Adapting Old Software to New Trust Models

Stephen Herwig
Content Delivery Networks host their customers’ websites
Customers share their keys with CDNs

CDN’s edge server

bank’s private key
Key sharing is widespread

Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem

Frank Cangialosi, Taejoong Chung, David Chefler, David Levin
*University of Maryland *Northeastern University *Gates University and Akamai Technologies

ABSTRACT

The prevalence of online authentication in the web ecosystem is widespread. If a host has a certificate binding its domain to a public key, and if its certificate chain can prove identity to a CA, then a user visiting the host will be presented with a certificate that, if accepted, establishes a secure channel between the user’s browser and the host. Key management for this infrastructure is complex and challenging. This paper presents an analysis of key sharing in the HTTPS ecosystem, and identifies patterns in the prevalence of key sharing.

We observe that key sharing is a widespread phenomenon. Key sharing is observed in 50% of all HTTPS certificates and in 70% of all certificates that share a common subject. Key sharing is also observed in 50% of all HTTPS sessions and in 70% of all sessions that share a common subject. Key sharing is observed in 50% of all HTTPS connections and in 70% of all connections that share a common subject.

We also observe that key sharing is a frequent phenomenon. Key sharing is observed in 50% of all HTTPS sessions and in 70% of all sessions that share a common subject. Key sharing is observed in 50% of all HTTPS connections and in 70% of all connections that share a common subject.

We conclude that key sharing is a frequent phenomenon. Key sharing is observed in 50% of all HTTPS sessions and in 70% of all sessions that share a common subject. Key sharing is observed in 50% of all HTTPS connections and in 70% of all connections that share a common subject.

1. INTRODUCTION

Public authentication infrastructure is a fundamental first step in ensuring user privacy and security on the web. In the early days, the Internet was built using public keys, and in the mid-1990s, the SSL/TLS protocol was developed to provide secure communication over the Internet. The success of SSL/TLS has led to the widespread adoption of public key infrastructure, and today, HTTPS is the protocol of choice for securing web traffic.

Despite the widespread adoption of public key infrastructure, key management remains a complex and challenging problem. The prevalence of key sharing in the HTTPS ecosystem is widespread. Key sharing is observed in 50% of all HTTPS certificates and in 70% of all certificates that share a common subject. Key sharing is also observed in 50% of all HTTPS sessions and in 70% of all sessions that share a common subject. Key sharing is observed in 50% of all HTTPS connections and in 70% of all connections that share a common subject.

We observe that key sharing is a widespread phenomenon. Key sharing is observed in 50% of all HTTPS certificates and in 70% of all certificates that share a common subject. Key sharing is also observed in 50% of all HTTPS sessions and in 70% of all sessions that share a common subject. Key sharing is observed in 50% of all HTTPS connections and in 70% of all connections that share a common subject.

We conclude that key sharing is a frequent phenomenon. Key sharing is observed in 50% of all HTTPS sessions and in 70% of all sessions that share a common subject. Key sharing is observed in 50% of all HTTPS connections and in 70% of all connections that share a common subject.

We observe that key sharing is a widespread phenomenon. Key sharing is observed in 50% of all HTTPS certificates and in 70% of all certificates that share a common subject. Key sharing is also observed in 50% of all HTTPS sessions and in 70% of all sessions that share a common subject. Key sharing is observed in 50% of all HTTPS connections and in 70% of all connections that share a common subject.

We conclude that key sharing is a frequent phenomenon. Key sharing is observed in 50% of all HTTPS sessions and in 70% of all sessions that share a common subject. Key sharing is observed in 50% of all HTTPS connections and in 70% of all connections that share a common subject.

We observe that key sharing is a widespread phenomenon. Key sharing is observed in 50% of all HTTPS certificates and in 70% of all certificates that share a common subject. Key sharing is also observed in 50% of all HTTPS sessions and in 70% of all sessions that share a common subject. Key sharing is observed in 50% of all HTTPS connections and in 70% of all connections that share a common subject.

We conclude that key sharing is a frequent phenomenon. Key sharing is observed in 50% of all HTTPS sessions and in 70% of all sessions that share a common subject. Key sharing is observed in 50% of all HTTPS connections and in 70% of all connections that share a common subject.
Key sharing is widespread

43% of the top 10k most popular websites

The web has consolidated keys in the hands of a few CDNs

Cangialosi et al., CCS 2016
Keyless SSL

Introduced by Cloudflare to mitigate key sharing
Keyless SSL

Introduced by Cloudflare to mitigate key sharing

Private keys stay at the key server (origin)
Keyless SSL

Introduced by Cloudflare to mitigate key sharing

Private keys stay at the key server (origin)
Key server performs actions requiring private key
Keyless SSL

Introduced by Cloudflare to mitigate key sharing

Private keys stay at the key server (origin)

Key server performs actions requiring private key
Keyless SSL

Introduced by Cloudflare to mitigate key sharing

Private keys stay at the key server (origin)
Key server performs actions requiring private key
Keyless SSL

Introduced by Cloudflare to mitigate key sharing

Private keys stay at the key server (origin)
Key server performs actions requiring private key
Outsourcing Applications

HTTPS
DNS
Mail

Application
Legacy software being used in new ways

- HTTPS
- DNS
- Mail

Application

Give up

Re-write

Difficult

Time-intensive

Repeat for each new posture

Boureanu et al., Trustcom 2020

Bhargavan et al., EuroS&P 2017
Thesis Statement

It is possible to run legacy application binaries with confidentiality and integrity guarantees that reflect a multi-party trust setting.
Thesis Statement

It is possible to run legacy application binaries with confidentiality and integrity guarantees that reflect a multi-party trust setting.

