Runtime Code Polymorphism as a Protection against Physical Attacks
Damien Couroussé, Bruno Robisson, Thierno Barry, P Jaillon, Olivier Potin

To cite this version:

HAL Id: emse-01232662
https://hal-emse.ccsd.cnrs.fr/emse-01232662
Submitted on 23 Nov 2015
Core Idea: Runtime Code Polymorphism

Definition

Regularly changing the behaviour of a (secured) component, at runtime, while maintaining unchanged its functional properties.

What for?

- Protection against reverse engineering of SW
- The secured code is not available before runtime
- The secured code regularly changes its form (code generation interval $\omega \geq 1$)
- Protection against physical attacks
- Polymorphism changes the spatial and temporal properties of the secured code: side-channel & fault attacks
- Combine with usual SW protections against focused attacks

How?

- deGoal: runtime code generation for embedded systems
- Fast code generation
- Tiny memory footprint: proof of concept on TI's MSP430 (512 bytes of RAM)

Compilettes & deGoal in a Nutshell

A compilette is:
- an ad hoc code generator that targets one kernel
- aimed to be invoked at runtime

Example: polymorphic AES

Polymorphic implementation of the SubBytes function:

```c
void gen_subBytes(unsigned char* state) {
    #define SUBBYTES_SIZE 8
    #define SUBBYTES_ADDR (0x1000)

    for (int i = 0; i < SUBBYTES_SIZE; i++) {
        state[i] = SUBBYTES_ADDR[i];
    }
}
```

Execution times (in cycles), over 1000 runs:

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>max</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference</td>
<td>6385</td>
<td>6385</td>
<td>6385</td>
</tr>
<tr>
<td>code generator</td>
<td>5671</td>
<td>12910</td>
<td>9345</td>
</tr>
<tr>
<td>polymorphic instance</td>
<td>7185</td>
<td>9745</td>
<td>8303</td>
</tr>
</tbody>
</table>

Impact of the code generation interval $\omega$:

<table>
<thead>
<tr>
<th>$\omega$</th>
<th>k</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.76</td>
<td>53.0%</td>
</tr>
<tr>
<td>5</td>
<td>1.59</td>
<td>18.4%</td>
</tr>
<tr>
<td>20</td>
<td>1.37</td>
<td>2.1%</td>
</tr>
<tr>
<td>100</td>
<td>1.31</td>
<td>1.1%</td>
</tr>
</tbody>
</table>

k: overhead vs. reference implementation

%: percentage contribution of runtime code generation to the performance overhead

References

- Runtime code generation for micro-controllers with less than 1kB RAM
- Instruction scheduling for VLIW processors