INTRINSEC, AN INTRINSICALLY SECURE PROCESSOR

Olivier SAVRY, Thomas HISCOCK, Mustapha EL MAJIHI
AN AVALANCHE OF VULNERABILITIES

- Security is not anticipated
- Security skills are difficult, rare and expensive
- Even for experts, it is difficult to deal with certain vulnerabilities
SOME ANSWERS TO DESIGN A SECURE EMBEDDED SYSTEM

• **Current paradigm:**
  • Append a Secure Element (or a virtual one like TEE) to the Application Processor
  • Application processor still have vulnerabilities and so requires many security functions
    • Secure boot, secure channel or mailbox, secure storage

• **New paradigm:**
  • Design an Application Processor which ensures its security by itself
  • An intrinsically secure application processor
  • « Plug & Secure » : compile your code with the good compiler and on the good platform without worrying about forgetting vulnerabilities
  • **IntrinSec : a family of Intrinsically Secure Application Processors for IoT and CPS**
PROCESSORS VULNERABILITIES

FAULT INJECTION & SIDE-CHANNEL

Spectre & Meltdown
- cache timing side-channel
  - Flush + Reload
  - Prime + Probe
  - Flush + flush
  - Cachebleed
  - Evict + Time
  - TLB

SRAM Memory:
- Cell to cell coupling -> software fault injection in cache memories with transistors < 10 nm

DRAM Memory:
- RowHammer
- DRAMA

FLASH Memory
- bit to bit programming interference with MLC

Many of those vulnerabilities could be treated as a lack of confidentiality & integrity of Data and Instructions
An holistic approach to address most threats with as few countermeasures as possible

Authenticated encryption of data AND instructions is the way:

- To provide confidentiality against coldboot, reverse engineering, to protect software IP
- To provide authenticity and integrity against software or data modifications, faults injection in the memory hierarchy, faults injection during runtime

« Encrypt then MAC » structure

- Random access ensures with Nonce=f(@)
- Cumbersome scheme but allows to answer lots of security objectives
- RI5CY 32 bit core On FPGA (Xilinx Zynq 7020)

Carefull use of PMP for M-mode only access

APB slave to:
- Set 128 Bits password
- Verify password
- Lock/Unlock
AUTHENTICATED ENCRYPTION DESIGN : CHALLENGING REQUIREMENTS

- Lightweight AND low latency design
- Cipher/decipher with the same block
- Encrypt instructions and data of programs
- Support associated data
- Random access to memory : Tweak or IV/Nonce
- Memory addressing, alignment
- compatible with operating systems (load/store of code lines, moving pages in physical memory, …)
- Compatible with already compiled binairies
- «Encrypt then MAC» with 128 bit key, 64 bits MAC
To improve throughput, instructions/data are bundled by four 32-bit words + metadata + MAC (64 bits)

- 256 bit memory alignment
- 100 % Overhead for memory size
AUTHENTICATED ENCRYPTION WITH ASCON

- ASCON sponge-based cipher: CAESAR finalist
- 128 bit key
- 128 bits ciphered data + 64 bits associated data + 64 bits tag
- Required 40 * permutation p
- Only one instance of p is implemented
- The permutations are computed asynchronously in 4 CPU cycles
- Fast + lightweight

Other candidates in the recent lightweight NIST competition must also to be evaluated
• This encryption scheme is totally independent of the compiler, it can be done on any binaries
PROGRAM ENCRYPTION WITH CFI (CONTROL FLOW INTEGRITY)

BB0 at @0

<table>
<thead>
<tr>
<th>Instr1 ⊕ MASK0</th>
<th>Ciphered with Nonce =@0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instr2 ⊕ p(MASK0)</td>
<td>Ciphered with Nonce =@0+1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(loadMASK MASKBRANCH,MASK1) ⊕p^{n-1}(MASK0)</td>
<td>Ciphered with Nonce =@0+n-1</td>
</tr>
<tr>
<td>(BEQ rs1,rs2,@1) ⊕ p^n(MASK0)</td>
<td>Ciphered with Nonce =@0+n</td>
</tr>
</tbody>
</table>

Conditional branches or jal

BB1 at @1

<table>
<thead>
<tr>
<th>Instr1 ⊕ MASK1</th>
<th>Ciphered with Nonce =@1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instr2 ⊕ p(MASK1)</td>
<td>Ciphered with Nonce =@1+1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

A new instruction « loadMASK » is introduced by the compiler to set a MASK that will be used by the next jump to unmask the next BB.

- Modification of LLVM and COMPCERT to include Masks and new instruction LoadMASK

Zürich RISC V WORKSHOP | SAVRY Olivier | 12/06/2019
PROGRAM ENCRYPTION WITH CFI (CONTROL FLOW INTEGRITY)

• Management of Indirect Jump (JALR Jump and Link Register)

<table>
<thead>
<tr>
<th>BB0 at @0</th>
<th>Ciphered with Nonce =@0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instr1 ⊕ MASK0</td>
<td>Ciphered with Nonce =@0+1</td>
</tr>
<tr>
<td>Instr2 ⊕ p(MASK0)</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>(jalr rd,rs=@1) ⊕ p₀(MASK0)</td>
<td></td>
</tr>
</tbody>
</table>

Instruction « loadMASK » is enriched to retrieve a mask to a certain address and not only a static value.
LoadMASK MASK_register, @MASK

• Compatible with old uninstrumented binaries (the loader just have to set MASK register to 0) and with dynamic libraries

• For function return, the mask to decipher at return address is simply pushed on the stack with the ra

<table>
<thead>
<tr>
<th>BB1 at @1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instr’1 ⊕ MASK1</td>
</tr>
<tr>
<td>Instr’2 ⊕ p(MASK1)</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Ciphered with Nonce =@1-1
Ciphered with Nonce =@1
Ciphered with Nonce =@1+1
CONCLUSION

• Authenticated encryption of data and instructions in the memory hierarchy
  • Allows to perform many security functions
• 100 % overhead on memory
• High degree of compatibility
  • with old binairies (encryption independent of compilation)
  • With dynamic librairies,
  • With OS
• LLVM, CompCert modified compilers to include CFI
  • Introduction of new instruction implies an overhead of 15% on the code size.
• Proof of concept on FPGA with RISC V Pulpino
• Future challenges
  • Protection of the pipeline against fault injection
  • Protection of the pipeline against side-channel leakages
  • Resilience when attacks are performed
  • Intrinsic spatial and temporal memory safety
  • Implementation on 64 bit processors (Ariane ?)
  • Development of multitasks OS : freeRTOS, Linux with intrinsic isolation of processes