Parties cannot leak data and cannot tamper with another’s. Application needs resources from multiple parties.
Threat Model

Assumptions

Trusted application

No side channels
  (control-flow, micro-architectural, etc.)
Threat Model

**Assumptions**
- Trusted application
- No side channels (control-flow, micro-architectural, etc.)

**Byzantine faulty**
- Conclaves
- Party hosting application can actively interfere with application

**Honest-but-curious**
- Conclaves
- SecureMigration
- Passive attacker
Approach

**Conclaves**
- Machine
  - Secure Hardware
    - Enclave
  - Application

**SecureMigration**
- Machine
  - Enclave
  - Application
Trusted execution environments

By default, assume all system components are untrusted.

Model: Code and data can safely reside inside an enclave.

Enclave: Isolated application memory

Application
  Code
  Enclave

Operating System
  Service

Hardware
  Small trusted CPU
  Resistant to physical attacks
Practical limitations of TEEs

Applications inside enclaves cannot make syscalls
libOSes

Idea: Implement a small “OS” inside the enclave

Enclave

Application → Code

"Syscalls"

libOS → Service

Service locally when possible

Sysscalls

Operating System → Service

Hardware
Graphene-SGX

A libOS for Intel SGX that supports some services
Graphene-SGX

A libOS for Intel SGX that supports some services

Graphene’s supported services:

- fork
- exec
- pipes, signals, semaphores
Graphene-SGX

A libOS for Intel SGX that supports some services

What constitutes a CDN?

- Multiple tenants
- Needs disk
- Needs plaintext
- Needs safe storage

Graphene’s supported services:
- fork
- exec
- pipes, signals, semaphores

Also critical to a CDN:
- Reading & writing files
- Shared memory
- Access to private keys
The first truly keyless CDN

Insight: Treat enclaves like a **distributed system**
Implement services using **kernel servers**
Phoenix: The first truly keyless CDN

Conclaves: Containers of enclaves

Insight: Treat enclaves like a distributed system. Implement services using kernel servers.
Insight: Treat enclaves like a *distributed system*.
Implement services using *kernel servers*.

**Phoenix**

The first truly keyless CDN

**Conclaves**

Containers of enclaves
**Conclaves**  
Shared memory

- **Enclave**
  - **Application**
    - Code
  - **libOS**
    - Service

- **Operating System**
  - Service

- **Hardware**
  - 

- **Enclave Memory Server**
  - Enclave
**Conclaves**

**Shared memory**

- **Enclave**
  - Application
  - fcntl()
  - "Syscall"
- **libOS**
  - Service
- **RPC**
- **Memory Server**
  - Coordinates locks
  - Maintains memory locations

**Operating System**
- Service

**Hardware**
- Disk

---

Application

libOS

"Syscall"

fcntl()

RPC

Memory Server

Coordinates locks
Maintains memory locations

---

31
Conclaves

Shared memory

Enclave

Application

libOS

"Syscall"

Service

Syscall

Operating System

Hardware

fcntl()

Memory Server

Coordinates locks

Maintains memory locations
Shared memory

Application

fcntl()

"Syscall"

libOS

Service

Syscall

Operating System

Service

Hardware

Memory file

Encrypted on untrusted disk

Enclave

Memory Server

Coordinates locks
Maintains memory locations
Enclave

Application  `fcntl()`  Shared Memory

libOS  Service

Operating System  Service

Hardware

Memory file
Encrypted on untrusted disk

Enclave

Memory Server

Coordinates locks
Maintains memory locations
Enclave

Web server
Cache
Web Application
Firewall

Enclave

Key Server
Memory Server

Phoenix The first truly keyless CDN

Conclaves Containers of enclaves

Insight: Treat enclaves like a distributed system
Implement services using kernel servers
Phoenix

The first truly keyless CDN

Conclaves

Containers of enclaves

Enclave

Web server

Cache

Web Application Firewall

Enclave

Key Server

Memory Server

File Server

Insight: Treat enclaves like a distributed system
Implement services using kernel servers
Phoenix - The first truly keyless CDN

Conclaves - Containers of enclaves

Insight: Treat enclaves like a distributed system
Implement services using kernel servers
Conclaves

File system access

Enclave

Application  □ Code

libOS  □ Service

Enclave  □ File Server

Operating System  □ Service

Hardware  □

Application

libOS

Operating System

Hardware
Conclaves

File system access

Enclave

Application

Code

libOS

Service

File Server

Operating System

Service

Hardware

Merkle Tree

Encrypted on untrusted disk
Conclaves

File system access

Application

read()

"Syscall"

libOS

Service

RPC

Enclave

File Server

Operating System

Service

Hardware

Merkle Tree
Encrypted on untrusted disk
Conclaves File system access

Application

libOS

Service

"Syscall"

RPC

ext2fs server

Block layer

Merkle root

Enclave

Verifies branches

Decrypts blocks

Operating System

Service

Syscall

Merkle Tree

Encrypted on untrusted disk

Hardware

Enclave

libOS

42
Conclaves

File system access

**Enclave**

- **Application**
  - `read()`
  - **Data**

- **libOS**
  - **Service**

**Operating System**

- **Service**

**Hardware**

- **Merkle Tree**
  - Encrypted on untrusted disk

**Enclave**

- **ext2fs server**
- **Block layer**
  - **Merkle root**
- **libOS**

Verifies branches
Decrpyts blocks
Execution environment is a distributed system of enclaves

Phoenix

The first truly keyless CDN

Conclaves

Containers of enclaves
Execution environment is a distributed system of enclaves
The first truly keyless CDN

Containers of enclaves

Conclaves’ supported services:
- fork
- exec
- pipes, signals, semaphores
- Reading & writing files
- Shared memory
- Access to private keys
- Trusted time server

Execution environment is a distributed system of enclaves
Phoenix

The first truly keyless CDN

Supports multi-tenancy
Both CDN and website can store private data
What is Phoenix’s request throughput?

