kR^X

Comprehensive Kernel Protection against Just-In-Time Code Reuse

Marios Pomonis 1
Theofilos Petsios 1  Angelos D. Keromytis 1
Michalis Polychronakis 2  Vasileios P. Kemerlis 3

1 Columbia University  2 Stony Brook University  3 Brown University
$> \text{whoami}$

- Ph.D. candidate @Columbia University
- Member of the Network Security Lab
  - http://nsl.cs.columbia.edu

- Research interests
  - Kernel security
  - Data-flow tracking

mpomonis@cs.columbia.edu
$> \text{whoami}$

- Ph.D. candidate @Columbia University
- Member of the Network Security Lab
  - http://nsl.cs.columbia.edu
- Research interests
  - **Kernel security**
  - Data-flow tracking
Kernel Vulnerabilities (all vendors)

Source: National Vulnerability Database (http://nvd.nist.gov)

mpomonis@cs.columbia.edu
Linux Kernel Vulnerabilities

Source: CVE Details (http://www.cvedetails.com)

mpomonis@cs.columbia.edu
Kernel Exploitation 101

▶ Userland Exploitation
  • Code Injection
  • Code Reuse
Kernel Exploitation 101

- Userland Exploitation
  - Code Injection [W^X]
  - Code Reuse [ASLR]
Kernel Exploitation 101

- Userland Exploitation
  - Code Injection [W^X]
  - Code Reuse [ASLR]
- Kernel Exploitation
Kernel Exploitation 101

- Userland Exploitation
  - Code Injection [W^X]
  - Code Reuse [ASLR]

- Kernel Exploitation
  - ret2usr
Kernel Exploitation 101

- Userland Exploitation
  - Code Injection [W^X]
  - Code Reuse [ASLR]

- Kernel Exploitation
  - ret2usr [SMEP, SMAP, ...]

Hund et al. [Oakland '13]
Jang et al. [CCS '16]
Gruss et al. [CCS '16]

mpomonis@cs.columbia.edu
Kernel Exploitation 101

- **Userland Exploitation**
  - Code Injection [W^X]
  - Code Reuse [ASLR]

- **Kernel Exploitation**
  - ret2usr [SMEP, SMAP, ...]
  - Code Injection
  - Code Reuse

mpomonis@cs.columbia.edu
Kernel Exploitation 101

- Userland Exploitation
  - Code Injection \([W^X]\)
  - Code Reuse \([\text{ASLR}]\)

- Kernel Exploitation
  - ret2usr \([\text{SMEP, SMAP, ...}]\)
  - Code Injection \([W^X]\)
  - Code Reuse \([\text{KASLR}]\)
Kernel Exploitation 101

- **Userland Exploitation**
  - Code Injection \([W^X]\)
  - Code Reuse \([\text{ASLR}]\)

- **Kernel Exploitation**
  - ret2usr \([\text{SMEP, SMAP, ...}]\)
  - Code Injection \([W^X]\)
  - Code Reuse \([\text{KASLR}]\)

Hund et al. [Oakland ’13]
Jang et al. [CCS ’16]
Gruss et al. [CCS ’16]
Code Reuse Attacks

“Offline” Code Reuse

```assembly
push %rbx
mov $0x5,%rbx
xor %rbx,%rax
pop %rbx
ret
mov $0x1,%rdi
call *%r8
jmp 0x4003e0
cmp %rsi,%rdx
jae 0x4000100
add $0x500,%rdi
jmp *%rdi
test %rax,%rax
jb 0x400043e
xor %rcx,%rcx
pop %r14
ret...
```

code pointer
Code Reuse Attacks

- “Offline” Code Reuse
  - Code snippets (gadgets)
    - Ending with an indirect branch
  - Stitch gadgets together
    - Perform arbitrary computations
Code Reuse Attacks

- “Offline” Code Reuse [Code Diversification]
  - Code snippets (gadgets)
    - Ending with an indirect branch
  - Stitch gadgets together
    - Perform arbitrary computations

```assembly
push %rbx
mov $0x5,%rbx
xor %rbx,%rax
pop %rbx
ret

mov $0x1,%rdi
call *%r8
jmp 0x4003e0
test %rax,%rax
jb 0x400043e
xor %rcx,%rcx
pop %r14
ret
cmp %rsi,%rdx
jae 0x4000100
add $0x500,%rdi
jmp *%rdi
push %rbx
mov $0x5,%rbx
xor %rbx,%rax
pop %rbx
ret ...
```

mpomonis@cs.columbia.edu

kR^X
Code Reuse Attacks

- **“Offline” Code Reuse [Code Diversification]**
  - Code snippets (**gadgets**)
    - Ending with an indirect branch
  - Stitch gadgets together
    - Perform arbitrary computations

- **“Just-In-Time” Code Reuse**
  - Direct
    - Read the (diversified) code
    - Construct the exploit on-the-fly

---

```assembly
mov $0x1,%rdi
```
```assembly
call *%r8
```
```assembly
jmp 0x4003e0
```
```assembly
test %rax,%rax
```
```assembly
jb 0x400043e
```
```assembly
xor %rcx,%rcx
```
```assembly
pop %r14
```
```assembly
ret
```
```assembly
cmp %rsi,%rdx
```
```assembly
jae 0x4000100
```
```assembly
add $0x500,%rdi
```
```assembly
jmp *%rdi
```
```assembly
```
```assembly
```
```assembly
memory disclosure
```
```assembly
```
```
```assembly
```
**Code Reuse Attacks**

- **“Offline” Code Reuse** [Code Diversification]
  - Code snippets (**gadgets**)
    - Ending with an indirect branch
  - Stitch gadgets together
    - Perform arbitrary computations

- **“Just-In-Time” Code Reuse**
  - Direct
    - Read the (diversified) code
    - Construct the exploit on-the-fly
Introduction

R^X

Fine-grained KASLR

Evaluation

Code Reuse Attacks

- **“Offline” Code Reuse** [Code Diversification]
  - Code snippets (**gadgets**)
    - Ending with an indirect branch
  - Stitch gadgets together
    - Perform arbitrary computations

- **“Just-In-Time” Code Reuse**
  - Direct
    - Read the (diversified) code
    - Construct the exploit on-the-fly
  - Indirect
    - Read code pointers from the data
    - *Infer* the randomized code layout

