Glamdring

Automatic Application Partitioning for Intel SGX

Joshua Lind, Christian Priebe, Divya Muthukumaran, Dan O’Keeffe, Pierre-Louis Aublin, Florian Kelbert
Imperial College London

Tobias Reiher
TU Dresden

David Goltzsche
TU Braunschweig

David Eyers
University of Otago

Rüdiger Kapitza
TU Braunschweig

Christof Fetzer
TU Dresden

Peter Pietzuch
Imperial College London

dmuthuku@imperial.ac.uk
Trust in Cloud Services

Application

OS
VMM
Firmware
Cloud platform
Trust in Cloud Services

Threats

- Insider Attacks
- Human error despite best practices
- Vulnerabilities in large code bases
Trust in Cloud Services

Traditional Security Models
- Protect privileged code from untrusted user-level code
Trusted Execution Environments

- Secure area of a processor
- Provides protection from higher privileged code
- Trusted environment on top of untrusted cloud
Intel Software Guard Extensions (SGX)

- On commodity processors starting with Skylake
- TEE’s are called enclaves
- 18 CPU instructions to manage enclave lifecycle
- Code & data reside in Enclave Page Cache (EPC)
  - Cache lines encrypted when written to memory
  - Restricted to 128MB
- Intel provides an SDK for Windows and Linux
Enclave Application Lifecycle

1. Start Enclave
2. Ecall

3. Trusted function
4. Ocall
5. Return

Higher Privileged Code (OS, VMM)
Enclave Application Lifecycle

1. Start Enclave
2. Ecall

3. Trusted function
4. Ocall
5. Return

Higher Privileged Code (OS, VMM)
Enclave Application Lifecycle

Enclave crossings through ecalls and ocalls incur a performance penalty
Porting applications to Enclaves

How do you port a key-value store to run in an enclave?
Library OS Inside Enclaves

Pros
• Run unmodified applications
• Fixed shielded interface

Cons
• TCB is millions LoC!
• Performance overhead

Haven [OSDI’14]
Standard Library Inside Enclaves

Pros
- Smaller TCB than Haven
- Fixed shielded interface

Cons
- TCB = 0.6x–2x of application size
- Recompilation needed

SCONE [OSDI’16]
Minimum TCB Inside Enclaves

Principle of Least Privilege
Only move the code needed to enforce security policy

Policy: Confidentiality and Integrity of key-value pairs
Minimum TCB Inside Enclaves

Principle of Least Privilege
Only move the code needed to enforce security policy

Policy: Confidentiality and Integrity of key-value pairs
Application Partitioning to Minimise TCB

Prior work has manually partitioned applications

SecureKeeper: Confidential ZooKeeper using Intel SGX

Stefan Brenner
TU Braunschweig, Germany
brenner@lbr.cs.tu-bs.de

Colin Wulf
TU Braunschweig, Germany
cwulf@lbr.cs.tu-bs.de

David Goltzsche
TU Braunschweig, Germany
goltzsche@lbr.cs.tu-bs.de

Nico Weichbrodt
TU Braunschweig, Germany
weichbr@lbr.cs.tu-bs.de

Matthias Lorenz
TU Braunschweig, Germany
mlorenz@lbr.cs.tu-bs.de

Christof Fetzer
TU Dresden, Germany
christof.fetzer@tu-dresden.de

Peter Pietzuch
Imperial College London, UK
prp@imperial.ac.uk

Rüdiger Kapitza
TU Braunschweig, Germany

ABSTRACT
Cloud computing, while ubiquitous, still suffers from trust issues, especially for applications managing sensitive data. Third-party cloud services often are untrusted, and the cloud itself may be malicious. SecureKeeper mitigates these threats by providing secure and transparent access to ZooKeeper.

