
Leandro Fiorin, Alberto Ferrante
Konstantinos Padarnitsas, Francesco Regazzoni

ALaRI, Faculty of Informatics
University of Lugano
Lugano, Switzerland
Outline

◆ Motivations
◆ Overview of SELinux
◆ SELinux on Embedded Systems
◆ Hardware-assisted SELinux
◆ Implementation
◆ Experimental results
Motivations (1/3)

◆ Security has gained importance in the embedded systems world
◆ Computing and communication increasingly pervade our lives:
  ▶ security and protection of sensitive data are extremely important
◆ Embedded Systems present several unique security challenges:
  ▶ Resource constrained
  ▶ Often very portable
Motivations (2/3)

◆ Vulnerability to attacks increases with the functionalities

◆ Possible attacks from:
  ► Cellular networks
  ► Bluetooth
  ► Network connection
  ► USB
  ► ...

◆ Threats to:
  ► data confidentiality
  ► data integrity
  ► availability of data and services
Motivations (3/3)

Security solutions available for computers are:
- often not scalable to ES
- partially effective
- rarely employed by final users
Process Isolation

◆ Restrict access of processes to system resources:
  ▶ It greatly limits the effects of security holes in the software.

◆ It can be implemented in different ways:
  ▶ Virtualization
  ▶ Fine-grain access control on resources
Virtualization

- Should be configured properly to provide secure isolation
- Resource demanding:
  - Not suitable for most embedded systems
Fine-grain access control on resources

- Allows controlling the access by processes to system resources
  - Files
  - Devices
  - Memory locations
  - Network sockets
  - ...

- Allows implementing sandboxing:
  - Each group of applications are confined in a different “environment”
    - No access to resources of other processes
Overview of SELinux (1/2)

- Designed by National Security Agency
- Introduced in 2001
- Included from 2003 as a standard kernel module in Linux
  - e.g., in Red Hat and Fedora Linux it is enabled by default
- Flexible mandatory method for controlling accesses to all the resources by processes:
  - Supplements Linux with a Mandatory Access Control (MAC)
    - Linux implements Discretionary access Control: Users may give or revoke access privileges to their own objects
Overview of SELinux (2/2)

◆ It defines the access and transition rights of:
  ► Users
  ► Applications
  ► Processes
  ► Files

◆ It governs the interactions of these entities:
  ► Based on a well-defined security policy
    ★ Set of rules
    ★ Written in accordance with the principle of least privilege: allowed actions must be explicitly defined
SELinux: elements (1/2)

◆ Subject:
  ► The system element requesting access to a resource:
    • Typically a process

◆ Object:
  ► The resource to be accessed:
    • a file, a network socket, a device, …
  ► Objects are divided in classes:
    • Specify the kind of object
SELinux: elements (2/2)

◆ Security server:
  ▶ Takes decisions (allowed/not allowed) on the requested actions
  ▶ Contains the SELinux Policy:
    • Each rule contains the triple subject, object, object class, and the associated access vector
    • The policy database is managed as a common single linked list hash table; the key is a concatenation of the subject identifier, the target identifier, and the object class

◆ Object Manager:
  ▶ Queries the Security Server
  ▶ Caches the decisions of the Security Server:
    • Access Vector Cache (AVC), managed in the same way as the policy database
  ▶ Enforces the access decisions
SELinux Architecture

Clean separation between policy enforcement code and decision-making core
SELinux – Query Example

taken from: Lars Strand, “RHEL5 SELinux: A benchmark”
http://blog.larsstrand.org/article.php?story=RHEL5-SELinux-Benchmark
## SELinux in Embedded Systems

- Overhead not negligible in terms of CPU usage and memory [1][2][3]:
  - Overhead on single operations might be high
  - Overhead on full applications depends on the application:
    - up to 10% for Apache
    - an average of 7% on the server activities
    - up to 66% for I/O intensive applications on embedded systems
    - Up to 8% for non-I/O intensive applications

