Draco: Architectural and Operating System Support for System Call Security

Dimitrios Skarlatos*, Qingrong Chen, Jianyan Chen, Tianyin Xu, Josep Torrellas
University of Illinois at Urbana-Champaign *Now at Carnegie Mellon University

International Symposium on Microarchitecture (MICRO), October 2020

Monday October 19 @ 10:45 AM CDT
Session 1A: Security & Privacy I
Conventional Cloud Computing

Virtual Machines

- App
- Lib
- Guest OS
- Hypervisor
- Hardware

Hardware

Hypervisor

Guest OS

Lib

App
New Era in Cloud Computing

1. Lightweight
2. Faster bringup
3. Higher consolidation
Security Limitations of Containers

1. Shared OS
2. Attack surface of Syscalls

Reducing the System Call Attack Surface

1. Shared OS
2. Attack surface of Syscalls

Linux Seccomp
Problem: Checks Add High Perf Overhead

1. Shared OS
2. Attack surface of Syscalls

Overhead of syscall+arg checks:
- Macro benchmarks: 14%
- Micro benchmarks: 25%

Expected future checks → 21% and 42%
Contribution: Draco Syscall Security

Insight: System Call Locality
- Proposal: To Cache Validated Syscalls

Draco SW: Augment Kernel
- Validated Argument Cache
- Overhead: 10% and 18%

Draco HW: Augment uarch
- Syscall Permissions Table
- Syscall Lookaside Buffer
- Syscall Target Buffer
- Overhead: 1%
Understanding System Calls

Syscall(#id, arg₁,...,argₙ)

<table>
<thead>
<tr>
<th>arg₁</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ...</td>
<td></td>
</tr>
<tr>
<td>2. movl 0xffffffff,%rdi</td>
<td></td>
</tr>
<tr>
<td>3. movl $135,%rax</td>
<td></td>
</tr>
<tr>
<td>4. SYSCALL</td>
<td></td>
</tr>
<tr>
<td>5. ...</td>
<td></td>
</tr>
</tbody>
</table>
Understanding System Calls

Syscall(#id, arg_1,...,arg_N)

Process

1. ...
2. movl 0xffffffff,%rdi
3. movl $135,%rax
4. SYSCALL
5. ...

Syscall #id
Understanding Seccomp

Syscall(#id, arg₁,...,argₙ)

Seccomp filters:
• Limit syscalls
• Limit args of syscalls
• Multiple 100s of lines of code
• Stateless

1. ...
2. movl 0xffffffff,%rdi
3. movl $135,%rax
4. SYSCALL
5. ...
6. ...
7. if (syscallID == 135 &&
    (arg1 == 0xffffffff ||
    arg1 == 0x20008)){
8.  return SCMP_ACT_ALLOW
9. }
10. ...
11. return SCMP_ACT_KILL_PROCESS

Carnegie Mellon University
Computer Science Department
Measuring Overhead

Intel Xeon (E5-2660 v3) @ 2.60GHz, Linux Kernel 5.3, Mitigations Off, Docker 19.03.12
Measuring Overhead

Macro Benchmarks

**Macro**: Httpd, Nginx, Redis, Elastic Search, Cassandra, MySQL, Grep, Pwgen

Intel Xeon (E5-2660 v3) @ 2.60GHz, Linux Kernel 5.3, Mitigations Off, Docker 19.03.12
Measuring Overhead

Macro: Httpd, Nginx, Redis, Elastic Search, Cassandra, MySQL, Grep, Pwgen

Intel Xeon (E5-2660 v3) @ 2.60GHz, Linux Kernel 5.3, Mitigations Off, Docker 19.03.12
Measuring Overhead

Macro: Httpd, Nginx, Redis, Elastic Search, Cassandra, MySQL, Grep, Pwgen
Micro: Unixbench Syscall, SysBench FIO, HPCC GUPS, IPCBench {fifo, pipe, mq, domain}
Intel Xeon (E5-2660 v3) @ 2.60GHz, Linux Kernel 5.3, Mitigations Off, Docker 19.03.12
Measuring Overhead

Macro:Httpd, Nginx, Redis, Elastic Search, Cassandra, MySQL, Grep, Pwgen
Micro: Unixbench Syscall, SysBench FIO, HPCC GUPS, IPCBench {fifo, pipe, mq, domain}
Intel Xeon (E5-2660 v3) @ 2.60GHz, Linux Kernel 5.3, Mitigations Off, Docker 19.03.12
Measuring Overhead

![Graph showing latency/execution time for different scenarios: Insecure, Docker, Syscall+Args.]

- **Macro:** Httpd, Nginx, Redis, Elastic Search, Cassandra, MySQL, Grep, Pwgen
- **Micro:** Unixbench Syscall, SysBench FIO, HPCC GUPS, IPCBench {fifo, pipe, mq, domain}

Intel Xeon (E5-2660 v3) @ 2.60GHz, Linux Kernel 5.3, Mitigations Off, Docker 19.03.12
Measuring Overhead

**Macro**: Httpd, Nginx, Redis, Elastic Search, Cassandra, MySQL, Grep, Pwgen
**Micro**: Unixbench Syscall, SysBench FIO, HPCC GUPS, IPCBench {fifo, pipe, mq, domain}

Intel Xeon (E5-2660 v3) @ 2.60GHz, Linux Kernel 5.3, Mitigations Off, Docker 19.03.12
Measuring Overhead

System Call Security Comes at High Cost!
Current → 14% and 25%
Future → 21% and 42%