VC3: Trustworthy Data Analytics in the Cloud using SGX

Felix Schuster*, Manuel Costa, Cédric Fourme, Christos Gkantsidis
Marcus Peinado, Gloria Mainar-Ruiz, Mark Russinovich
Microsoft Research

Abstract—We present VC3, the first system that allows users to run distributed MapReduce computations in the cloud while keeping their code and data secret, and ensuring the correctness and completeness of their results. VC3 runs on unmodified Hadoop, but crucially keeps Hadoop, the operating system and the hypervisor out of the TCB, thus confidentiality and integrity data [22]. However, FHE is not efficient for most computations [23], [65]. The computation can also be shared between independent parties while guaranteeing confidentiality for individual inputs (e.g., garbled circuits [29]) and providing protection against corrupted parties (see e.g.,
Application Partitioning to Minimise TCB

Prior work has manually partitioned applications.

“Automatically determine the minimum functionality to be run inside an enclave in order to enforce a security policy.”
Challenges in Automated Partitioning

- Identifying security-sensitive code relevant to a security policy
- Preventing interfaces from violating security policy
- Avoiding performance degradation

**Policy**: Confidentiality and Integrity of key-value pairs
Challenges in Automated Partitioning

- Identifying security-sensitive code relevant to a security policy
- Preventing interfaces from violating security policy
- Avoiding performance degradation

Policy: Confidentiality and Integrity of key-value pairs
Challenges in Automated Partitioning

- Identifying security-sensitive code relevant to a security policy
- Preventing interfaces from violating security policy
- Avoiding performance degradation

**Policy**: Confidentiality and Integrity of key-value pairs
Challenges in Automated Partitioning

- Identifying security-sensitive code relevant to a security policy
- Preventing interfaces from violating security policy
- Avoiding performance degradation

**Policy**: Confidentiality and Integrity of key-value pairs
Glamdring Partitioning Framework

Annotation | Application Code

Static Analysis
- Forward Analysis
- Backward Analysis

Enclave Boundary Relocation
- Partition specification

Source-Source Transformation
- Instrumentation of Runtime Invariants

Enclave Code
- Outside Code
- Interface Spec

Invariants

1. Static Analysis
2. Enclave Boundary Relocation
3. Source-Source Transformation
4. Invariants
1. Identify Security-Sensitive Code

Static Analysis conservatively identifies subset of code dependent on programmer annotated security-sensitive data.
Annotation of Security-Sensitive Data

What to Annotate
- Indicate where security-sensitive data enters or leaves the program
- Security-sensitive data can be encrypted and signed until first use
Annotation of Security-Sensitive Data

What to Annotate
- Indicate where security-sensitive data enters or leaves the program
- Sensitive data can be encrypted and signed until first use

Diagram:
- Client
- Dispatch(cmd)
  - read()
  - If (cmd == "GET")
    - Get()
    - Update()
Annotation of Security-Sensitive Data

What to Annotate

- Indicate where security-sensitive data enters or leaves the program
- Sensitive data can be **encrypted** and **signed** until first use
Annotation of Security-Sensitive Data

```c
#pragma glamdring sensitive source(cmd)
void Dispatch(char *cmd) {
    ...
}
```
Static Analysis Goals

• Enforcing **Confidentiality**: Identify all functions that depend on sensitive data.

• Enforcing **Integrity**: Identify all functions on which the value of sensitive data depends

**Why Static Analysis?**

• Static Analysis is **conservative**, independent of the input to the program
Program Dependence Graph

Captures the control and data dependencies in the program
Program Dependence Graph

Captures the **control** and **data** dependencies in the program

Nodes = Statements

- \( \text{cmd} = \text{read}(..) \) (S1)
- \( \text{Dispatch}(\text{cmd}) \) (S2)
- \( \text{If} (\text{cmd} == \text{"GET"}) \) (S3)
- \( \text{Get()} \) (S5)
- \( \text{Update()} \) (S4)
Program Dependence Graph