<table>
<thead>
<tr>
<th>Operation</th>
<th>Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lmbench Null read</td>
<td>130</td>
</tr>
<tr>
<td>Lmbench Null write</td>
<td>147</td>
</tr>
<tr>
<td>Lmbench Create</td>
<td>168</td>
</tr>
<tr>
<td>Lmbench TCP</td>
<td>22</td>
</tr>
<tr>
<td>Unixbench 256B read</td>
<td>67</td>
</tr>
<tr>
<td>Unixbench 1024B read</td>
<td>44</td>
</tr>
<tr>
<td>Unixbench 4096B read</td>
<td>16</td>
</tr>
</tbody>
</table>

SELinux in ES – previous work

◆ Previous work focused on:
  ► reducing the size of policies by removing unnecessary configurations
  ► tuning kernel and userland by removing unneeded functions
  ► reducing memory usage by removing non-required data structures from kernel

◆ Selinux was ported to Android devices and to Nokia 770 internet tablet

◆ Security Enhanced Android released by NSA

◆ In our work we decrease the performances overhead:
  ► Lower energy overhead
  ► Lower computational cost
  ► No modifications on memory consumption
  ► No modifications to the architecture of SELinux
Hardware-assisted SELinux – Possible architectures

Arch 1

Arch 3
Hardware-assisted SELinux

- Main source of overhead is in the policy lookup operations:
  - Done at each access!

- Most of the lookups are normally executed in the AVC

The AVC can be implemented in hardware to reduce the time/energy overhead

Tests on mplayer playing a mp3 file over internet
Hardware AVC (HAVC)

- Mainly composed of a LUT
- The CAM stores keys to be looked up when performing access control
- The RAM stores the encoded access rights for the key:
  - Different encoding used to reduce memory occupation
Implementation (1/2)

◆ Prototype system implemented on a Xilinx ML510:
  ► Linux (kernel 2.6.34) running on the onboard PowerPC 440
  ► HAVC implemented on the Xilinx Virtex V FPGA
Implementation (2/2)

► AVC query and insert added as new instructions
► Use of Auxiliary Processor Unit (APU) for implementing the HW API
► The Control Unit:
  • Implements a FSM
  • Controls the execution of queries/insertions
Integration with the OS

- SELinux code modified for using the HAVC
- Added a hardware API for SELinux
- Proper interfaces for the hardware accelerator added by modifying the related function calls:
  - In `avc_has_perm()` the `avc_lookup()` function substituted with a call to the HW API
  - In `avc_has_perm()` the `avc_insert()` function substituted with a call to the HW API
### Results – Single lookup cost