Fetch a file 10,000 times over non-persistent HTTPS connections from among 128 concurrent clients.
What is Phoenix’s request throughput?

Fetch a file 10,000 times over non-persistent HTTPS connections from among 128 concurrent clients.
What is Phoenix’s request throughput?

Fetch a file 10,000 times over non-persistent HTTPS connections from among 128 concurrent clients

What is Phoenix's request throughput?
How does Phoenix scale to multiple tenants?

![Bar chart showing time per request (ms) vs number of tenants for Linux (shared NGINX).]

- Time per request (ms)
- Number of tenants
- Linux (shared NGINX)

- 1 tenant: 40 ms
- 2 tenants: 264 ms
- 4 tenants: 51 ms
- 6 tenants: 264 ms
How does Phoenix scale to multiple tenants?

- **Linux (shared NGINX)**
- **Phoenix-crypt (shared nothing)**

<table>
<thead>
<tr>
<th>Number of tenants</th>
<th>Time per request (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40 ms</td>
</tr>
<tr>
<td>2</td>
<td>8 ms</td>
</tr>
<tr>
<td>4</td>
<td>16 ms</td>
</tr>
<tr>
<td>8</td>
<td>127 ms</td>
</tr>
<tr>
<td>16</td>
<td>32 ms</td>
</tr>
<tr>
<td>32</td>
<td>264 ms</td>
</tr>
<tr>
<td>64</td>
<td>48 ms</td>
</tr>
<tr>
<td>48</td>
<td>1437 ms</td>
</tr>
</tbody>
</table>
How does Phoenix scale to multiple tenants?

- Linux (shared NGINX)
- Phoenix-crypt (shared NGINX)
- Phoenix-crypt (shared nothing)

<table>
<thead>
<tr>
<th>Number of Tenants</th>
<th>Linux (shared NGINX)</th>
<th>Phoenix-crypt (shared NGINX)</th>
<th>Phoenix-crypt (shared nothing)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40 ms</td>
<td>128 ms</td>
<td>40 ms</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>127 ms</td>
<td>806 ms</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>264 ms</td>
<td>806 ms</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>1437 ms</td>
</tr>
</tbody>
</table>

Time per request (ms)
How does Phoenix scale to multiple tenants?

- **Linux (shared NGINX)**
- **Phoenix-crypt (shared NGINX)**
- **Phoenix-crypt (shared nothing)**

<table>
<thead>
<tr>
<th>Number of tenants</th>
<th>Time per request (ms)</th>
<th>Number of enclaves</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>128 ms</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>40 ms</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>128 ms</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>127 ms</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>806 ms</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>264 ms</td>
<td>18</td>
</tr>
<tr>
<td>14</td>
<td>32</td>
<td>18</td>
</tr>
<tr>
<td>18</td>
<td>1437 ms</td>
<td>18</td>
</tr>
<tr>
<td>26</td>
<td>1481 ms</td>
<td>18</td>
</tr>
</tbody>
</table>

SGX paging events

Time per request (ms)

Number of tenants

Number of enclaves
Approach

Conclaves

SecureMigration

Machine

Application

Machine
Motivating Example

- CDN Edge Server
- Web Server
- WAF rules
- private

- private key

- Customer Machine
Load WAF Rules

CDN Edge Server

Web Server

WAF rules

Web Server's Memory

Customer Machine

private key
Load Private Key

CDN Edge Server

Web Server

WAF rules

net

fs

Customer Machine

private key

Web Server’s Memory

cloaked

access key file?

migrate

checkpoint
Track Private Data

CDN Edge Server

Customer Machine

Web Server

private key

WAF rules

Web Server’s Memory

cloaked

copied
Listen for clients

CDN Edge Server

Customer Machine

Web Server

private key

Web Server’s Memory

WAF rules

net

fs

listen for clients?
TLS Connection

- **TLS Handshake**
  - **ClientHello**

- **CDN Edge Server**
  - **Web Server**
    - **WAF rules**
    - **fs**
    - **net**

- **Customer Machine**
  - **private key**

- **Web Server’s Memory**
  - **randoms, DH param**
  - **signature**

_Web Server’s Memory does not need to be cloaked_
The diagram illustrates a TLS connection involving a CDN Edge Server, a Web Server, and a Customer Machine.

- **TLS Handshake**: The process begins with the ClientHello message sent by the client to the CDN Edge Server.
- **ServerHello**: The server responds with a ServerHello message, including randoms, DH param, and signature.
- **Web Server’s Memory**: The Web Server stores the received randoms, DH param, and signature in its memory.
- **migrate**: The migrated data from the Web Server is then sent to the Customer Machine.
- **private key**: The private key is stored on the Customer Machine.

