Partitioning System Software for Hardware Enclaves

Chia-Che Tsai
Texas A&M University / Graphene Project / Anjuna
Why Partition System Software?

Target of enclave protection

- Application
- Language Runtime
- Libraries
- Operating System

Data Flow (in both data & control planes)

1. Where is the partition boundary?
2. How to declassify data?

IO Devices (less trustworthy)
Attack Surface vs TCB?

Large attack surface
Small TCB

System API
Redirection Layer

Untrusted Host OS
Shim Layer (SCONE)

Small attack surface
Large TCB

System API

Guest OS
Hypervisor interface

Untrusted Host OS
Library OS / Unikernel
(Graphene, SGX-LKL)
Danger in A Partition Interface

**lago attacks** [ASPLOS 2013]

Untrusted OS exploits semantic vulnerabilities misplaced in legacy applications

Ex: Assuming PID and time are reliable source of entropy

Pervasive threats in libraries, runtimes, even hypervisor.

Integral to design of systems
**Legacy System API (e.g., Linux System Calls)**

<table>
<thead>
<tr>
<th>Secure Host Interface</th>
<th>Potential DoS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Secure Boot</strong></td>
<td>Secure Scheduling</td>
</tr>
<tr>
<td>Start proc</td>
<td>Yield</td>
</tr>
<tr>
<td>Start thread</td>
<td>Poll</td>
</tr>
<tr>
<td><strong>Secure Clock</strong></td>
<td>Signal/Wait</td>
</tr>
<tr>
<td>Monotonic time</td>
<td></td>
</tr>
<tr>
<td>Date / clock</td>
<td></td>
</tr>
<tr>
<td><strong>Secure VM</strong></td>
<td></td>
</tr>
<tr>
<td>Stack</td>
<td></td>
</tr>
<tr>
<td>Heap</td>
<td></td>
</tr>
<tr>
<td><strong>Secure IO</strong></td>
<td></td>
</tr>
<tr>
<td>Files</td>
<td></td>
</tr>
<tr>
<td>Sockets</td>
<td></td>
</tr>
<tr>
<td><strong>Secure RPC</strong></td>
<td></td>
</tr>
<tr>
<td>FIFOs</td>
<td></td>
</tr>
</tbody>
</table>

**Partition for A Secure Interface**
Partition for A Secure Interface

Legacy System API (e.g., Linux System Calls)

Secure Host Interface

Secure Boot
- Start proc
- Start thread

Secure Clock
- Monotonic time
- Date / clock

Secure VM
- Stack
- Heap

Secure IO
- Files
- Sockets

Secure RPC
- FIFOs

Potential DoS
- Secure Scheduling
  - Yield
  - Poll
  - Signal/wait

fork
Partition for A Secure Interface

Legacy System API (e.g., Linux System Calls)

fork

IPC
(Message Queue, Semaphore, Signals, File locks)

Secure Boot
Start proc
Start thread

Secure Clock
Monotonic time
Date / clock

Secure VM
Stack
Heap

Secure IO
Files
Sockets

Secure RPC
FIFOs

Secure Scheduling
Yield
Poll
Signal/wait

Potential DoS
Legacy System API (e.g., Linux System Calls)

- fork
  - (Message Queue, Semaphore, Signals, File locks)
- Async IO
- Secure Host Interface
  - Secure Boot: Start proc, Start thread
  - Secure Clock: Monotonic time, Date / clock
  - Secure VM: Stack, Heap
  - Secure IO: Files, Sockets
  - Secure RPC: FIFOs
  - Secure Scheduling: Yield, Poll, Signal/wait

Potential DoS
The Graphene Architecture

Graphene LibOS

Graphene Host ABI (40 Calls)

With portable & secure semantics

140 / 318 system calls
(core features)

63 KLOC
Source code

1.4 MB
Library size

SGX Port + Shield

Container Port

Non-Linux Platform Ports

Not just for enclaves
Ideally you want to isolate out a minimum partition.

Partitioning across system stack and components is difficult.

Example: Hadoop

- Java App: 6.3 MLoC
- Language Runtime: 2.3 MLoC
- Libraries: 0.9 MLoC
Civet: Partitioned Java Software Stack

Joint work with Raluca Ada Popa, Jeongseok Son (Berkeley), Don Porter, Bhushan Jan (UNC)

Hadoop library
(6.3 MLoC)

Mapper
Reducer

Partition Tool

Trusted JAR
(Contains only needed classes)

Enclave

Mapper
Reducer

Defense

Untrusted JAR
(Synthesized RPC interfaces)

Job

Partitioned JVM

RPC

RPC

Interface classes
System partitioning is a critical challenge:

**OS-level:** Graphene library OS
   Emulating legacy system API on minimal secure abstractions

**Runtime-level:** Civet framework for Java
   Static cross-stack partitioning + language defense & optimization

https://grapheneproject.io
support@graphene-project.io