Captures the **control** and **data** dependencies in the program

**Data Dependence Edge**
Data defined in a statement is used in the another statement

```
cmd = read(..)       S1

Dispatch(cmd)       S2

If (cmd == "GET")   S3

S5 Get()           Update() S4
```
Program Dependence Graph

Captures the **control** and **data** dependencies in the program

Control Dependence Edge
One Statement determines if another gets executed

1. cmd = read(..) (S1)
2. Dispatch(cmd) (S2)
3. If (cmd == “GET”) (S3)
4. Get() (S5)
5. Update() (S4)
Program Dependence Graph

```
Dispatch(cmd)

Get()
Update()

If (cmd == “GET”)

Rest of the program

... cmd = read(..) ...

Format()

Write(res)
```
Forwards Dataflow Analysis

**Confidentiality** Using Graph Reachability identify all nodes with transitive control/data dependency on annotated node

```
Dispatch(cmd)
Get()
Update()
If (cmd == "GET")
  Write(res)
```

Rest of the program
Forwards Dataflow Analysis

**Confidentiality** Using Graph Reachability identify all nodes with transitive control/data dependency on annotated node

```
Dispatch(cmd)
Get()
Update()
If (cmd == "GET")
  Write(res)
Rest of the program
```

#pragma glamdring sensitive data(cmd)
Forwards Dataflow Analysis

**Integrity** Using Graph Reachability identify all nodes that are transitive control/data dependent on annotated node

Rest of the program

```
 Dispatch(cmd)
 Get()
 Update()
 If (cmd == "GET")
 Write(res)
 cmd = read(..)
 #pragma glamdring sensitive data(cmd)
```
Forwards Dataflow Analysis

**Integrity** Using Graph Reachability identify all nodes that are transitive control/data dependent on annotated node

```
Dispatch(cmd)
Get()
Update()
If (cmd == "GET")
Write(res)
```

Rest of the program
Security Sensitive Code

Union of nodes found with forwards and backwards analyses

Rest of the program

... Format() Write(res) Dispatch(cmd) If (cmd == ”GET”) Get() Update() cmd = read(..) ...
Produce Partition Specification

Partition Specification

* Enclave Functions:
  Dispatch
  Get
  Update

* Enclave Allocations:
  malloc@241

* Enclave Allocated Globals
  hash_items
2. Producing a Partitioned Application

Automatically move code into enclave and outside codebases; Generate interface specification for SDK

Annotation → Application Code

Static Analysis

Forward Analysis
Backward Analysis

Partition specification

Source-Source Transformation

Enclave Code
Outside Code
Interface Spec
Source-Source Transformation

Partition Spec
* **Enclave Functions:**
  - Dispatch,
  - Get,
  - Update
* **Enclave Allocations:**
  - malloc@241
* **Enclave Allocated Globals**
  - hash_items

```c
void Read(...) {
    Dispatch();
}

void Dispatch(...){
    ...
}

void Get(...) {
    ...
}

void Put(...) {
    ...
}
```
Source-Source Transformation

Partition Spec

* **Enclave Functions:**
  - Dispatch,
  - Get,
  - Update

* **Enclave Allocations:**
  - malloc@241

* **Enclave Allocated Globals**
  - hash_items

```c
void Read(...) {
    Dispatch();
}

void Dispatch(...) {
    ...
}

void Get(...) {
    ...
}

void Put(...) {
    ...
}
```
Source-Source Transformation

**Partition Spec**
- **Enclave Functions:** Dispatch, Get, Update
- **Enclave Allocations:** malloc@241
- **Enclave Allocated Globals:** hash_items

**Outside**

```c
void Read(...) {
  ecall__Dispatch();
}
```

**Enclave**

```c
void ecall__Dispatch(...){
  ...
}
void Get(...) {
  ...
}
void Put(...) {
  ...
}
```
3. Upholding Static Analysis Invariants

Ensure that invariants on program state used by the static analysis are enforced at runtime.
Infeasible Program Paths