```
push %rbx
mov $0x5,%rbx
xor %rbx,%rax
pop %rbx
ret

mov $0x1,%rdi
call *%r8
jmp 0x4003e0
test %rax,%rax
jb 0x400043e
xor %rcx,%rcx
pop %r14
ret
cmp %rsi,%rdx
jae 0x4000100
add $0x500,%rdi
jmp *%rdi
```
kR^X

- **Comprehensive** kernel protection against code reuse attacks
  - “Offline” Code Reuse
  - JIT Code Reuse (direct/indirect)
  - No privileged entity (e.g., hypervisor)
  - Low overhead
**kR^X**

- **Comprehensive** kernel protection against code reuse attacks
  - X “Offline” Code Reuse  X JIT Code Reuse (direct/indirect)
  - No privileged entity (e.g., hypervisor)
  - Low overhead

**R^X:**

- **Execute-only Memory**
  - Separate code and data regions
    - New kernel memory layout
  - Mem. read → **range check** (RC)
    - SFI-inspired
    - ✓ Data region  X Code region
**kR^X**

- **Comprehensive** kernel protection against code reuse attacks
  - “Offline” Code Reuse
  - JIT Code Reuse (direct/indirect)
  - No privileged entity (e.g., hypervisor)
  - Low overhead

**R^X:**
- Execute-only Memory
  - Separate code and data regions
    - New kernel memory layout
  - Mem. read → range check (RC)
    - SFI-inspired
    - Data region ✓ Code region

**Fine-grained KASLR:**
- Randomized Code Layout
  - No gadgets at known location
  - High entropy → no guessing
- Return address protection
  - Encryption (XOR-based)
  - Deception (Decoys)
R^X: Memory Layout

- Multiple code sections → multiple RCs
- High overhead
- Interleaved code and data

Vmemmap space
Fixmap area
Physmap
Vmalloc arena
Kernel .text
Kernel .rodata
Kernel .data
Kernel .bss
Kernel .brk
Module1 .data
Module1 .text
Module2 .data
Module2 .text

mpomonis@cs.columbia.edu
Introduction

R^X

Fine-grained KASLR

Evaluation

R^X: Memory Layout

- Multiple code sections → multiple RCs
  - High overhead
- Interleaved code and data

mpomonis@cs.columbia.edu
R^X: Memory Layout

- Disjoint code and data regions
  - Kernel image $\rightarrow$ linker
  - Modules $\rightarrow$ module loader

- Single range check

- No code region synonyms in physmap
- No other region affected
  
  ✓ kmalloc(), vmalloc()...

mpomonis@cs.columbia.edu
R^X: Range Checks

cmpl $0x7,0x154(%rsi)
mov 0x140(%rsi),%rcx
jg L1

mov 0x130(%rsi),%rax
or $0x400000,%rax
mov %rax,%rdx
shr $0x20,%rdx
jmp L2

L1:
xor %edx,%edx
mov $0x1,%eax
wrmsr
retq

L2:

nhm_uncore_msr_enable_event()

x86-64 Linux kernel (v3.19, GCC v4.7.2)
R^X: Range Checks (-O0)

- For every memory read
  - Spill/Fill the %rflags register
  - Effective address → reserved register (%r11)
  - Compare with the end of the data region (_krx_edata)

✓ Data read
✗ Code read
  - Violation handler (krx_handler)
## Micro-benchmarks (LMBench)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>SFI(-00)</th>
</tr>
</thead>
<tbody>
<tr>
<td>syscall()</td>
<td>126.90%</td>
</tr>
<tr>
<td>open() / close()</td>
<td>306.24%</td>
</tr>
<tr>
<td>read() / write()</td>
<td>215.04%</td>
</tr>
<tr>
<td>select(10 fds)</td>
<td>119.33%</td>
</tr>
<tr>
<td>select(100 TCP fds)</td>
<td>1037.33%</td>
</tr>
<tr>
<td>fstat()</td>
<td>489.79%</td>
</tr>
<tr>
<td>mmap() / munmap()</td>
<td>180.88%</td>
</tr>
<tr>
<td>fork() + exit()</td>
<td>208.86%</td>
</tr>
<tr>
<td>fork() + execve()</td>
<td>191.83%</td>
</tr>
<tr>
<td>fork() + /bin/sh</td>
<td>113.77%</td>
</tr>
<tr>
<td>sigaction()</td>
<td>63.49%</td>
</tr>
<tr>
<td>Signal delivery</td>
<td>123.29%</td>
</tr>
<tr>
<td>Protection fault</td>
<td>13.40%</td>
</tr>
<tr>
<td>Page fault</td>
<td>202.84%</td>
</tr>
<tr>
<td>Pipe I/O</td>
<td>126.26%</td>
</tr>
<tr>
<td>UNIX socket I/O</td>
<td>148.11%</td>
</tr>
<tr>
<td>TCP socket I/O</td>
<td>171.93%</td>
</tr>
<tr>
<td>UDP socket I/O</td>
<td>208.75%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>224.89%</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipe I/O</td>
</tr>
<tr>
<td>UNIX socket I/O</td>
</tr>
<tr>
<td>TCP socket I/O</td>
</tr>
<tr>
<td>mmap() I/O</td>
</tr>
<tr>
<td>File I/O</td>
</tr>
<tr>
<td><strong>Average</strong></td>
</tr>
</tbody>
</table>
R^X: Range Checks (-O1)

- For every memory read
  - Spill/Fill the %rflags register
  - Effective address → reserved register (%r11)
  - Compare with the end of the data region (_krx_edata)

✔ Data read
✗ Code read
  - Violation handler (krx_handler)
\( \mathbb{R}^X \): Range Checks (-O1)

- For every memory read
  - Spill/Fill the %rflags register
  - pushfq/popfq Elimination [\( \sim 94\% \)]
  - Effective address \( \rightarrow \) reserved register (%r11)
  - Compare with the end of the data region (\texttt{.krx_edata})

\[ \text{Data read} \]
\[ \text{Code read} \]
  - Violation handler (\texttt{krx_handler})

\begin{verbatim}
mpomonis@cs.columbia.edu
\end{verbatim}

\[ \text{kr}^X \]

13 / 30
## Micro-benchmarks (LMBench)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>SFI(-O0)</th>
<th>SFI(-O1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>syscall()</td>
