HardBlare: a Hardware-Assisted Approach for Dynamic Information Flow Tracking
Mounir Nasr Allah, Guillaume Hiet, Muhammad Abdul Wahab, Pascal Cotret, Guy Gogniat, Vianney Lapotre

To cite this version:
Introduction

HardBlare proposes a software/hardware codeign methodology to ensure that security properties are preserved all along the execution of the system but also during files storage. The general context is to address Dynamic Information Flow Tracking (DIFT) that generally consists in attaching marks (also known as tags) to denote the type of information that are saved or generated within the system.

Let’s suppose that “print” function is public and the tag of a variable $x$ is underlined variable $x$.

Example code

```
int p = 3, s = 42;
int a = 3;
int x = p + s;
if (x != public) raise interruption
```

State of the art

Hybrid

<table>
<thead>
<tr>
<th>Software</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flexible security policies</td>
<td>Flexible security policies</td>
<td>Overhead</td>
</tr>
<tr>
<td></td>
<td>Multiple attacks detected</td>
<td>Overhead (from 300% to 3700%)</td>
</tr>
<tr>
<td>In-core DIFT</td>
<td>Low overhead (~10%)</td>
<td>Fixed Security policies</td>
</tr>
<tr>
<td></td>
<td>Invasive modifications</td>
<td>Invasive modifications</td>
</tr>
<tr>
<td>Dedicated CPU for DIFT</td>
<td>Low overhead (~10%)</td>
<td>Wasting resources</td>
</tr>
<tr>
<td></td>
<td>Few security policies</td>
<td>Energy consumption ($x$ 2)</td>
</tr>
<tr>
<td>Dedicated DIFT Coprocessor</td>
<td>Low overhead (&lt;10%)</td>
<td>Communication between CPU and DIFT</td>
</tr>
<tr>
<td></td>
<td>CPU not modified</td>
<td>Coprocessor</td>
</tr>
</tbody>
</table>

Static Analysis

» During the compilation phase, a static analysis is done on the LLVM intermediate representation produced from the source code, and propagated to the ARM backend for the machine code generation.

» The result of static analysis gives a list of dependencies between information containers (e.g., registers, memory spaces...) for every basic blocks which are stored on a dedicated section in a ELF File.

» During run-time, the Program Trace Macrocell (PTM) generates a trace containing the address for each committed instruction modifying the PC value.

» Annotations related to the basic block identified by its address, given by the trace, are processed by the coprocessor to propagate tags.

Definitions

- Tag dependencies: block contains annotations loaded when the program is launched.
- Memory tags: block contains tags related to information containers.
- Tag register file: contains tags related to CPU registers.

DIFT step-by-step

- ARM CoreSight Components export trace (for both CPUs) towards PL in PFT (Program Flow Trace) protocol.
- PFT Decoder decodes trace in usable format.
- Using decoded trace, DIFT Coprocessor reads tags dependencies block.
- DIFT Coprocessor looks for the tags either in memory or tag register file.
- DIFT Coprocessor computes tags depending on propagation rules.
- DIFT Coprocessor updates corresponding tags.
- DIFT Coprocessor checks for security policy violation and raise an interruption.

Some References