Problem
Static Analysis prunes infeasible paths by inferring invariants on program state

```c
int flag = 0;

int SomeFunc() {
    if (flag == 1)
        memcpy(data, sensitive_data);
    else
        memcpy(data, declassify(sensitive_data));
    Write(data);
}
```
Infeasible Program Paths

Problem
Static Analysis prunes infeasible paths by inferring invariants on program state

```c
int flag = 0; /* flag == 0 */

int SomeFunc() {
    if(flag == 1)
        memcpy(data, sensitive_data);
    else
        memcpy(data, declassify(sensitive_data));
    Write(data);
}
```
Violating Static Analysis Invariants

Problem
Attacker controlling untrusted code can violate the assumptions made by static analysis after partitioning

```c
int flag = 0;

int SomeFunc() {
    if(flag == 1)
        memcpy(data, sensitive_data);
    else
        memcpy(data, declassify(sensitive_data));
    Write(data);
}
```
Adding Runtime Invariant Checks

Solution
Add assertions to enforce statically inferred invariants on program state

```c
int flag = 0;

int SomeFunc() {
    assert(flag == 0);
    if(flag == 1)
        memcpy(data, sensitive_data);
    else
        memcpy(data, declassify(sensitive_data));
    Write(data);
}
```
4. Improving Performance After Partitioning

Use results of runtime profiling to remove expensive functions from enclave interface.
Performance of Partitioned Applications

Expensive Interface Functions
Some of the interface functions may be ‘hotspots’ called too frequently

- Dispatch(cmd)
  - If (cmd == “GET”)
    - Get()
    - Update()
Performance of Partitioned Applications

Expensive Interface Functions
Some of the interface functions may be ‘hotspots’ called too frequently

Runtime profiling can help identify hotspots
Enclave Boundary Relocation

Adding Functions to Enclave
Move additional functions into enclave to create a new interface that avoid ‘hotspots’
Evaluation Goals

- How does Glamdring compare to other design choices
  - Security: Size of TCB
  - Performance: Throughput
Applications and Implementation

<table>
<thead>
<tr>
<th>Application</th>
<th>Data</th>
<th>Confidentiality</th>
<th>Integrity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memcached</td>
<td>Key-Value pairs</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LibreSSL</td>
<td>CA Root certificate</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Digital Bitbox</td>
<td>Private Keys</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Implementation

- **Static Analysis:**
  - Existing tools

- **Code Generation:**
  - LLVM/Clang 3.9 — around 5000 LoC
## Security Evaluation - TCB size

How big is the TCB of applications?

<table>
<thead>
<tr>
<th>Applications</th>
<th>Code Size (kLoC)</th>
<th>TCB size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memcached</td>
<td>31</td>
<td>12 (40%)</td>
</tr>
<tr>
<td>DigitalBitbox</td>
<td>23</td>
<td>8 (38%)</td>
</tr>
<tr>
<td>LibreSSL</td>
<td>176</td>
<td>38 (22%)</td>
</tr>
</tbody>
</table>

TCB is less than 40% of the application size.
## Security Evaluation - TCB size

TCB size comparison with Graphene and SCONE

<table>
<thead>
<tr>
<th>Applications</th>
<th>TCB size (kLoC)</th>
<th>Binary Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memcached (Glamdring)</td>
<td>42</td>
<td>770 kB</td>
</tr>
<tr>
<td>Memcached (SCONE)</td>
<td>149</td>
<td>3.3 MB</td>
</tr>
<tr>
<td>Memcached (Graphene)</td>
<td>746</td>
<td>4.1 MB</td>
</tr>
</tbody>
</table>
## Security Evaluation - TCB size