The diagram emphasizes that the signature does not need to be cloaked.
Design

Domain A

CDN Edge Server

Application

fs

net

WAF rules

Application's Memory

Domain B

Customer Machine

private key
Design

Resource policy

<table>
<thead>
<tr>
<th>resource</th>
<th>owner</th>
<th>taint-src?</th>
</tr>
</thead>
<tbody>
<tr>
<td>/srv/waf.rules</td>
<td>A</td>
<td>yes</td>
</tr>
<tr>
<td>/srv/key.pem</td>
<td>B</td>
<td>yes</td>
</tr>
<tr>
<td>&lt;fs&gt;*</td>
<td>A</td>
<td>no</td>
</tr>
<tr>
<td>&lt;networking&gt;*</td>
<td>A</td>
<td>no</td>
</tr>
</tbody>
</table>

Application's Memory

Domain A
- Application
- WAF rules
- fs
- net

Domain B
- private key

Application's Memory

64
Design

Domain A

Spin
Application
real registers
shadow registers
real page
shadow page
real page

DBI

Domain B

Resource policy

<table>
<thead>
<tr>
<th>resource</th>
<th>owner</th>
<th>taint-src?</th>
</tr>
</thead>
<tbody>
<tr>
<td>/srv/waf.rules</td>
<td>A</td>
<td>yes</td>
</tr>
<tr>
<td>/srv/key.pem</td>
<td>B</td>
<td>yes</td>
</tr>
<tr>
<td>&lt;fs&gt;*</td>
<td>A</td>
<td>no</td>
</tr>
<tr>
<td>&lt;networking&gt;*</td>
<td>A</td>
<td>no</td>
</tr>
</tbody>
</table>

Application’s Memory
## Design

### Domain A

- **Spin**
  - Application
    - real registers
    - shadow registers

- **DBI**
  - UNIX domain socket
    - System calls
  - Spry
    - UNIX domain socket
      - Checkpoint/restore RPCs
  - CRIU

### Domain B

- **Resource policy**

<table>
<thead>
<tr>
<th>resource</th>
<th>owner</th>
<th>taint-src?</th>
</tr>
</thead>
<tbody>
<tr>
<td>/srv/waf.rules</td>
<td>A</td>
<td>yes</td>
</tr>
<tr>
<td>/srv/key.pem</td>
<td>B</td>
<td>yes</td>
</tr>
<tr>
<td>&lt;fs&gt;*</td>
<td>A</td>
<td>no</td>
</tr>
<tr>
<td>&lt;networking&gt;*</td>
<td>A</td>
<td>no</td>
</tr>
</tbody>
</table>

- **Spry**
  - UNIX domain socket
    - Checkpoint/restore RPCs

- **CRIU**
  - TCP connection
    - Image transfers, process notifications
Intel Pin

Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation

Chia-Kueang Luk, Robert Cohn, Robert Mah, Harish Patil, Artur Klausner, Geoff Lowery
Steve Wallace, Vipin Janaa Reddi, Kim Hazelwood
Intel Corporation, University of Utah
Website: http://regis.colorado.edu/Pin, Email: pin-project@intel.com

Abstract
Robust yet powerful software instrumentation tools are essential for program analysis tools such as profiling, performance evaluation, and bug detection. To meet this need, we have developed Pin, a new instrumentable program analysis environment that allows users to write new, portable, transparent, and efficient instrumenta-
tion. The Pin environment includes a Pin API that captures events such as function calls, data access, cache misses, and errors and allows programs to be instrumented by writing simple programs that are compiled into the target executable. The Pin API is designed to be architecture-independent while providing per-process instrumentation capabilities. Pin also provides a novel technique for dynamic instrumentation that eliminates the need to analyze assembly code. The Pin system is implemented as a dynamic instrumenter that integrates seamlessly into existing software compiles, debuggers, and runtimes. Pin has been used to instrument over 1500 different programs, including commercial software, open-source applications, and benchmarks.

1. Introduction
The goal of Pin is to provide an instrumentation platform for building program analysis tools. Pin allows users to write programs that instrument the target program's execution, using a simple API. Pin's architecture is designed to be architecture-independent while providing per-process instrumentation capabilities. Pin also provides a novel technique for dynamic instrumentation that eliminates the need to analyze assembly code.

Pin is a software system that performs runtime tracing of an application. The core of Pin is a library that provides an instrumentation API and a set of tools for analyzing trace data. Pin is designed to be portable and to work efficiently with applications running on a variety of architectures. Pin is implemented as a dynamic instrumenter that integrates seamlessly into existing software compiles, debuggers, and runtimes. Pin has been used to instrument over 1500 different programs, including commercial software, open-source applications, and benchmarks.

Luk et al., PLDI 2005
Intel Pin

Pin’s features:

- instrument instructions
- interpose on system calls
- hook application symbols

Luk et al., PLDI 2005
libdft: Practical Dynamic Data Flow Tracking for Commodity Systems

Valerios P. Kemerlis, Georgios Poteas, Kanghoo Lee, and Angelos D. Keromytis

Network Security Lab
Division of Computer Science
Brown University, Providence, RI, USA

Abstract

Dynamic data flow tracking (DDFT) lends itself naturally to commodity systems, which are increasingly required to support complex applications with stringent performance guarantees. In this paper, we present libdft, a dynamic DDFT framework that enables precise, lightweight tracking of both remote procedure calls and local function invocations, even in low-trust environments. libdft is designed to be lightweight, having an overhead of less than 10% for both server and client processes, and it supports both local and remote execution. We describe the design of libdft and present an implementation of the framework. We also report on our experience in using libdft in a variety of settings, including a real-world application.