<td>126.90%</td>
<td>13.41%</td>
</tr>
<tr>
<td>open()/close()</td>
<td>306.24%</td>
<td>39.01%</td>
</tr>
<tr>
<td>read()/write()</td>
<td>215.04%</td>
<td>22.05%</td>
</tr>
<tr>
<td>select(10 fds)</td>
<td>119.33%</td>
<td>10.24%</td>
</tr>
<tr>
<td>select(100 TCP fds)</td>
<td>1037.33%</td>
<td>59.03%</td>
</tr>
<tr>
<td>fstat()</td>
<td>489.79%</td>
<td>15.31%</td>
</tr>
<tr>
<td>mmap()/munmap()</td>
<td>180.88%</td>
<td>7.24%</td>
</tr>
<tr>
<td>fork()+exit()</td>
<td>208.86%</td>
<td>14.32%</td>
</tr>
<tr>
<td>fork()+execve()</td>
<td>191.83%</td>
<td>10.30%</td>
</tr>
<tr>
<td>fork()+/bin/sh</td>
<td>113.77%</td>
<td>11.62%</td>
</tr>
<tr>
<td>sigaction()</td>
<td>63.49%</td>
<td>0.19%</td>
</tr>
<tr>
<td>Signal delivery</td>
<td>123.29%</td>
<td>18.05%</td>
</tr>
<tr>
<td>Protection fault</td>
<td>13.40%</td>
<td>1.26%</td>
</tr>
<tr>
<td>Page fault</td>
<td>202.84%</td>
<td>~0%</td>
</tr>
<tr>
<td>Pipe I/O</td>
<td>126.26%</td>
<td>22.91%</td>
</tr>
<tr>
<td>UNIX socket I/O</td>
<td>148.11%</td>
<td>12.39%</td>
</tr>
<tr>
<td>TCP socket I/O</td>
<td>171.93%</td>
<td>25.15%</td>
</tr>
<tr>
<td>UDP socket I/O</td>
<td>208.75%</td>
<td>25.71%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>224.89%</td>
<td>17.12%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipe I/O</td>
<td>46.70%</td>
</tr>
<tr>
<td>UNIX socket I/O</td>
<td>35.77%</td>
</tr>
<tr>
<td>TCP socket I/O</td>
<td>53.96%</td>
</tr>
<tr>
<td>mmap() I/O</td>
<td>~0%</td>
</tr>
<tr>
<td>File I/O</td>
<td>23.57%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>32%</td>
</tr>
</tbody>
</table>
R^X: Range Checks (-O2)

- For every memory read
  - Spill/Fill the %rflags register
  - pushfq/popfq Elimination [~94%]
  - Effective address → reserved register (%r11)

- Compare with the end of the data region (_krx_edata)

✅ Data read
❌ Code read
  - Violation handler (krx_handler)
**R^X: Range Checks (-O2)**

- For every memory read
  - **Spill/Fill the %rflags register**
  - **pushfq/popfq Elimination [~94%]**
  - **Effective address → reserved register (%r11)**
  - **lea Elimination [~95%]**
  - Compare with the end of the data region (_krx_edata)

- ![Code read](https://mpomonis.cs.columbia.edu/kR^X)
  - Violation handler (_krx_handler)
# Micro-benchmarks (LMBench)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>SFI(-00)</th>
<th>SFI(-01)</th>
<th>SFI(-02)</th>
</tr>
</thead>
<tbody>
<tr>
<td>syscall()</td>
<td>126.90%</td>
<td>13.41%</td>
<td>13.44%</td>
</tr>
<tr>
<td>open()/close()</td>
<td>306.24%</td>
<td>39.01%</td>
<td>37.45%</td>
</tr>
<tr>
<td>read()/write()</td>
<td>215.04%</td>
<td>22.05%</td>
<td>19.51%</td>
</tr>
<tr>
<td>select(10 fds)</td>
<td>119.33%</td>
<td>10.24%</td>
<td>9.93%</td>
</tr>
<tr>
<td>select(100 TCP fds)</td>
<td>1037.33%</td>
<td>59.03%</td>
<td>49.00%</td>
</tr>
<tr>
<td>fstat()</td>
<td>489.79%</td>
<td>15.31%</td>
<td>13.22%</td>
</tr>
<tr>
<td>mmap()/munmap()</td>
<td>180.88%</td>
<td>7.24%</td>
<td>6.62%</td>
</tr>
<tr>
<td>fork()+exit()</td>
<td>208.86%</td>
<td>14.32%</td>
<td>14.26%</td>
</tr>
<tr>
<td>fork()+execve()</td>
<td>191.83%</td>
<td>10.30%</td>
<td>21.75%</td>
</tr>
<tr>
<td>fork()+/bin/sh</td>
<td>113.77%</td>
<td>11.62%</td>
<td>19.22%</td>
</tr>
<tr>
<td>sigaction()</td>
<td>63.49%</td>
<td>0.19%</td>
<td>~0%</td>
</tr>
<tr>
<td>Signal delivery</td>
<td>123.29%</td>
<td>18.05%</td>
<td>16.74%</td>
</tr>
<tr>
<td>Protection fault</td>
<td>13.40%</td>
<td>1.26%</td>
<td>0.97%</td>
</tr>
<tr>
<td>Page fault</td>
<td>202.84%</td>
<td>~0%</td>
<td>~0%</td>
</tr>
<tr>
<td>Pipe I/O</td>
<td>126.26%</td>
<td>22.91%</td>
<td>21.39%</td>
</tr>
<tr>
<td>UNIX socket I/O</td>
<td>148.11%</td>
<td>12.39%</td>
<td>17.31%</td>
</tr>
<tr>
<td>TCP socket I/O</td>
<td>171.93%</td>
<td>25.15%</td>
<td>20.85%</td>
</tr>
<tr>
<td>UDP socket I/O</td>
<td>208.75%</td>
<td>25.71%</td>
<td>30.89%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>224.89%</td>
<td>17.12%</td>
<td>17.36%</td>
</tr>
<tr>
<td>Pipe I/O</td>
<td>46.70%</td>
<td>0.96%</td>
<td>1.62%</td>
</tr>
<tr>
<td>UNIX socket I/O</td>
<td>35.77%</td>
<td>3.54%</td>
<td>4.81%</td>
</tr>
<tr>
<td>TCP socket I/O</td>
<td>53.96%</td>
<td>10.90%</td>
<td>10.25%</td>
</tr>
<tr>
<td>mmap() I/O</td>
<td>~0%</td>
<td>~0%</td>
<td>~0%</td>
</tr>
<tr>
<td>File I/O</td>
<td>23.57%</td>
<td>~0%</td>
<td>~0%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>32%</td>
<td>3.08%</td>
<td>3.34%</td>
</tr>
</tbody>
</table>

mpomonis@cs.columbia.edu
R^X: Range Checks (-O3)

- For every memory read
  - Spill/Fill the %rflags register
  - `pushfq/popfq` Elimination [~94%]
  - Effective address → reserved register (%r11)
  - `lea` Elimination [~95%]
  - Compare with the end of the data region (_krx_edata)

✔ Data read