TCB size comparison with Graphene and SCONE

<table>
<thead>
<tr>
<th>Applications</th>
<th>TCB size (kLoC)</th>
<th>Binary Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memcached (Glamdring)</td>
<td>42</td>
<td>770 kB</td>
</tr>
<tr>
<td>Memcached (SCONE)</td>
<td>149</td>
<td>3.3 MB</td>
</tr>
<tr>
<td>Memcached (Graphene)</td>
<td>746</td>
<td>4.1 MB</td>
</tr>
</tbody>
</table>

1/3 size of TCB when using SCONE
## Security Evaluation - TCB size

TCB size comparison with Graphene and SCONE

<table>
<thead>
<tr>
<th>Applications</th>
<th>TCB size (kLoC)</th>
<th>Binary Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memcached (Glamdring)</td>
<td>42</td>
<td>770 kB</td>
</tr>
<tr>
<td>Memcached (SCONE)</td>
<td>149</td>
<td>3.3 MB</td>
</tr>
<tr>
<td>Memcached (Graphene)</td>
<td>746</td>
<td>4.1 MB</td>
</tr>
</tbody>
</table>

1/3 size of TCB when using SCONE

Order of magnitude less than with Graphene
Comparing Performance of Design Approaches

Throughput of Memcached ported using Glamdring with native, SCONE and Graphene
Comparing Performance of Design Approaches

Throughput of Memcached ported using Glamdring with native, SCONE and Graphene

Native
Throughput vs Latency

Avoids enclave transitions with user-level threading; higher TCB than Glamdring

Native

SCONE
Throughput vs Latency

Entire Library OS inside enclave

Throughput vs Latency graph showing Throughput on the x-axis and Latency on the y-axis. The graph compares three different systems: Native, SCONE, and Graphene. The Native system has the highest latency at higher throughputs. SCONE and Graphene show a performance trade-off, with Graphene achieving lower latency at lower throughputs compared to SCONE.
Throughput vs Latency

Latency

Throughput

Native  SCONE  Graphene  Glamdring
Throughput vs Latency

Enclave transitions dominate the cost of request handling; batching requests into multi-get gets 210k req/sec
Conclusions

• Port applications into Intel SGX enclaves with minimal TCB

• **Glamdring** — Automated program partitioning using static analysis
  - Identifies minimum TCB, produces partitioned code, enforces program state invariants, uses

• Evaluated three applications - smaller TCB than prior approaches with acceptable performance

**Divya Muthukumaran**
dmuthuku@imperial.ac.uk
Security Evaluation - Attacks and Defences

- **Enclave Call Ordering Attacks**: By construction. EBR does not affect this.

- **Iago Attacks**: By enforcing invariants

- **Replay Attacks**: Freshness counter

- **Enclave Code Vulnerabilities**: TCB is reduced — enables code analysis
Evaluation - Impact of EBR

How many functions were moved into the enclave, and what was the impact on enclave crossings

<table>
<thead>
<tr>
<th>Application</th>
<th>EBR Enclave Functions</th>
<th>Enclave Crossings (No EBR)</th>
<th>Enclave Crossings (With EBR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memcached</td>
<td>1</td>
<td>54</td>
<td>6</td>
</tr>
<tr>
<td>LibreSSL</td>
<td>2</td>
<td>24,780</td>
<td>6727</td>
</tr>
<tr>
<td>Digital Bitbox</td>
<td>4</td>
<td>10,943</td>
<td>38</td>
</tr>
</tbody>
</table>
Evaluation - Impact of EBR

Even **few functions** inside…. reduced enclave crossings by orders of magnitude

<table>
<thead>
<tr>
<th>Application</th>
<th>EBR Enclave Functions</th>
<th>Enclave Crossings (No EBR)</th>
<th>Enclave Crossings (With EBR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memcached</td>
<td>1</td>
<td>54</td>
<td>6</td>
</tr>
<tr>
<td>LibreSSL</td>
<td>2</td>
<td>24,780</td>
<td>6727</td>
</tr>
<tr>
<td>Digital Bitbox</td>
<td>4</td>
<td>10,943</td>
<td>38</td>
</tr>
</tbody>
</table>