Macro: Httpd, Nginx, Redis, Elastic Search, Cassandra, MySQL, Grep, Pwgen
Micro: Unixbench Syscall, SysBench FIO, HPCC GUPS, IPCBench {fifo, pipe, mq, domain}
Intel Xeon (E5-2660 v3) @ 2.60GHz, Linux Kernel 5.3, Mitigations Off, Docker 19.03.12
Main Idea: Cache Seccomp Results

Why Draco works?
- Seccomp is stateless
- Fall back to Seccomp upon cache miss
- Exploit the locality of system calls

```
syscall(arg1 ... argN)
```

- Check Draco Cache
- Present?
  - Yes: Proceed with the System Call
  - No: Execute the Seccomp Profile
- Allowed?
  - Yes: Update Draco Cache
  - No: Terminate User Program
Draco Overview

```plaintext
syscall(#, arg₁ ... argₙ)
```

- Application
- Allow
- Deny

Main Memory

```
Validated Argument Cache
```

Arg₁  Argₙ
Draco SW Overview

Application

Core

L1 Cache
Syscall(id, Arg₁, Argₙ)

L2 Cache

L3 Cache

Validated Argument Cache

Main Memory

Arg₁ Argₙ

Allow

Deny
Draco HW Overview

Application

Core

L1 Cache

L2 Cache

L3 Cache

Syscall Lookaside Buffer

{SID, (arg₁, argₙ)}

Main Memory

Validated Argument Cache

Arg₁ Argₙ

Carnegie Mellon University
Computer Science Department
Draco HW Overview

Application

Core

L1 Cache

L2 Cache

L3 Cache

Main Memory

Validated Argument Cache

Arg₁, Argₙ

Syscall(id, Arg₁, Argₙ)

SLB
Draco HW Overview

Application

Core

L1 Cache

L2 Cache

L3 Cache

Syscall(id, Arg₁, Argₙ)

SLB

Arg₁ Arg₂ Argₙ

Main Memory

Validated Argument Cache

Arg₁ Argₙ

Allow

Deny

Carnegie Mellon University

Computer Science Department
Draco HW Overview

Diagram showing the flow from Application to Main Memory through the following stages:

1. Core
2. L1 Cache
3. L2 Cache
4. L3 Cache
5. Syscall Lookaside Buffer
6. Validated Argument Cache

The diagram indicates the structure and flow of data from the application through the cache levels to the main memory.
Draco HW Preloading Overview

Diagram showing the preloading overview:
- Application
- Core
- Syscall Target Buffer
- L1 Cache
- L2 Cache
- L3 Cache
- Syscall Lookaside Buffer
- Validated Argument Cache
- Main Memory
- {PC, SID, ..}

Diagram depicts the flow from application to core, through various levels of caches and buffers, leading to main memory and validated argument cache.
Draco HW Preloading Overview

Main Memory

Validated Argument Cache

Application

Core

L1 Cache

L2 Cache

L3 Cache

SLB

STB
Draco HW Preloading Overview
Draco HW Preloading Overview
Evaluation Draco SW

Macro: Httpd, Nginx, Redis, Elastic Search, Cassandra, MySQL, Grep, Pwgen
Micro: Unixbench Syscall, SysBench FIO, HPCC GUPS, IPCBench {fifo, pipe, mq, domain}
Intel Xeon (E5-2660 v3) @ 2.60GHz, Linux Kernel 5.3, Mitigations Off, Docker 19.03.12
Evaluation Draco SW

Draco SW remains scalable!

Macro Benchmarks
Micro Benchmarks

Latency/Execution Time

Insecure 14% 25% 10% 18% 21% 42% 10% 23%
Syscall+Args 14% 25% 10% 18% 21% 42% 10% 23%
Draco SW 10% 23%
Syscall+Args 2x 21% 42% 10% 23%
Draco SW 2x 42% 10% 23%

Macro: Httpd, Nginx, Redis, Elastic Search, Cassandra, MySQL, Grep, Pwgen
Micro: Unixbench Syscall, SysBench FIO, HPCC GUPS, IPCBench {fifo, pipe, mq, domain}
Intel Xeon (E5-2660 v3) @ 2.60GHz, Linux Kernel 5.3, Mitigations Off, Docker 19.03.12
Evaluation Draco HW

Macro Benchmarks

Insecure: 14% 25%
Syscall+Args: 10% 18%
Draco SW: 21%
Syscall+Args 2x: 10% 23%
Draco SW 2x: 1%
Draco HW: 1%

Micro Benchmarks

Insecure: 0.0
Syscall+Args: 0.2
Draco SW: 0.4
Syscall+Args 2x: 0.6
Draco SW 2x: 0.8
Draco HW: 1.0

Macro: Httpd, Nginx, Redis, Elastic Search, Cassandra, MySQL, Grep, Pwgen
Micro: Unixbench Syscall, SysBench FIO, HPCC GUPS, IPCBench {fifo, pipe, mq, domain}
Full System Simulation with Simics + SST
Evaluation Draco HW

Draco SW Reduces Syscall Security Overhead
Draco HW Eliminates it!

Macro: Httpd, Nginx, Redis, Elastic Search, Cassandra, MySQL, Grep, Pwgen
Micro: Unixbench Syscall, SysBench FIO, HPCC GUPS, IPCBench {fifo, pipe, mq, domain}
Full System Simulation with Simics + SST
More in The Paper

Draco Kernel Module Implementation
Detailed hardware design
Detailed characterization of system calls
  • Performance overhead
  • Locality of system calls and arguments
Timing, area, power of hardware structures
Security characterization
Side-channel mitigations
Takeaway:
Draco kernel module and hardware extensions minimize the overhead of system call checks

• Exploits locality in syscalls by caching validated syscalls+args
• Draco SW reduces the cost of software checks
• Draco HW is within 1% of insecure