✘ Code read
  - Violation handler (krx_handler)

mpomonis@cs.columbia.edu

kR^X
R^X: Range Checks (-O3)

- For every memory read
  - cmp/ja Coalescing [\sim 50\%]
  - Spill/Fill the %rflags register
  - pushfq/popfq Elimination [\sim 94\%]
  - Effective address \rightarrow reserved register (%r11)
  - lea Elimination [\sim 95\%]
  - Compare with the end of the data region (_krx_edata)

✅ Data read
❌ Code read
  - Violation handler (_krx_handler)
## Micro-benchmarks (LMBench)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>SFI(-00)</th>
<th>SFI(-01)</th>
<th>SFI(-02)</th>
<th>SFI(-03)</th>
</tr>
</thead>
<tbody>
<tr>
<td>syscall()</td>
<td>126.90%</td>
<td>13.41%</td>
<td>13.44%</td>
<td>12.74%</td>
</tr>
<tr>
<td>open() / close()</td>
<td>306.24%</td>
<td>39.01%</td>
<td>37.45%</td>
<td>24.82%</td>
</tr>
<tr>
<td>read() / write()</td>
<td>215.04%</td>
<td>22.05%</td>
<td>19.51%</td>
<td>18.11%</td>
</tr>
<tr>
<td>select(10 fds)</td>
<td>119.33%</td>
<td>10.24%</td>
<td>9.93%</td>
<td>10.25%</td>
</tr>
<tr>
<td>select(100 TCP fds)</td>
<td>1037.33%</td>
<td>59.03%</td>
<td>49.00%</td>
<td>~0%</td>
</tr>
<tr>
<td>fstat()</td>
<td>489.79%</td>
<td>15.31%</td>
<td>13.22%</td>
<td>7.91%</td>
</tr>
<tr>
<td>mmap() / munmap()</td>
<td>180.88%</td>
<td>7.24%</td>
<td>6.62%</td>
<td>1.97%</td>
</tr>
<tr>
<td>fork() + exit()</td>
<td>208.86%</td>
<td>14.32%</td>
<td>14.26%</td>
<td>7.22%</td>
</tr>
<tr>
<td>fork() + execve()</td>
<td>191.83%</td>
<td>10.30%</td>
<td>21.75%</td>
<td>23.15%</td>
</tr>
<tr>
<td>fork() + /bin/sh</td>
<td>113.77%</td>
<td>11.62%</td>
<td>19.22%</td>
<td>12.98%</td>
</tr>
<tr>
<td>sigaction()</td>
<td>63.49%</td>
<td>0.19%</td>
<td>~0%</td>
<td>0.16%</td>
</tr>
<tr>
<td>Signal delivery</td>
<td>123.29%</td>
<td>18.05%</td>
<td>16.74%</td>
<td>7.81%</td>
</tr>
<tr>
<td>Protection fault</td>
<td>13.40%</td>
<td>1.26%</td>
<td>0.97%</td>
<td>1.33%</td>
</tr>
<tr>
<td>Page fault</td>
<td>202.84%</td>
<td>~0%</td>
<td>~0%</td>
<td>7.38%</td>
</tr>
<tr>
<td>Pipe I/O</td>
<td>126.26%</td>
<td>22.91%</td>
<td>21.39%</td>
<td>15.12%</td>
</tr>
<tr>
<td>UNIX socket I/O</td>
<td>148.11%</td>
<td>12.39%</td>
<td>17.31%</td>
<td>11.69%</td>
</tr>
<tr>
<td>TCP socket I/O</td>
<td>171.93%</td>
<td>25.15%</td>
<td>20.85%</td>
<td>16.33%</td>
</tr>
<tr>
<td>UDP socket I/O</td>
<td>208.75%</td>
<td>25.71%</td>
<td>30.89%</td>
<td>16.96%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>224.89%</td>
<td>17.12%</td>
<td>17.36%</td>
<td>10.88%</td>
</tr>
<tr>
<td>Pipe I/O</td>
<td>46.70%</td>
<td>0.96%</td>
<td>1.62%</td>
<td>0.68%</td>
</tr>
<tr>
<td>UNIX socket I/O</td>
<td>35.77%</td>
<td>3.54%</td>
<td>4.81%</td>
<td>6.43%</td>
</tr>
<tr>
<td>TCP socket I/O</td>
<td>53.96%</td>
<td>10.90%</td>
<td>10.25%</td>
<td>6.05%</td>
</tr>
<tr>
<td>mmap() I/O</td>
<td>~0%</td>
<td>~0%</td>
<td>~0%</td>
<td>~0%</td>
</tr>
<tr>
<td>File I/O</td>
<td>23.57%</td>
<td>~0%</td>
<td>~0%</td>
<td>0.67%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>32%</td>
<td>3.08%</td>
<td>3.34%</td>
<td>2.77%</td>
</tr>
</tbody>
</table>
**R^X: Range Checks (-O3)**

- For every memory read
  - `cmp/ja` Coalescing [\(\sim 50\%\)]
  - Spill/Fill the `%rflags` register
  - `pushfq/popfq` Elimination [\(\sim 94\%\)]
  - Effective address \(\rightarrow\) reserved register (%r11)
  - `lea` Elimination [\(\sim 95\%\)]
  - Compare with the end of the data region (_krx_edata_)

- Data read
- Code read
  - Violation handler (krx_handler)
R^X: Range Checks (MPX)

- New ISA extension (Intel Skylake CPUs)
  - Registers: %bnd0 – %bnd3
  - Instructions: bndcu, bndcl, bndmk...
- Hardware-assisted bounds checking
R^X: Range Checks (MPX)

- New ISA extension (Intel Skylake CPUs)
  - Registers: %bnd0 – %bnd3
  - Instructions: bndcu, bndcl, bndmk...
- *Hardware-assisted* bounds checking
- Upper bound (%bnd0.ub) → _krx_edata
  - Check before reading memory
R^X: Range Checks (MPX)

- New ISA extension (Intel Skylake CPUs)
  - Registers: %bnd0 – %bnd3
  - Instructions: bndcu, bndcl, bndmk...

- Hardware-assisted bounds checking

- Upper bound (%bnd0.ub) → _krx_edata
  - Check before reading memory
  - cmp/ja Coalescing
## Introduction

R\textsuperscript{\textregistered}X

### Fine-grained KASLR Evaluation

**Benchmark**

<table>
<thead>
<tr>
<th>Function</th>
<th>SFI(-00)</th>
<th>SFI(-01)</th>
<th>SFI(-02)</th>
<th>SFI(-03)</th>
<th>MPX</th>
</tr>
</thead>
<tbody>
<tr>
<td>syscall()</td>
<td>126.90%</td>
<td>13.41%</td>
<td>13.44%</td>
<td>12.74%</td>
<td>0.49%</td>
</tr>
<tr>
<td>open()/close()</td>
<td>306.24%</td>
<td>39.01%</td>
<td>37.45%</td>
<td>24.82%</td>
<td>3.47%</td>
</tr>
<tr>
<td>read()/write()</td>
<td>215.04%</td>
<td>22.05%</td>
<td>19.51%</td>
<td>18.11%</td>
<td>0.63%</td>
</tr>
<tr>
<td>select(10 fds)</td>
<td>119.33%</td>
<td>10.24%</td>
<td>9.93%</td>