1. Introduction

Dynamic data flow tracking (DDFT) is also referred to as information flow tracking or code instrumentation. It is used to track data flows across different programs to prevent information leakage. DDFT has been used in various applications, such as security, performance monitoring, and debugging. In this paper, we introduce libdft, a dynamic DDFT framework that enables precise, lightweight tracking of both remote procedure calls and local function invocations, even in low-trust environments. Libdft is designed to be lightweight, having an overhead of less than 10% for both server and client processes, and it supports both local and remote execution. We describe the design of libdft and present an implementation of the framework. We also report on our experience in using libdft in a variety of settings, including a real-world application.

Categories and Subject Descriptors D.3.3 [Programming Environments]: Debugging; B.5.2.5 [Networked Information Systems]: Security

General Terms Security, Performance

Keywords Data flow tracking, dynamic binary instrumentation, code analysis, information flow detection, runtime performance

Kemerlis et al., VEE 2012
libdft

libdft’s features:

- 32-bit mode
- core instructions

Our extensions:

- 64-bit mode
- MMX, SSE, SSE2

Caveats:

- control flow taint

if (x)  
  y = 1  
  not tainted (but should be)

- control registers (RFLAGS)

- ptr tainted $\not\Rightarrow$ *ptr tainted

Kemerlis et al., VEE 2012
Taint Tracking

High-level Rules

- Each byte has a shadow taint byte
- `taint ::= 0 | domain_id`
- `domain_id ::= 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128`
- Byte tainted to domain $\implies$ domain exclusively owns byte

Shadow registers

<table>
<thead>
<tr>
<th></th>
<th>RAX</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>R15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MM0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XMM0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Address

- Sign extension
- P4 index
- P3 index
- P2 index
- P1 index
- Offset

Shadow page

- P4 table
- P3 table
- P2 table
- P1 table
Instrumentation

Instruction stream

mov rax, rcx
mov qword ptr [rsi+rax*8], rdx
add rax, qword ptr [r13+0x10]
jmp rdx
xor eax, eax
jmp 0x7fc2e42a72c8
push r14 ; implicit: stack memory, rsp

Cached operand analysis
Access Checks

read written

mov qword ptr[rsi+rax*8], rdx

shadow registers

rax

0
active domain’s id
migrate

rsi

peer domain’s id

otherwise

deadlock

OR

rdx, rdx

OR

OR
# Functional-Level Taint Analysis

## Base64-encoded

```
a2V5
```

## Tainted?

Yes

## Base64-decoded

```
T
['a']
['2']
['V']
['5']
```

```
0x1A
0x36
0x15
0x39
```

<table>
<thead>
<tr>
<th>Character</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'k'</td>
<td>0x1A</td>
</tr>
<tr>
<td>'e'</td>
<td>0x36</td>
</tr>
<tr>
<td>'y'</td>
<td>0x15</td>
</tr>
</tbody>
</table>

## Static Constant

```
static const unsigned char T[128] = {
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
  0xFF, 0xE0, 0xF0, 0xFF, 0xFF, 0xF1, 0xFF, 0xFF,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
  0xE0, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
  0xFF, 0xFF, 0xFF, 0x3E, 0xFF, 0xF2, 0xFF, 0x3F,
  0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B,
  0x3C, 0x3D, 0xFF, 0xFF, 0xFF, 0x00, 0xFF, 0xFF,
  0xFF, 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06,
  0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E,
  0x0F, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16,
  0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E,
  0x1F, 0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26,
  0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E,
  0x2F, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36,
  0x37, 0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E,
  0x3F, 0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46,
  0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D, 0x4E,
  0x4F, 0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56,
  0x57, 0x58, 0x59, 0x5A, 0x5B, 0x5C, 0x5D, 0x5E,
  0x5F, 0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66,
  0x67, 0x68, 0x69, 0x6A, 0x6B, 0x6C, 0x6D, 0x6E,
  0x6F, 0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76,
  0x77, 0x78, 0x79, 0x7A, 0x7B, 0x7C, 0x7D, 0x7E,
  0x7F, 0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86,
  0x87, 0x88, 0x89, 0x8A, 0x8B, 0x8C, 0x8D, 0x8E,
  0x8F, 0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96,
  0x97, 0x98, 0x99, 0x9A, 0x9B, 0x9C, 0x9D, 0x9E,
  0x9F, 0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6,
  0xA7, 0xA8, 0xA9, 0xAA, 0xAB, 0xAC, 0xAD, 0xAE,
  0xAF, 0xB0, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6,
  0xB7, 0xB8, 0xB9, 0xBA, 0xBB, 0xBC, 0xBD, 0xBE,
  0xBF, 0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6,
  0xC7, 0xC8, 0xC9, 0xCA, 0xCB, 0xCC, 0xCD, 0xCE,
  0xCF, 0xD0, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6,
  0xD7, 0xD8, 0xD9, 0xDA, 0xDB, 0xDC, 0xDD, 0xDE,
  0xDF, 0xE0, 0xE1, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6,
  0xE7, 0xE8, 0xE9, 0xEA, 0xEB, 0xEC, 0xED, 0xEE,
  0xEF, 0xF0, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6,
  0xF7, 0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE,
  0xFF
};
```
Functional-Level Taint Analysis

Base64-encoded

a2V5

T[‘a’] T[‘2’] T[‘V’] T[‘5’]

0x1A 0x36 0x15 0x39

Tainted?

