Who and why should fear hardware trojans?
Disclaimer:

The presented work disseminates the results of the author’s spare time activities done solely using his own, private resources. Therefore, the views and opinions expressed in this presentation are those of the author and author only and do not necessarily reflect the official policy or position of his employer. Examples of analysis performed within this article are only for demonstration purposes and do not necessarily reflect real-world products.
Can we trust integrated circuits and silicon chips?
Do we have a choice? Due to:
• technological barriers,
• and high production costs
we must count on manufacturers’ honesty and popularity.
Therefore, we commonly buy/use:
• high volume products from leading producers, which should enable faster detection of potential threats,
• and have legal consequences in case of threats.
(possible in mainframe and telecommunication, difficult in embedded)
Questions of this Presentation

• Is there a reason for concern?

• Is the equipment from a large supplier safe?

• What is the threat posed by hardware trojans?

• Is it worth getting a closer look at them?

This presentation presents introduction to the domain.
What are hardware trojans?

**Definition:** function of a hardware component, hidden from the user, which can *add, remove or modify* the functionality of a hardware component and, therefore, reduce its reliability or create a potential threat.

Constructed from:
- **payload** – modification of a circuit
- **trigger** – signal activating the payload
  (combinational or sequential)
Relation to Software Trojans

Similarities:
• malicious intention, e.g., attacks against data confidentiality, integrity and system’s availability
• evasion of detection, and seldom activation,

Differences:
• cannot be removed post-deployment (no updates)
• do not spread, must be manufactured
• high production costs (equipment and skilled labor)

Do not forget about HW/SW co-design!
‟Hardware Trojans“ in Google Scholar (May 2018)

Snowden Affair

Results gathered by author
Characteristics of Trojans

**Payload (Physical)**

- **Type**
  - functional
  - parametric

- **Size**
  - small (transistors / wires)
  - large (gates, circuits)

- **Distribution**
  - tight
  - loose

- **Structure**
  - modify layout
  - no changes

**Trigger (Activation)**

- **Externally Activated**
  - Receiver
  - Access data

- **Internally Activated**
  - Always On
  - Condition Based

  - sensor (voltage, temperature, external)
  - logic (internal state, clock/counter, input – data, instruction, interrupt)
Talk's Outline

- Motivation
- Work principles and objectives of a HW trojan
- CPU Example (+ Demo of an exemplary attack)
- Where and when can a hardware trojan be introduced?
- Defense methods and their costs
- Summary
Hardware Aspects of OS Security

The security of an operating system is based on the assumption that the processor is operating according to a strict specification and a known set of predefined rules.

**Commonly applied:** hierarchical protection domains (protection rings) - introduced in the 70’s for MULTICS.

• at least two modes of operation (hypervisor and user)
• in hypervisor mode, kernel has access to all commands and addresses
• only a subset of commands is available in user mode
• transition can happen only according to a predefined set of rules (e.g. syscalls and interrupts)
### Modes of Processor Work

**Privileged**
An instruction (or register) that can only be executed (or accessed) when the processor is in supervisor mode (when PSR[S]=1).

**Supervisor Mode**
A processor state that is active when the S bit of the PSR is set (PSR[S]=1).

**Supervisor Software**
Software that executes when the processor is in supervisor mode.

**Trap**
A vectored transfer of control to supervisor software through a table whose address is given by a privileged IU register (the Trap Base Register (TBR)).

**User Mode**
A processor state that is active when the S bit of the PSR is not set (when PSR[S]=0).

**User Application Program**
A program executed with the processor in user mode. Also simply called "application program." Note that statements made in this context are implementation-dependent.