<td>10.25%</td>
<td>1.26%</td>
</tr>
<tr>
<td>select(100 TCP fds)</td>
<td><strong>1037.33%</strong></td>
<td><strong>59.03%</strong></td>
<td><strong>49.00%</strong></td>
<td>~0%</td>
<td>~0%</td>
</tr>
<tr>
<td>fstat()</td>
<td>489.79%</td>
<td>15.31%</td>
<td>13.22%</td>
<td>7.91%</td>
<td>~0%</td>
</tr>
<tr>
<td>mmap()/munmap()</td>
<td>180.88%</td>
<td>7.24%</td>
<td>6.62%</td>
<td>1.97%</td>
<td>1.12%</td>
</tr>
<tr>
<td>fork()+exit()</td>
<td>208.86%</td>
<td>14.32%</td>
<td>14.26%</td>
<td>7.22%</td>
<td>~0%</td>
</tr>
<tr>
<td>fork()+execve()</td>
<td>191.83%</td>
<td>10.30%</td>
<td>21.75%</td>
<td>23.15%</td>
<td>~0%</td>
</tr>
<tr>
<td>fork()+/bin/sh</td>
<td>113.77%</td>
<td>11.62%</td>
<td>19.22%</td>
<td>12.98%</td>
<td><strong>6.27%</strong></td>
</tr>
<tr>
<td>sigaction()</td>
<td>63.49%</td>
<td>0.19%</td>
<td>~0%</td>
<td>0.16%</td>
<td>1.01%</td>
</tr>
<tr>
<td>Signal delivery</td>
<td>123.29%</td>
<td>18.05%</td>
<td>16.74%</td>
<td>7.81%</td>
<td>1.12%</td>
</tr>
<tr>
<td>Protection fault</td>
<td>13.40%</td>
<td>1.26%</td>
<td>0.97%</td>
<td>1.33%</td>
<td>~0%</td>
</tr>
<tr>
<td>Page fault</td>
<td>202.84%</td>
<td>~0%</td>
<td>~0%</td>
<td>7.38%</td>
<td>1.64%</td>
</tr>
<tr>
<td>Pipe I/O</td>
<td>126.26%</td>
<td>22.91%</td>
<td>21.39%</td>
<td>15.12%</td>
<td>0.42%</td>
</tr>
<tr>
<td>UNIX socket I/O</td>
<td>148.11%</td>
<td>12.39%</td>
<td>17.31%</td>
<td>11.69%</td>
<td>4.74%</td>
</tr>
<tr>
<td>TCP socket I/O</td>
<td>171.93%</td>
<td>25.15%</td>
<td>20.85%</td>
<td>16.33%</td>
<td>1.91%</td>
</tr>
<tr>
<td>UDP socket I/O</td>
<td>208.75%</td>
<td>25.71%</td>
<td>30.89%</td>
<td>16.96%</td>
<td>~0%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>224.89%</td>
<td>17.12%</td>
<td>17.36%</td>
<td>10.88%</td>
<td>1.34%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bandwidth</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipe I/O</td>
<td>46.70%</td>
<td>0.96%</td>
<td>1.62%</td>
<td>0.68%</td>
<td>~0%</td>
</tr>
<tr>
<td>UNIX socket I/O</td>
<td>35.77%</td>
<td>3.54%</td>
<td>4.81%</td>
<td><strong>6.43%</strong></td>
<td><strong>1.43%</strong></td>
</tr>
<tr>
<td>TCP socket I/O</td>
<td><strong>53.96%</strong></td>
<td><strong>10.90%</strong></td>
<td><strong>10.25%</strong></td>
<td>6.05%</td>
<td>~0%</td>
</tr>
<tr>
<td>mmap() I/O</td>
<td>~0%</td>
<td>~0%</td>
<td>~0%</td>
<td>~0%</td>
<td>~0%</td>
</tr>
<tr>
<td>File I/O</td>
<td>23.57%</td>
<td>~0%</td>
<td>~0%</td>
<td>0.67%</td>
<td>0.28%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>32%</td>
<td>3.08%</td>
<td>3.34%</td>
<td>2.77%</td>
<td>0.34%</td>
</tr>
</tbody>
</table>
Special Cases

- Safe reads [\(\sim 4\%\)]
  - \%rip-relative symbols
  - Absolute memory reads

- String operations (cmps, lods, movs)
  - Check using \%rsi
  - rep-prefixed instructions \(\rightarrow\) place RC after read operation
    - Postmortem detection
    - Allows code optimizations
Stack Reads

- offset(%rsp,%index, scale) → RC
- offset(%rsp) → no RC
  - Guard section (.krx_phantom) between .krx_edata and beginning of code
    - `sizeof(.krx_phantom) ≥ max offset`
Fine-grained KASLR

- Code block permutation

- Randomly permute the code blocks
- Preserve control flow: jmp instructions
- Unpredictable internal function layout

- Too few blocks → Phantom blocks
- Random size
- Unpredictable surrounding area of function
Fine-grained KASLR

- Code block permutation
  - Too few blocks → **Phantom blocks**
    - int3 instructions
    - Random size
Fine-grained KASLR

- Code block permutation
  - Too few blocks $\rightarrow$ **Phantom blocks**
    - int3 instructions
    - Random size
  - Randomly permute the code blocks
Fine-grained KASLR

- Code block permutation
  - Too few blocks → **Phantom blocks**
    - int3 instructions
    - Random size
  - Randomly permute the code blocks
    - Preserve control flow → jmp instructions
Fine-grained KASLR

- Code block permutation
  - Too few blocks → **Phantom blocks**
    - int3 instructions
    - Random size
  - Randomly permute the code blocks
    - Preserve control flow → jmp instructions
  - Unpredictable internal function layout

mpomonis@cs.columbia.edu
Fine-grained KASLR

- **Code block permutation**
  - Too few blocks → **Phantom blocks**
    - `int3` instructions
    - Random size
  - Randomly permute the code blocks
    - Preserve control flow → `jmp` instructions
  - Unpredictable internal function layout

- **Function permutation**
Fine-grained KASLR

- Code block permutation
  - Too few blocks → **Phantom blocks**
    - int3 instructions
    - Random size
  - Randomly permute the code blocks
    - Preserve control flow → jmp instructions
  - Unpredictable internal function layout

- Function permutation
  - Unpredictable surrounding area of function
Return Address Protection

Return Address Encryption (X)