Base64-decoded

‘k’ ‘e’ ‘y’

Other cases

- Encryption/Decryption
- Hash functions
- HMACs

static const unsigned char T[128] = {
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
  0xE0, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
  0xFF, 0xFF, 0xFF, 0x3E, 0xFF, 0xF2, 0xFF, 0x3F,
  0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B,
  0x3C, 0x3D, 0xFF, 0xFF, 0xFF, 0x00, 0xFF, 0xFF,
  0xFF, 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06,
  0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E,
  0x0F, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16,
  0x17, 0x18, 0x19, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
  0xFF, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F, 0x20,
  0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28,
  0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F, 0x30,
  0x31, 0x32, 0x33, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
};
Clearing Abandoned Taint

Prologue
- call 0x12345
- push rbp
- mov rbp, rsp
- sub rsp, 16

Stack
- High address
- 0x150
- 0x14f
- ret addr
- prev rbp
- local var0
- local var1
- 0x12f
- 0x120
- 0x11f

Low address

Previous call frame

- rsp
- tracked rsp
Clearing Abandoned Taint

**Prologue**
- call 0x12345
- push rbp
- mov rbp, rsp
- sub rsp, 16

**Epilogue**
- add rsp, 16
- mov rsp, rbp
- pop rbp
- ret

**Stack**

<table>
<thead>
<tr>
<th>Address</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x11f</td>
<td></td>
</tr>
<tr>
<td>0x120</td>
<td></td>
</tr>
<tr>
<td>0x12f</td>
<td></td>
</tr>
<tr>
<td>0x130</td>
<td>local var0</td>
</tr>
<tr>
<td>0x13f</td>
<td>local var1</td>
</tr>
<tr>
<td>0x140</td>
<td>prev rbp</td>
</tr>
<tr>
<td>0x14f</td>
<td>ret addr</td>
</tr>
<tr>
<td>0x150</td>
<td>previous call frame</td>
</tr>
</tbody>
</table>

**Tracking**
- rsp
- tracked rsp
Clearing Abandoned Taint

Stack

High address
0x150
0x14f
0x140
0x13f
0x130
0x12f
0x120
0x11f

Low address

ret addr
prev rbp
local var0
local var1

tracked rsp

Shadow Heap Allocator

addr, size

Preserved (across calls)

Not preserved

Return register

Registers

RAX | R8
RBX | R9
RCX | R10
RDX | R11
RSI | R12
RDI | R13
RBP | R14
RSP | R15

 MM0 | XMM0
 MM1 | XMM1
 MM2 | XMM2
 MM3 | XMM3
 MM4 | XMM4
 MM5 | XMM5
 MM6 | XMM6
 MM7 | XMM7
 MM8 | XMM8
 MM9 | XMM9
 MM10| XMM10
 MM11| XMM11
 MM12| XMM12
 MM13| XMM13
 MM14| XMM14
 MM15| XMM15
System calls

**Spry**

*Process Control Block*
- uid: 500
- gid: 500
- pid: 5140
- umask: 0022
- root: /
- cwd: /srv

**Spin**

*Active domain: 1*

**Application**

- syscall (nanosleep, gettimeofday, etc.)

*UNIX domain socket*

*Pass through to kernel*
Open owned file

Process Control Block
- uid: 500
- gid: 500
- pid: 5140
- umask: 0022
- root: /
- cwd: /srv

Resource policy
<table>
<thead>
<tr>
<th>path</th>
<th>owner</th>
<th>taint-src?</th>
</tr>
</thead>
<tbody>
<tr>
<td>/srv/waf.rules</td>
<td>1</td>
<td>yes</td>
</tr>
<tr>
<td>/srv/key.pem</td>
<td>2</td>
<td>yes</td>
</tr>
</tbody>
</table>

RPC
- open(waf.rules, O_RDONLY)

UNIX domain socket

Application
- Active domain: 1

Spin
- syscall (open)
Virtual File System

**Spry**

**Process Control Block**
- uid: 500
- gid: 500
- pid: 5140
- umask: 0022
- root: /
cwd: /srv

virtual file system

**fd table**
- fd_flags

0   …   4

**file table**
- real_fd: 9
- status_flags

**vnode table**
- owner: 1
taint-src: yes
path: /srv/waf.rules

**Spin**

Active domain: 1

Application

syscall (open)

open(waf.rules, O_RDONLY)

UNIX domain socket

**Resource policy**

path | owner | taint-src?
---|---|---
/srv/waf.rules | 1 | yes
/srv/key.pem | 2 | yes
Read Tainted File

**Spry**

**Process Control Block**
- **uid**: 500
- **gid**: 500
- **pid**: 5140
- **umask**: 0022
- **root**: /
- **cwd**: /srv

---

**fd table**
- **fd_flags**

**file table**
- **real_fd**: 9
- **status_flags**

**vnode table**
- **owner**: 1
- **taint-src**: yes
- **path**: /srv/waf.rules

---

**Spin**

**Active domain**: 1

**Application**

**syscall (read)**

---

**UNIX domain socket**

**read(fd=4, count=512)**

---

**Data, tainted=**yes

**Resource policy**

<table>
<thead>
<tr>
<th>path</th>
<th>owner</th>
<th>taint-src?</th>
</tr>
</thead>
<tbody>
<tr>
<td>/srv/waf.rules</td>
<td>1</td>
<td>yes</td>
</tr>
<tr>
<td>/srv/key.pem</td>
<td>2</td>
<td>yes</td>
</tr>
</tbody>
</table>
Open Peer-Owned File

**Spry**

**Process Control Block**
- uid: 500
- gid: 500
- pid: 5140
- umask: 0022
- root: /
- cwd: /srv

**fd table**
- fd_flags

**file table**
- real_fd: 9
- status_flags

**vnode table**
- owner: 1
- taint-src: yes
- path: /srv/waf.rules

**Resource policy**
- path: /srv/waf.rules
- owner: 1
- taint-src: yes
- path: /srv/key.pem
- owner: 2
- taint-src: yes

**Spin**

**Active domain: 1**

**Application**

**UNIX domain socket**

**syscall (open)**

- open(key.pem, O_RDONLY)