---

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUL67c</td>
<td>Multiply Step (and modify icc)</td>
</tr>
<tr>
<td>UMUL(UMUL6c)</td>
<td>Unsigned Integer Multiply (and modify icc)</td>
</tr>
<tr>
<td>SMUL(SMUL6c)</td>
<td>Signed Integer Multiply (and modify icc)</td>
</tr>
<tr>
<td>UDIV(U DIV6c)</td>
<td>Unsigned Integer Divide (and modify icc)</td>
</tr>
<tr>
<td>SDIV(S DIV6c)</td>
<td>Signed Integer Divide (and modify icc)</td>
</tr>
<tr>
<td>SAVE</td>
<td>Save caller’s window</td>
</tr>
<tr>
<td>RESTORE</td>
<td>Restore caller’s window</td>
</tr>
<tr>
<td>Bicc</td>
<td>Branch on integer condition codes</td>
</tr>
<tr>
<td>FBfic</td>
<td>Branch on floating-point condition codes</td>
</tr>
<tr>
<td>CBiccc</td>
<td>Branch on coprocessor condition codes</td>
</tr>
<tr>
<td>CALL</td>
<td>Call and Link</td>
</tr>
<tr>
<td>IMPL</td>
<td>Jump and Link</td>
</tr>
<tr>
<td>RETT?</td>
<td>Return from Trap</td>
</tr>
<tr>
<td>Trap</td>
<td>Trap on integer condition codes</td>
</tr>
<tr>
<td>RDASR?</td>
<td>Read Ancillary State Register</td>
</tr>
<tr>
<td>RDY</td>
<td>Read Y Register</td>
</tr>
<tr>
<td>RDPSSR?</td>
<td>Read Processor State Register</td>
</tr>
<tr>
<td>RDWDIMF</td>
<td>Read Window Invalid Mask Register</td>
</tr>
<tr>
<td>RDWB?</td>
<td>Read Trap Base Register</td>
</tr>
<tr>
<td>WRASR?</td>
<td>Write Ancillary State Register</td>
</tr>
<tr>
<td>WRY</td>
<td>Write Y Register</td>
</tr>
<tr>
<td>WRPSR?</td>
<td>Write Processor State Register</td>
</tr>
<tr>
<td>WRWIMF?</td>
<td>Write Window Invalid Mask Register</td>
</tr>
<tr>
<td>WRWB?</td>
<td>Write Trap Base Register</td>
</tr>
<tr>
<td>STBAR</td>
<td>Store Barrier</td>
</tr>
<tr>
<td>UNIMP</td>
<td>Unimplemented</td>
</tr>
<tr>
<td>FLUSH</td>
<td>Flush Instruction Memory</td>
</tr>
<tr>
<td>FProp</td>
<td>Floating-point Operate: FMOV(x,d,q), Fx(x,d,q), FQT(x,d,q), FQTO(x,d,q), FQTOF(x,d,q), FMUL(x,d,q), FMULM2(x,d,q), FMULMF(x,d,q), FCMPS(x,d,q), FCPME(x,d,q)</td>
</tr>
<tr>
<td>CProp</td>
<td>Coprocessor Operate: implementation-dependent</td>
</tr>
</tbody>
</table>

1. privileged instruction
2. privileged instruction if the referenced ASR register is privileged
Hardware Implementation – RISC Pipeline

Processor Datapath

Instruction +1
Instruction +2
Instruction +3
Instruction +4
Hardware Implementation - Pipeline

**IF**
- Decode instruction's address
  - Is the selected address available in the processor mode?
  - EXCEPTION Memory protection violation
  - Fetch Instruction
  - Decode Instruction
  - EXCEPTION Illegal Instruction

**EX**
- Execution
  - Decode memory address
  - EXCEPTION Memory protection violation

**ME**
- Conduct Operation
  - Is the selected address available in the processor mode?
  - EXCEPTION Illegal Instruction
  - Conduct Operation
  - EXCEPTION Illegal Access

**WB**
- Conduct Operation
  - Is the selected register available in the processor mode
How it is implemented?

Example using Leon3 Sparc Processor OpenSource VHDL implementation
Trojan Design

Payload
• change the status of the PSR [Processor State Register]
• and switch to the hypervisor mode

Trigger
• selected ASM command available in the user mode
• with the selected operands
• e.g., add OP CODE which results in 878787

HW/SW co-design (trigger in SW, payload in HW)
Trojan Payload Code (CPU Backdoor, PWNing2017)

Example using Leon3 Sparc Processor
Do I need a foundry to test HW trojans?