```assembly
mov offset(%rip),%r11
xor %r11,(%rsp)
```

- XOR-based encryption
- Unique key per routine
  - Placed in the non-readable region
  - Replenished at boot/load time
Return Address Protection (cont’d)

Return Address Decoys (D)

<table>
<thead>
<tr>
<th>Decoy</th>
<th>Real</th>
<th>Decoy</th>
<th>Real</th>
</tr>
</thead>
<tbody>
<tr>
<td>push %r11</td>
<td>mov (%rsp),%rax</td>
<td>mov %r11,(%rsp)</td>
<td>push %rax</td>
</tr>
</tbody>
</table>

- Decoy return address
  - Point at phantom instructions
    - Call site → address in %r11
  - Placed before/after the real one
Phantom Instructions

49 C7 C3 CC 00 00 00 mov $0xcc, %r11

- Conceptually NOP instructions
- Contain unaligned “tripwire” opcodes
  - Raise #BR exception
- Inserted in routines’ code stream
- Address of the “tripwire” → callee
Phantom Instructions

- Conceptually NOP instructions
- Contain unaligned “tripwire” opcodes
  - Raise #BR exception
- Inserted in routines’ code stream
- Address of the “tripwire” → callee

```
49  C7  C3  CC  00  00  00
```

```
int3
mov $0xc3, %r11
```
Limitations

- Race hazards
  - Read the addresses before being encrypted/hidden among decoys
Limitations

- Race hazards
  - Read the addresses before being encrypted/hidden among decoys
  - Difficult to time reliably in the kernel setting
Limitations

- Race hazards
  - Read the addresses before being encrypted/hidden among decoys
  - Difficult to time reliably in the kernel setting
    - System-call interface
    - Process scheduling
    - Cache/TLB
Limitations

- Race hazards
  - Read the addresses before being encrypted/hidden among decoys
  - Difficult to time reliably in the kernel setting
    - System-call interface
    - Process scheduling
    - Cache/TLB

- Substitution attacks
  - Replace the (protected) return address with one of a different call-site
Limitations

- **Race hazards**
  - Read the addresses before being encrypted/hidden among decoys
  - Difficult to time reliably in the kernel setting
    - System-call interface
    - Process scheduling
    - Cache/TLB

- **Substitution attacks**
  - Replace the (protected) return address with one of a different call-site
  - Must use **valid** dynamically leaked return sites
  - Can be prevented with register randomization
## Performance Evaluation

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Metric</th>
<th>SFI</th>
<th>MPX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>Req/s</td>
<td>0.54%</td>
<td>0.48%</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>Trans/s</td>
<td>3.36%</td>
<td>1.06%</td>
</tr>
<tr>
<td>Kbuild</td>
<td>sec</td>
<td>1.48%</td>
<td>0.03%</td>
</tr>
<tr>
<td>Kextract</td>
<td>sec</td>
<td>0.52%</td>
<td>~ 0%</td>
</tr>
<tr>
<td>GnuPG</td>
<td>sec</td>
<td>0.15%</td>
<td>~ 0%</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>Sign/s</td>
<td>~ 0%</td>
<td>~ 0%</td>
</tr>
<tr>
<td>PyBench</td>
<td>msec</td>
<td>~ 0%</td>
<td>~ 0%</td>
</tr>
<tr>
<td>PHPBench</td>
<td>Score</td>
<td>0.06%</td>
<td>~ 0%</td>
</tr>
<tr>
<td>IOzone</td>
<td>MB/s</td>
<td>4.65%</td>
<td>~ 0%</td>
</tr>
<tr>
<td>DBench</td>
<td>MB/s</td>
<td>0.86%</td>
<td>~ 0%</td>
</tr>
<tr>
<td>PostMark</td>
<td>Trans/s</td>
<td>13.51%</td>
<td>1.81%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td>2.15%</td>
<td>0.45%</td>
</tr>
</tbody>
</table>

Macro-benchmarks (Phoronix Test Suite)
# Performance Evaluation

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Metric</th>
<th>SFI</th>
<th>MPX</th>
<th>SFI+D</th>
<th>SFI+X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>Req/s</td>
<td>0.54%</td>
<td>0.48%</td>
<td>0.97%</td>
<td>1.00%</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>Trans/s</td>
<td>3.36%</td>
<td>1.06%</td>
<td>6.15%</td>
<td>6.02%</td>
</tr>
<tr>
<td>Kbuild</td>
<td>sec</td>
<td>1.48%</td>
<td>0.03%</td>
<td>3.21%</td>
<td>3.50%</td>
</tr>
<tr>
<td>Kextract</td>
<td>sec</td>
<td>0.52%</td>
<td>~ 0%</td>
<td>~ 0%</td>
<td>~ 0%</td>
</tr>
<tr>
<td>GnuPG</td>
<td>sec</td>
<td>0.15%</td>
<td>~ 0%</td>
<td>0.15%</td>
<td>0.15%</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>Sign/s</td>
<td>~ 0%</td>
<td>~ 0%</td>
<td>0.03%</td>
<td>~ 0%</td>
</tr>
<tr>
<td>PyBench</td>
<td>msec</td>
<td>~ 0%</td>
<td>~ 0%</td>
<td>~ 0%</td>
<td>0.15%</td>
</tr>
<tr>
<td>PHPBench</td>
<td>Score</td>
<td>0.06%</td>
<td>~ 0%</td>
<td>0.03%</td>
<td>0.50%</td>
</tr>
<tr>
<td>IOzone</td>
<td>MB/s</td>
<td>4.65%</td>
<td>~ 0%</td>
<td>8.96%</td>
<td>8.59%</td>
</tr>
<tr>
<td>DBench</td>
<td>MB/s</td>
<td>0.86%</td>
<td>~ 0%</td>
<td>4.98%</td>
<td>~ 0%</td>
</tr>
<tr>
