax, ebx; add ecx, [eax + 0x01EF0000]; pop eax; |
| Displacement | Full/Partial sequence | Change mode to use register | 4 | add eax, 0x0F01EF00 → (push ebx;) mov ebx, 0x0F010000; add ebx, 0x0000EA00; add eax, ebx; (pop ebx;) |
| | Jump-like instruction | Move code segment to alter constant used in address | 5 | call [rip + 0xffef010f] → call [rip + 0xffef0100] |
| Immediate | Full/Partial sequence | Change mode to use register | 6 | add eax, 0x0F01EF → (push ebx;) mov ebx, 0x0F01EE00; add ebx, 0x00000100; add eax, ebx; (pop ebx;) |
| | Associative opcode | Apply instruction twice with different immediates to get equivalent effect | 7 | add ebx, 0x0F01EF00 → add ebx, 0x0E01EF00; add ebx, 0x01000000 |

# WRPKRU Occurrances



| Distribution | Debian 8 | Ubuntu 14 | Ubuntu 16 | Hardened Gentoo | Hardened Gentoo Gold |
|---|---|---|---|---|---|
| ELF files | 61364 | 69829 | 79169 | 10212 | 10212 |
| ELF files with WRPKRU | 182 (.30%) | 223 (.32%) | 219 (.28%) | 9 (.09%) | 0 (.0%) |
| Executable WRPKRUs | 301 | 454 | 273 | 16 | 0 |
| WRPKRUs in code section | 69 (22.9%) | 72 (15.9%) | 101 (37.0%) | 0 | 0 |
| Inter-instruction WRPKRUs | 35 (50.7%) | 42 (58.3%) | 43 (42.6%) | 0 | 0 |
| Intra-instruction WRPKRUs | 34 (49.3%) | 30 (41.6%) | 58 (57.4%) | 0 | 0 |
| Rewritable by Dyninst | 58 (84%) | 59 (81.9%) | 91 (90%) | 0 | 0 |

# Nginx Throughput with protected session keys

| File size | Native (req./s) | ERIM rel. (%) | Switches/s | CPU load |
|---|---|---|---|---|
| 0 | 95,761 | 95.83 | 1,342,605 | 100 |
| 1 | 87,022 | 95.18 | 1,220,266 | 100 |
| 2 | 82,137 | 95.44 | 1,151,877 | 100 |
| 4 | 76,562 | 95.25 | 1,073,843 | 100 |
| 8 | 67,855 | 95.98 | 974,780 | 100 |
| 16 | 45,483 | 97.10 | 812,173 | 100 |
| 32 | 32,381 | 97.31 | 779,141 | 100 |
| 64 | 17,827 | 100.0 | 679,371 | 96.7 |
| 128 | 8,937 | 99.99 | 556,152 | 86.4 |