<table>
<thead>
<tr>
<th>Benchmark</th>
<th><code>avc_lookup()</code> (SW)</th>
<th>Impr. (%)</th>
<th><code>avc_insert()</code> (SW)</th>
<th>Impr. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bitcount</td>
<td>315.70</td>
<td>95.6</td>
<td>3,476.53</td>
<td>99.0</td>
</tr>
<tr>
<td>blowfish (enc/dec)</td>
<td>535.65</td>
<td>97.4</td>
<td>3,460.72</td>
<td>99.0</td>
</tr>
<tr>
<td>susan</td>
<td>547.58</td>
<td>97.4</td>
<td>3,466.23</td>
<td>99.0</td>
</tr>
<tr>
<td>syscall write</td>
<td>497.76</td>
<td>97.2</td>
<td>3,447.06</td>
<td>99.0</td>
</tr>
<tr>
<td>syscall fstat</td>
<td>408.97</td>
<td>96.6</td>
<td>3,458.65</td>
<td>99.0</td>
</tr>
<tr>
<td>pipe latency</td>
<td>356.46</td>
<td>96.1</td>
<td>3,478.92</td>
<td>99.0</td>
</tr>
</tbody>
</table>
## Results – Overhead on benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>SW SELinux overhead (%)</th>
<th>HW SELinux overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MiBench</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bitcount</td>
<td>0.7</td>
<td>0.0001</td>
</tr>
<tr>
<td>blowfish (enc/dec)</td>
<td>1.4</td>
<td>0.1</td>
</tr>
<tr>
<td>susan</td>
<td>0.46</td>
<td>0.0043</td>
</tr>
<tr>
<td>stringsearch</td>
<td>2.9</td>
<td>0.01</td>
</tr>
<tr>
<td>basicmath</td>
<td>0.4</td>
<td>0.0087</td>
</tr>
<tr>
<td>jpeg (enc/dec)</td>
<td>2.7</td>
<td>0.7</td>
</tr>
<tr>
<td>ADPCM (enc)</td>
<td>8.1</td>
<td>3.7</td>
</tr>
<tr>
<td>syscall write</td>
<td>43.0</td>
<td>28.1</td>
</tr>
<tr>
<td>syscall stat</td>
<td>61.1</td>
<td>26.2</td>
</tr>
<tr>
<td>syscall fstat</td>
<td>66.2</td>
<td>21.0</td>
</tr>
<tr>
<td>syscall open/close</td>
<td>46.7</td>
<td>18.1</td>
</tr>
<tr>
<td>signal handler install</td>
<td>1.8</td>
<td>0.1</td>
</tr>
<tr>
<td>signal handler overhead</td>
<td>18.9</td>
<td>3.3</td>
</tr>
<tr>
<td>protection fault</td>
<td>21.4</td>
<td>4.1</td>
</tr>
<tr>
<td>pipe latency</td>
<td>29.0</td>
<td>4.0</td>
</tr>
<tr>
<td>UNIX socket</td>
<td>17.8</td>
<td>8.53</td>
</tr>
</tbody>
</table>
Results – Area and Energy Measurements

◆ CACTI was used for estimating the area of:
  ▶ RAM
  ▶ CAM

◆ Synopsys Design Compiler was used for estimating the area of the other components of the HAVC

◆ Wattch used to estimate energy consumption of
  ▶ software-only SELinux
  ▶ hardware-accelerated SELinux:
    • HAVC instructions were added to the instruction set
    • Energy consumption of a single HAVC instruction evaluated through CACTI and Synopsys
## Results – Area

<table>
<thead>
<tr>
<th># Entries in HAVC</th>
<th>Area (mm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>0.0342</td>
</tr>
<tr>
<td>64</td>
<td>0.0537</td>
</tr>
<tr>
<td>128</td>
<td>0.0957</td>
</tr>
<tr>
<td>256</td>
<td>0.1256</td>
</tr>
</tbody>
</table>

- A HAVC with 64 entries uses an area that is 2.5% the one of a PowerPC 405-F6 implemented on a 90nm technology.
### Results - Energy

<table>
<thead>
<tr>
<th>Function</th>
<th>Average Energy (nJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW-only lookup</td>
<td>826</td>
</tr>
<tr>
<td>SW-only insert</td>
<td>2,923</td>
</tr>
<tr>
<td>HW-SW lookup</td>
<td>0.0106</td>
</tr>
<tr>
<td>HW-SW insert</td>
<td>0.0107</td>
</tr>
</tbody>
</table>

◆ Energy consumption for single queries/_inserts is orders of magnitude lower when the hardware accelerator is used.
Conclusions

◆ We proposed a solution for reducing the performance overhead due to SELinux
◆ We tested the solution with good performances by implementing it on a FPGA-based platform
◆ We evaluated area and energy of an ASIC implementation
Future Work

◆ Write a policy targeted for embedded systems:
  ▶ Generic policy
  ▶ Policy for sandboxing of non-trusted applications

◆ Improve the management of the HAVC

◆ Optimize the architecture of the accelerator for multicore/multiprocessors

◆ Investigate the adoption of the accelerator for PCs and servers.
Thanks for your attention!

alberto.ferrante@usi.ch