**EMIGRATE, domain=2**

**real registers**

**shadow registers**

- real page
- shadow page
- real page
- shadow page
- real page
- shadow page

**Resource policy**

- path: /srv/waf.rules
- owner: 1
- taint-src: yes
- path: /srv/key.pem
- owner: 2
- taint-src: yes
Spin-Spry Migration Protocol

**Spry**

**Process Control Block**
- uid: 500
- gid: 500
- pid: 5140
- umask: 0022
- root: /
- cwd: /srv
- virtual file system
- stashed, memory

**Spin**

**Active domain: 1**

**Application**

**Application Thread**

- syscall (open)
- EMIGRATE, domain=2

**Migration Queue**

- 2

**Internal thread**

- Semaphore
- sleep

**UNIX domain socket**

- open(key.pem, O_RDONLY)

**Restart**
Spin-Spry Migration Protocol

**Process Control Block**
- `uid`: 500
- `gid`: 500
- `pid`: 5140
- `umask`: 0022
- `root`: /
- `cwd`: /srv

**Virtual File System**
- Stashed memory

**Real Registers**
- Integer: 1
- Integer: 1

**Shadow Registers**
- Integer: 1
- Integer: 1

**Syscall (open)**

**Application Thread**
- **YIELD**

**Migration Queue**

**Internal Thread**
- Semaphore
- `sleep`
Dumping the process

Spry

Process Control Block

Virtual File System

Stashed Memory

CRIU

Spin

Active domain: 1

Application

Internal thread

Semaphore

sleep

real registers

shadow registers

86
Spry-Spry Migration Protocol

Domain A

Spry
Process Control Block
Virtual File System
Stashed Memory

Domain B

Spry
Process Control Block
Virtual File System
Stashed Memory

Transfer images and vfs
Updated
**Restore**

**Spin**

*Active domain: 1*

- Application
  - Application Thread
  - `syscall (open)`
  - Migration Queue
- Internal thread
  - Semaphore
  - `sleep`

**CRIU**

**Spry**

- Process Control Block
- Virtual File System
- Stashed Memory

**88**
Resume

**Spin**

*Active domain: 2*

Application

Application Thread

YIELD

Resume

syscall (open)

Migration Queue

Internal thread

Semaphore

sleep

real registers

shadow registers

real page

shadow page

real page

shadow page

real registers

shadow registers

1 1

1

**Spry**

- Process Control Block
- Virtual File System
- Stashed Memory

**UNIX domain socket**

Swap in
Use Cases

domain A

ClientHello

Net

fs

ServerHello, ...

domain B

private key

signatures

DH param

randoms

tls-key
Use Cases

tls-key

- **ClientHello**
  - Domain A
  - net
  - fs
  - DH param
  - signature

- **ServerHello**
  - Domain B
  - Private key

- **randoms**

**ticket-key**

- **ClientHello**
  - Domain A
  - net
  - ticket

- **Ticket**

- **ServerHello**
  - Domain B
  - ticket key

- **Session key**
Use Cases

**tls-key**
- Domain A
  - Net
  - FS
  - DH param
  - Randoms
  - Signature
- Domain B
  - Private key

**ticket-key**
- Domain A
  - Net
  - FS
- Domain B
  - Ticket
  - Session key

**njs-script**
- Domain A
  - Net
  - FS
- Domain B
  - JS

HTTP Request
- Domain A
- Domain B

HTTP Response
- Domain B

<html>Hello</html>
Use Cases
Overheads

Normalized request latency

Overheads

- Normal latency: 7.3 ms
- Taint tracking
- Access Checks
- Taint in system
- System call RPCs
- Migrations
Overheads

Normalized request latency

<table>
<thead>
<tr>
<th>Component</th>
<th>Normal Latency</th>
<th>Pin Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>tls-key</td>
<td>7.3 ms</td>
<td>1.6</td>
</tr>
<tr>
<td>ticket-key</td>
<td></td>
<td>1.1</td>
</tr>
<tr>
<td>njs-script</td>
<td></td>
<td>1.6</td>
</tr>
<tr>
<td>auth-webdav</td>
<td></td>
<td>1.5</td>
</tr>
</tbody>
</table>

- Taint tracking
- Access Checks
- Taint in system
- System call RPCs
- Migrations

95
Overheads

Normalized request latency

<table>
<thead>
<tr>
<th>Service</th>
<th>Normal latency</th>
<th>Pin</th>
<th>Taint-Tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td>tls-key</td>
<td>7.3 ms</td>
<td>11.7 ms</td>
<td>1.6 ms</td>
</tr>
<tr>
<td>ticket-key</td>
<td>6.2 ms</td>
<td></td>
<td>1.1 ms</td>
</tr>
<tr>
<td>njs-script</td>
<td>530 s</td>
<td></td>
<td>1.6 s</td>
</tr>
<tr>
<td>auth-webdav</td>
<td>509 s</td>
<td></td>
<td>1.5 s</td>
</tr>
</tbody>
</table>

- Taint tracking
- Access Checks
- Taint in system
- System call RPCs
- Migrations
Overheads