<td>PostMark</td>
<td>Trans/s</td>
<td>13.51%</td>
<td>1.81%</td>
<td>19.99%</td>
<td>19.98%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td>2.15%</td>
<td>0.45%</td>
<td>4.04%</td>
<td>3.63%</td>
</tr>
</tbody>
</table>

Macro-benchmarks (Phoronix Test Suite)
## Performance Evaluation

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Metric</th>
<th>SFI</th>
<th>MPX</th>
<th>SFI+D</th>
<th>SFI+X</th>
<th>MPX+D</th>
<th>MPX+X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>Req/s</td>
<td>0.54%</td>
<td>0.48%</td>
<td>0.97%</td>
<td>1.00%</td>
<td>0.81%</td>
<td>0.68%</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>Trans/s</td>
<td>3.36%</td>
<td>1.06%</td>
<td>6.15%</td>
<td>6.02%</td>
<td>3.45%</td>
<td>4.74%</td>
</tr>
<tr>
<td>Kbuild</td>
<td>sec</td>
<td>1.48%</td>
<td>0.03%</td>
<td>3.21%</td>
<td>3.50%</td>
<td>2.82%</td>
<td>3.52%</td>
</tr>
<tr>
<td>Kextract</td>
<td>sec</td>
<td>0.52%</td>
<td>~ 0%</td>
<td>~ 0%</td>
<td>~ 0%</td>
<td>~ 0%</td>
<td>~ 0%</td>
</tr>
<tr>
<td>GnuPG</td>
<td>sec</td>
<td>0.15%</td>
<td>~ 0%</td>
<td>0.15%</td>
<td>0.15%</td>
<td>~ 0%</td>
<td>~ 0%</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>Sign/s</td>
<td>~ 0%</td>
<td>~ 0%</td>
<td>0.03%</td>
<td>~ 0%</td>
<td>0.01%</td>
<td>~ 0%</td>
</tr>
<tr>
<td>PyBench</td>
<td>msec</td>
<td>~ 0%</td>
<td>~ 0%</td>
<td>~ 0%</td>
<td>0.15%</td>
<td>~ 0%</td>
<td>~ 0%</td>
</tr>
<tr>
<td>PHPBench</td>
<td>Score</td>
<td>0.06%</td>
<td>~ 0%</td>
<td>0.03%</td>
<td>0.50%</td>
<td>0.66%</td>
<td>~ 0%</td>
</tr>
<tr>
<td>IOzone</td>
<td>MB/s</td>
<td>4.65%</td>
<td>~ 0%</td>
<td>8.96%</td>
<td>8.59%</td>
<td>3.25%</td>
<td>4.26%</td>
</tr>
<tr>
<td>DBench</td>
<td>MB/s</td>
<td>0.86%</td>
<td>~ 0%</td>
<td>4.98%</td>
<td>~ 0%</td>
<td>4.28%</td>
<td>3.54%</td>
</tr>
<tr>
<td>PostMark</td>
<td>Trans/s</td>
<td>13.51%</td>
<td>1.81%</td>
<td>19.99%</td>
<td>19.98%</td>
<td>10.09%</td>
<td>12.07%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td><strong>2.15%</strong></td>
<td><strong>0.45%</strong></td>
<td><strong>4.04%</strong></td>
<td><strong>3.63%</strong></td>
<td><strong>2.32%</strong></td>
<td><strong>2.62%</strong></td>
</tr>
</tbody>
</table>

Macro-benchmarks (Phoronix Test Suite)

mpomonis@cs.columbia.edu
Conclusion

- Comprehensive solution against code reuse attacks
  - R\(^X\) (Execute-only memory)
  - Fine-grained KASLR
- Utilizes hardware assistance whenever possible
  - Memory Protection Extensions (MPX)
- Low overhead
  - SFI-based \(\rightarrow 3.63\%\)
  - MPX-based \(\rightarrow 2.32\%\)

Code available soon:
http://nsl.cs.columbia.edu/projects/krx
Backup Slides
## Micro-benchmarks (LMBench)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>SFI(-O0)</th>
<th>SFI(-O1)</th>
<th>SFI(-O2)</th>
<th>SFI(-O3)</th>
<th>MPX</th>
<th>D</th>
<th>X</th>
<th>SFI+D</th>
<th>SFI+X</th>
<th>MPX+D</th>
<th>MPX+X</th>
</tr>
</thead>
<tbody>
<tr>
<td>syscall()</td>
<td>126.90%</td>
<td>13.41%</td>
<td>13.44%</td>
<td>12.74%</td>
<td>0.49%</td>
<td>0.62%</td>
<td>2.70%</td>
<td>13.67%</td>
<td>15.91%</td>
<td>2.24%</td>
<td>2.92%</td>
</tr>
<tr>
<td>open()/close()</td>
<td>306.24%</td>
<td>39.01%</td>
<td>37.45%</td>
<td>24.82%</td>
<td>3.47%</td>
<td>15.03%</td>
<td>18.30%</td>
<td>40.68%</td>
<td>44.56%</td>
<td>19.44%</td>
<td>22.79%</td>
</tr>
<tr>
<td>read()/write()</td>
<td>215.04%</td>
<td>22.05%</td>
<td>19.51%</td>
<td>18.11%</td>
<td>0.63%</td>
<td>7.67%</td>
<td>10.74%</td>
<td>29.37%</td>
<td>34.88%</td>
<td>9.61%</td>
<td>12.43%</td>
</tr>
<tr>
<td>select(10 fds)</td>
<td>119.33%</td>
<td>10.24%</td>
<td>9.93%</td>
<td>10.25%</td>
<td>1.26%</td>
<td>3.00%</td>
<td>5.49%</td>
<td>15.05%</td>
<td>16.96%</td>
<td>4.59%</td>
<td>6.37%</td>
</tr>
<tr>
<td>select(100 TCP fds)</td>
<td><strong>1037.33%</strong></td>
<td><strong>59.03%</strong></td>
<td><strong>49.00%</strong></td>
<td><strong>10%</strong></td>
<td><strong>0%</strong></td>
<td><strong>10%</strong></td>
<td><strong>5.08%</strong></td>
<td><strong>15.05%</strong></td>
<td><strong>16.96%</strong></td>
<td><strong>4.59%</strong></td>
<td><strong>6.37%</strong></td>
</tr>
<tr>
<td>fstat()</td>
<td>489.79%</td>
<td>15.31%</td>
<td>13.22%</td>
<td>7.91%</td>
<td><strong>0%</strong></td>
<td>4.46%</td>
<td>12.92%</td>
<td>16.30%</td>
<td>26.68%</td>
<td>8.36%</td>
<td>14.64%</td>
</tr>
<tr>
<td>mmap()/munmap()</td>
<td>180.88%</td>
<td>7.24%</td>
<td>6.62%</td>
<td>1.97%</td>
<td>1.12%</td>
<td>4.83%</td>
<td>5.89%</td>
<td>7.57%</td>
<td>8.71%</td>
<td>6.86%</td>
<td>8.27%</td>
</tr>
<tr>
<td>fork()+exit()</td>
<td>208.86%</td>
<td>14.32%</td>