CPU bound

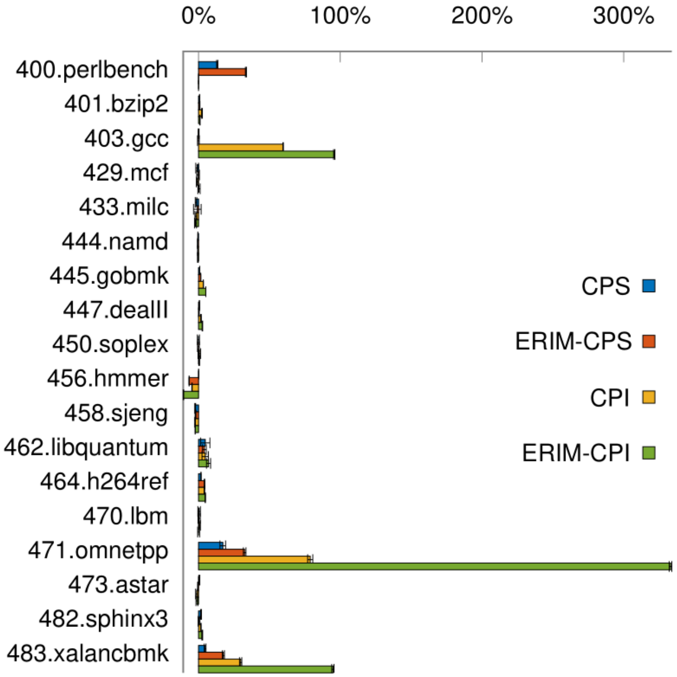
Network bound

# ERIMized C Program

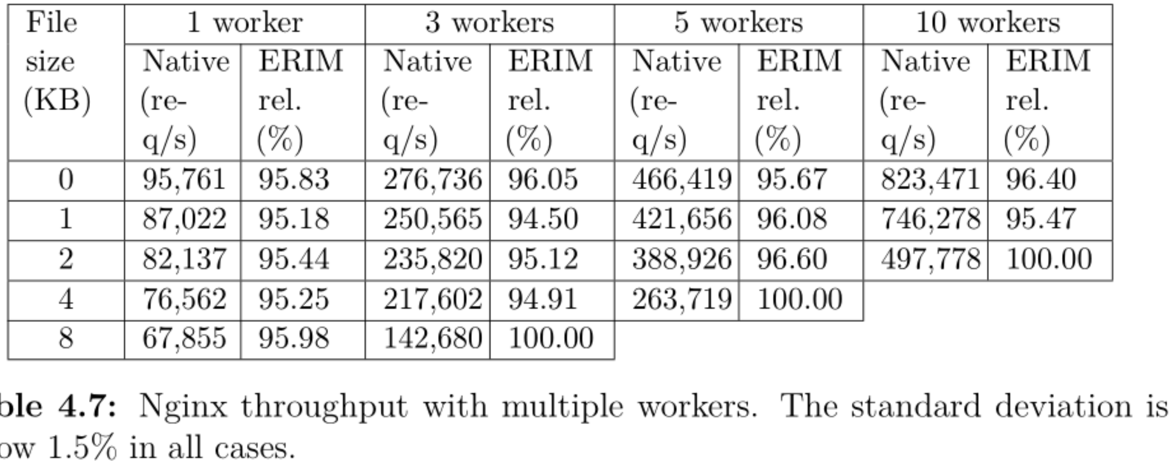
```
typedef struct secret {
  int number;
} secret;
secret* initSecret() {
  ERIM_SWITCH_T;
  secret * s = malloc(sizeof(secret));
  s->number = random();
  ERIM_SWITCH_U;
  return s;
}
```

```
int compute(secret* s, int m) {
  int ret = 0;
  ERIM_SWITCH_T;
  ret = f(s->number, m);
  ERIM_SWITCH_U;
  return ret;
}
```

# SPEC 2006 with CPS/CPI



# NGINX multiple worker



| File size (KB) | 1 worker | | 3 workers | | 5 workers | | 10 workers | |
|---|---|---|---|---|---|---|---|---|
| | Native (req/s) | ERIM rel. (%) | Native (req/s) | ERIM rel. (%) | Native (req/s) | ERIM rel. (%) | Native (req/s) | ERIM rel. (%) |
| 0 | 95,761 | 95.83 | 276,736 | 96.05 | 466,419 | 95.67 | 823,471 | 96.40 |
| 1 | 87,022 | 95.18 | 250,565 | 94.50 | 421,656 | 96.08 | 746,278 | 95.47 |
| 2 | 82,137 | 95.44 | 235,820 | 95.12 | 388,926 | 96.60 | 497,778 | 100.00 |
| 4 | 76,562 | 95.25 | 217,602 | 94.91 | 263,719 | 100.00 | | |
| 8 | 67,855 | 95.98 | 142,680 | 100.00 | | | | |

**Table 4.7:** Nginx throughput with multiple workers. The standard deviation is below 1.5% in all cases.