Normalized request latency

<table>
<thead>
<tr>
<th>Component</th>
<th>Normal Latency</th>
<th>Pin</th>
<th>Taint-Tracking</th>
<th>Access-Checks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tls-key</td>
<td>7.3 ms</td>
<td>11.7 ms</td>
<td>3.9 s</td>
<td>4.4 s</td>
</tr>
<tr>
<td>Ticket-key</td>
<td></td>
<td>1.6</td>
<td>6.2</td>
<td>7</td>
</tr>
<tr>
<td>Njs-script</td>
<td></td>
<td>1.6</td>
<td>539</td>
<td>591</td>
</tr>
<tr>
<td>Auth-webdav</td>
<td></td>
<td>1.5</td>
<td>509</td>
<td>568</td>
</tr>
</tbody>
</table>

Taint tracking  ✔️
Access Checks   ✔️
Taint in system
System call RPCs
Migrations
**Overheads**

Normalized request latency

- **Pin**
- **Taint-Tracking**
- **Access-Checks**

<table>
<thead>
<tr>
<th>Service</th>
<th>Normal latency</th>
<th>Pin latency</th>
<th>Taint Tracking</th>
<th>Access Checks</th>
<th>Taint in system</th>
<th>System call RPCs</th>
<th>Migrations</th>
</tr>
</thead>
<tbody>
<tr>
<td>tls-key</td>
<td>7.3 ms</td>
<td>11.7 ms</td>
<td>3.9 s</td>
<td>4.4 s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ticket-key</td>
<td>1.6</td>
<td>1.1</td>
<td>7.7</td>
<td>6.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>njs-script</td>
<td>1.6</td>
<td>5.3</td>
<td>5.9</td>
<td>5.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>auth-webdav</td>
<td>1.5</td>
<td>5.9</td>
<td>6.1</td>
<td>6.2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Normal latency**

- **Pin**: 7.3 ms
- **Taint-Tracking**: 3.9 s
- **Access Checks**: 4.4 s
- **1-domain**: 4.5 s
Overheads

Normalized request latency

<table>
<thead>
<tr>
<th>Service</th>
<th>1-domain</th>
<th>2-domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taint-tracking</td>
<td>530</td>
<td>591</td>
</tr>
<tr>
<td>Access-checks</td>
<td>562</td>
<td>602</td>
</tr>
<tr>
<td>Auth-webdav</td>
<td>505</td>
<td>568</td>
</tr>
</tbody>
</table>

**Normalized latency**

- **Taint tracking**: 3.9 s
- **Access Checks**: 4.4 s
- **Taint in system**: 4.5 s
- **System call RPCs**: 5.1 s
- **Migrations**: 8.6 s

**Pin**

- Normal latency: 7.3 ms
- Access checks: 11.7 ms
Migration Patterns: tls-key

Load Key

domain B

private key

domain A

net

fs

_close(confdd)

ngx_ssl_load_certificate_key → openat(key.pem)

Time (sec)

\( t_A \) 21.0%

\( t_B \) 43.8%

\( t_{\text{mig}} \) 35.2%

\( \text{BIO\_write} \rightarrow \text{write(sockfd)} \)
Migration Patterns: waf/tls-key

Domain B
- Private key

Domain A
- Filesystem
- WAF rules

Time (sec)
0 10 20 30 40 50 60 70 80

Load Key
- openat(key.pem)
- fstat(keyfd)
- close(keyfd)
- SSL_CTX_use_PrivateKey

Tainted stack

GET0
GET1
GET2
GET3
Migration Patterns: ticket-key

**ngx_ssl_session_ticket_keys → open(ticket.key)**

**ngx_ssl_session_ticket_ticket_callback → AES_set_encrypt_key**

**ngx_ssl_session_ticket_ticket_callback → memcmp**

---

**domain B**

**ticket key**

**domain A**

**net**

**fs**

---

**Time (sec)**

0 5 10 15 20 25 30 35 40 45 50

**load Key**

**close(conf_fd)**

**BIO_write → write(sockfd)**

---

\[ t_A = 54.7\% \]

\[ t_B = 2.4\% \]

\[ t_{mig} = 42.9\% \]
Migration Patterns: njs-script

- njs_module_relative_path → openat(http.js)
- njs_module realpath_equal → lstat(http.js)
- njs_vmcode_interpreter
- Load script
- Form module path
- tainted stack
- GET0
- GET1
- GET2
- GET3
- close(conf_fd)
- BIO_write → write(sockfd)
- Time (sec)
  - $t_A$: 49.4%
  - $t_B$: 9.4%
  - $t_{mig}$: 41.2%

Domain A
- net
- fs

Domain B
- JS
Migration Patterns: waf/njs-script
Taint growth over time

**tls-key**

- Bytes tainted: 0
- Time (sec): 0 to 250

**ticket-key**

- Bytes tainted: 0
- Time (sec): 0 to 250

**njs-script**

- Bytes tainted: 0
- Time (sec): 0 to 250

**auth-webdav**

- Bytes tainted: 0
- Time (sec): 0 to 250
Taint growth over time

- **tls-key**
  - Initial Taint
  - Size

- **ticket-key**
  - Bytes tainted

- **njs-script**
  - Bytes tainted

- **auth-webdav**
  - Bytes tainted

**Time (sec)**
Taint growth over time

- tls-key
- ticket-key
- njs-script
- auth-webdav

Bytes tainted vs Time (sec)
Taint growth over time

Bytes tainted

Time (sec)

Pages tainted
Adapting Old Software to New Trust Models

Conclaves
- trusted CPU
- strong threat models
- broad use cases
- reasonable overhead (but still high)

SecureMigration
- conventional hardware
- honest-but-curious
- restricted use cases
- high overhead
It is possible to run legacy application binaries with confidentiality and integrity guarantees that reflect a multi-party trust setting.