<td>14.26%</td>
<td>7.22%</td>
<td><strong>0%</strong></td>
<td>12.37%</td>
<td>16.57%</td>
<td>24.03%</td>
<td>21.48%</td>
<td>13.77%</td>
<td>11.64%</td>
</tr>
<tr>
<td>fork()+execve()</td>
<td>191.83%</td>
<td>10.30%</td>
<td>21.75%</td>
<td>23.15%</td>
<td><strong>0%</strong></td>
<td>13.93%</td>
<td>16.38%</td>
<td>29.91%</td>
<td>34.18%</td>
<td>17.00%</td>
<td>17.42%</td>
</tr>
<tr>
<td>fork()+/bin/sh</td>
<td>113.77%</td>
<td>11.62%</td>
<td>19.22%</td>
<td>12.98%</td>
<td><strong>6.27%</strong></td>
<td>12.37%</td>
<td>15.44%</td>
<td>23.66%</td>
<td>22.94%</td>
<td>18.40%</td>
<td>16.66%</td>
</tr>
<tr>
<td>sigaction()</td>
<td>63.49%</td>
<td>0.19%</td>
<td><strong>0%</strong></td>
<td>0.16%</td>
<td>1.01%</td>
<td>0.59%</td>
<td><strong>2.20%</strong></td>
<td>0.46%</td>
<td><strong>2.27%</strong></td>
<td>0.95%</td>
<td><strong>2.43%</strong></td>
</tr>
<tr>
<td>Signal delivery</td>
<td>123.29%</td>
<td>18.05%</td>
<td>16.74%</td>
<td>7.81%</td>
<td>1.12%</td>
<td>3.49%</td>
<td>4.94%</td>
<td>11.39%</td>
<td>13.31%</td>
<td>5.37%</td>
<td>6.52%</td>
</tr>
<tr>
<td>Protection fault</td>
<td><strong>13.40%</strong></td>
<td>1.26%</td>
<td>0.97%</td>
<td>1.33%</td>
<td><strong>0%</strong></td>
<td>1.69%</td>
<td>3.27%</td>
<td>3.34%</td>
<td>5.73%</td>
<td>1.60%</td>
<td>3.39%</td>
</tr>
<tr>
<td>Page fault</td>
<td>202.84%</td>
<td><strong>0%</strong></td>
<td><strong>0%</strong></td>
<td>7.38%</td>
<td>1.64%</td>
<td>7.83%</td>
<td>9.40%</td>
<td>15.69%</td>
<td>17.30%</td>
<td>10.80%</td>
<td>12.11%</td>
</tr>
<tr>
<td>Pipe I/O</td>
<td>126.26%</td>
<td>22.91%</td>
<td>21.39%</td>
<td>15.12%</td>
<td>0.42%</td>
<td>4.30%</td>
<td>6.89%</td>
<td>19.39%</td>
<td>22.39%</td>
<td>6.07%</td>
<td>7.62%</td>
</tr>
<tr>
<td>UNIX socket I/O</td>
<td>148.11%</td>
<td>12.39%</td>
<td>17.31%</td>
<td>11.69%</td>
<td>4.74%</td>
<td>7.34%</td>
<td>10.04%</td>
<td>16.09%</td>
<td>16.64%</td>
<td>6.88%</td>
<td>8.80%</td>
</tr>
<tr>
<td>TCP socket I/O</td>
<td>171.93%</td>
<td>25.15%</td>
<td>20.85%</td>
<td>16.33%</td>
<td>1.91%</td>
<td>4.83%</td>
<td>8.30%</td>
<td>21.63%</td>
<td>24.43%</td>
<td>8.20%</td>
<td>9.71%</td>
</tr>
<tr>
<td>UDP socket I/O</td>
<td>208.75%</td>
<td>25.71%</td>
<td>30.89%</td>
<td>16.96%</td>
<td><strong>0%</strong></td>
<td>7.38%</td>
<td>12.76%</td>
<td>24.98%</td>
<td>26.80%</td>
<td>11.22%</td>
<td>13.28%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>224.89%</strong></td>
<td><strong>17.12%</strong></td>
<td><strong>17.36%</strong></td>
<td><strong>10.88%</strong></td>
<td><strong>1.34%</strong></td>
<td><strong>6.20%</strong></td>
<td><strong>9.3%</strong></td>
<td><strong>17.5%</strong></td>
<td><strong>20.25%</strong></td>
<td><strong>8.43%</strong></td>
<td><strong>10.25%</strong></td>
</tr>
<tr>
<td>Pipe I/O</td>
<td>46.70%</td>
<td>0.96%</td>
<td>1.62%</td>
<td>0.68%</td>
<td><strong>0%</strong></td>
<td>0.59%</td>
<td>1.00%</td>
<td>2.80%</td>
<td>3.53%</td>
<td>0.78%</td>
<td>1.61%</td>
</tr>
<tr>
<td>UNIX socket I/O</td>
<td>35.77%</td>
<td>3.54%</td>
<td>4.81%</td>
<td><strong>6.43%</strong></td>
<td><strong>1.43%</strong></td>
<td>2.79%</td>
<td>3.39%</td>
<td>5.71%</td>
<td>7.00%</td>
<td>3.17%</td>
<td>3.41%</td>
</tr>
<tr>
<td>TCP socket I/O</td>
<td><strong>53.96%</strong></td>
<td><strong>10.90%</strong></td>
<td><strong>10.25%</strong></td>
<td><strong>6.05%</strong></td>
<td><strong>0%</strong></td>
<td><strong>3.71%</strong></td>
<td><strong>4.40%</strong></td>
<td><strong>9.82%</strong></td>
<td><strong>9.85%</strong></td>
<td><strong>3.64%</strong></td>
<td><strong>4.87%</strong></td>
</tr>
<tr>
<td>mmap() I/O</td>
<td><strong>0%</strong></td>
<td><strong>0%</strong></td>
<td><strong>0%</strong></td>
<td><strong>0%</strong></td>
<td><strong>0%</strong></td>
<td><strong>0%</strong></td>
<td><strong>0%</strong></td>
<td><strong>0%</strong></td>
<td><strong>0%</strong></td>
<td><strong>0%</strong></td>
<td><strong>0%</strong></td>
</tr>
<tr>
<td>File I/O</td>
<td>23.57%</td>
<td><strong>0%</strong></td>
<td><strong>0%</strong></td>
<td>0.67%</td>
<td>0.28%</td>
<td>1.21%</td>
<td>1.46%</td>
<td>1.81%</td>
<td>2.23%</td>
<td>1.74%</td>
<td>1.92%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>32%</strong></td>
<td><strong>3.08%</strong></td>
<td><strong>3.34%</strong></td>
<td><strong>2.77%</strong></td>
<td><strong>0.34%</strong></td>
<td><strong>1.66%</strong></td>
<td><strong>2.05%</strong></td>
<td><strong>4.03%</strong></td>
<td><strong>4.52%</strong></td>
<td><strong>1.87%</strong></td>
<td><strong>2.36%</strong></td>
</tr>
</tbody>
</table>