ARMlock: Hardware-based Fault Isolation for ARM

Yajin Zhou, Xiaoguang Wang, Yue Chen, and Zhi Wang

North Carolina State University
Xi’an Jiaotong University
Florida State University
Software is Complicated and Vulnerable

- 17 million SLOC
- 15 million SLOC
- 2 million SLOC
- 400 thousands SLOC

Number of CVEs:

- Linux
- Chrome
- Apache
- Libpng

Software is Complicated and Vulnerable

- Code: different sources
  - Third-party libraries, plugins …
- Vulnerabilities in one module could compromise the whole application
Software Fault Isolation

- SFI: security by isolation
  - Split application into different fault domains
  - Separate each domain from others
  - Compromised fault domains cannot affect others

- Widely used in x86 systems
  - Linux kernel: LXFI
  - User level applications: Native client, Vx32 …

Our work focuses on ARM architecture
ARM Architecture is Popular

750 million Android devices in 2013
99% are based on ARM architecture

ARM is catching up in the data center server market
SFI on ARM Architecture

- Native client for ARM
  - Compiler based solution
  - Limitations: assumption on memory layout, hard to efficiently support self-modifying code, and JIT compiling

- ARMor
  - Binary rewriting
  - High performance overhead
Our Solution: ARMlock

- **Strict isolation**
  - Memory read/write, code execution, system calls
- **Low performance overhead**
  - Sandbox context switch, sandbox itself
- **Compatibility**
  - Memory layout, self-modifying code, JIT compiling
- **Leverage an often overlooked hardware feature:** Memory domain
Background: ARM Memory Domain

Virtual memory space

Sandboxed code

lw r0, [r1]
...
...
lw r1, [r2]

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Access</td>
<td>00</td>
<td>No access permitted</td>
</tr>
<tr>
<td>Client</td>
<td>01</td>
<td>Permissions defined by page tables</td>
</tr>
<tr>
<td>Reserved</td>
<td>10</td>
<td>Reserved</td>
</tr>
<tr>
<td>Manager</td>
<td>11</td>
<td>No permissions check (unlimited access)</td>
</tr>
</tbody>
</table>

DACR Register

ARM domain access control

[Diagram showing ARM Memory Domain with lw instructions and DACR register values]
Threat Model

- OS kernel is trusted
- Host application is benign but could be vulnerable
- External modules: vulnerable or malicious

Isolate compromised or malicious modules from the host application
ARMlock Architecture

- **Sandboxed untrusted module**
  - Data
  - Function

- **Host application**
  - Data
  - Function

- **User mode**
  - System call interposition

- **Kernel mode**
  - ARMlock kernel extension

- **Cross-sandbox communication** (with the help of ARMlock kernel extension)
Sandbox Creation

- Host application asks ARMlock kernel module to create a sandbox
- Kernel module initializes the sandbox
  - Locate first level page table entries
  - Assign different memory domains to the host application and sandboxes
- Memory domain assignment cannot be changed by the sandbox
Sandbox Switch

- DACR register is saved in the thread control block
- DACR register is updated when switching sandboxes
  - Only current domain (and kernel) are accessible, not other domains
- Multithreading is naturally supported
  - Each CPU core has its own DACR register
Cross-sandbox Communication

- Inter-module function call
- Inter-module memory reference
Inter-module Function Invocation

- Two new system calls
  - ARMlock_CALL: inter-module function call
  - ARMlock RET: inter-module function return
Inter-module Function Invocation

- Inter-module function invocation

Host application

- caller
- stub
- gate

Sandboxed untrusted module

- gate
- callee

User mode

Linux kernel

ARMlock kernel extension

- Inter-domain transfer (with the help of ARMlock kernel extension)
- Intra-domain transfer (with the help of ARMlock user library)
Inter-module Function Invocation

1. **Caller**: Call *Sandbox1_func*

2. **Stub**
   - Prepare context
   - Issue **ARMlock_CALL**

3. **Entry gate**
   - Prepare context
   - Call real function
   - Issue **ARMlock_RET**

4. **Callee**: *Sandbox1_func*

5. **Return gate**
   - Switch stack etc.
   - Set PC to entry gate
   - **ARMlock_CALL** returns

6. **Sandbox 0**
   - Restore context
   - Return

7. **ARMlock kernel module**
   - Switch stack etc.
   - Set PC to return gate
   - **ARMlock_RET** returns

8. **Sandbox 1**
   - Return
Inter-module Memory Reference

- Kernel assisted memory copy
  - Kernel marks both domains as accessible
  - Copy data into the destination sandbox
  - Restore the DACR register
- Shared memory domain: using a domain which is accessible in both sandboxes
- Data from sandboxed modules should be sanitized
System Call Interposition

- Recent Linux system has 380+ system calls
  - Normal applications may use less than that, e.g., around 60
  - More system calls may expose more kernel vulnerabilities
- Host applications in ARMlock could control system calls available to sandboxed modules
- Implemented through the seccomp-BPF framework
Evaluation

- Security analysis
- Performance overhead
  - Sandbox switch latency
  - Sandbox itself
Security Analysis

- Cross-sandbox communication
  - Inter-module function invocation
  - Inter-module memory reference
    - Kernel assisted memory copy
    - Shared memory domain: race condition
## Performance Evaluation: Configuration

<table>
<thead>
<tr>
<th>Item</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>ARM1176JZF-S 700MHz</td>
</tr>
<tr>
<td>RAM</td>
<td>512MB</td>
</tr>
<tr>
<td>OS</td>
<td>Raspbian (based on Debian)</td>
</tr>
<tr>
<td>Kernel</td>
<td>Linux 3.6.11</td>
</tr>
<tr>
<td>LMbench</td>
<td>Version 2</td>
</tr>
<tr>
<td>nbench</td>
<td>Version 2.2.3</td>
</tr>
</tbody>
</table>
Call a simple *inc* function inside the sandbox

- 1 second: 903,343 inter-module calls -- 1.1 µs for each call
## Sandbox Switch Latency

- **One sandbox switch: two system calls**

<table>
<thead>
<tr>
<th>Command</th>
<th>Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARMlock</td>
<td>2.6</td>
</tr>
<tr>
<td>clock</td>
<td>5.8</td>
</tr>
<tr>
<td>exec</td>
<td>2778.3</td>
</tr>
<tr>
<td>fork</td>
<td>2631</td>
</tr>
<tr>
<td>getpid</td>
<td>1</td>
</tr>
<tr>
<td>null</td>
<td>1</td>
</tr>
<tr>
<td>sig_handle</td>
<td>16.4</td>
</tr>
<tr>
<td>sig_install</td>
<td>3.1</td>
</tr>
<tr>
<td>stat</td>
<td>17.4</td>
</tr>
</tbody>
</table>
Performance Overhead

- Assignment
- Bitfield
- FP_emulation
- Fourier
- Huffman
- IDEA
- LU_decomposition
- Neural_net
- Numeric_sort
- String_sort

0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%

Internal External

Values:
- Fourier: 0.584
- Neural_net: 0.661
Some developer efforts are required
  - Refactor the application into domains
  - Avoid frequent domain switch
Need to use short format page table in latest ARM architecture
Kernel-level sandbox
Other OS support
Takeaway

- ARMlock: a hardware-based fault isolation for ARM
  - Strict isolation
  - Low performance overhead
  - Better compatibility
Q&A