Example based on QEMU implementation of the SPARC architecture (compatible with VHDL from Gaisler)

- QEMU emulator version 2.12.50 (v2.12.0-rc3-71-g6af2692e86-dirty)
- buildroot-2018.02.1 for cross compilation
- and kernel 4.11.12 #1 Wed May 2 10:20:04 CEST 2018 sparc GNU/Linux

- Implementation details and sources
  https://adamkostrzewa.github.io
QEMU
asm
translation for SPARC

qemu/target/sparc/translate.c

```c
#ifdef CONFIG_USER_ONLY
} else if (xop == 0x21) { /* rdpsr */
#endif

if (!supervisor(dc)) {
    goto priv_insn;
}

update_psr(dc);
gen_helper_rdpsr(cpu_dst, cpu_env);

#else

CHECK IU FEATURE(dc, HYPV);
if (!hypervisor(dc))
    goto priv_insn;

rs1 = GET_FIELD(insn, 13, 17);
switch (rs1) {
    case 0: // hstate
        tcg_gen_ld_i64(cpu_dst, cpu_env,
                       offsetof(CPUSPARCState, hstate));
        break;
    case 1: // hstate
        // gen_op_rdhstate();
        break;
    case 3: // hintp
        tcg_gen_mov_tl(cpu_dst, cpu_hintp);
        break;
    case 5: // hta
        tcg_gen_mov_tl(cpu_dst, cpu_hta);
        break;
    case 6: // hver
        tcg_gen_mov_tl(cpu_dst, cpu_hver);
        break;
    case 31: // hstick cmp
        tcg_gen_mov_tl(cpu_dst, cpu_hstick_cmp);
        break;
    default:
        goto illegal_insn;
}
#endif

gen_store_gpr(dc, rd, cpu_dst);
break;
```
Trojan Payload
in SDIV
(signed divide)

qemu/target/sparc/helper.c

```c
static target_ulong do_sdiv(CPUSPARCState *env, target_ulong a,
                             target_ulong b, int cc, uintptr_t ra)
{
    int overflow = 0;
    int64_t x0;
    int32_t x1;

    x0 = (a & 0xffffffff) | ((int64_t) (env->y) << 32);
    x1 = (b & 0xffffffff);

    if(x0 == 1024 & x1 == 64){
        CPUState *cs = CPU[sparc_env_get_cpu(env)];
        SPARC CPU *cpu = SPARC_CPU(cs);
        CPUSPARCState *sc = &cpu->env;
        sc->psrs = i;
    }

    if (x1 == 0) {
        cpu_raise_exception_ra(env, TT_DIV ZERO, ra);
    } else if (x1 == -1 && x0 == INT64_MIN) {
        x0 = INT32_MAX;
        overflow = 1;
    } else {
        x0 = x0 / x1;
        if ((int32_t) x0 != x0) {
            x0 = x0 < 0 ? INT32_MIN : INT32_MAX;
            overflow = 1;
        }
    }

    if (cc) {
        env->cc_dst = x0;
        env->cc_src2 = overflow;
        env->cc_op = CC_OP_DIV;
    }

    return x0;
}
```

trigger division
with operands
1024 and 64
Live Demo
Talk's Outline

- Motivation
- Work principles and objectives of a HW trojan
- CPU Example (+ Demo of an exemplary attack)
- Where and when can a hardware trojan be introduced?
- Defense methods and their costs
- Summary
Phase 1: Logic Design

Trojan Implementation can be done:
• by adding a new functionality, usually in HDL language (as in the CPU example),
• using a methodology similar to software projects.

Relatively cheap – designer is working on a high abstraction level
Difficult to hide – most of developers are included in Phase 1

Obfuscation – code semantically correct but difficult to understand
Easier in big projects with backwards compatibility, e.g. x86 architecture
Phase 2: Layout (Synthesis)

Synthesis process (~ code compilation)
• trojans added as modification of electric circuits
• many hours of work, e.g., usually layouts are heavily optimized
• ~ analogy of "in-compilation" or post-compilation modification of code

Easier to hide – less developers than in Phase 1
More expensive – requires qualified and experienced personnel

„White-box cryptography“ - produce complicated layouts to hide malicious circuits or components
Phase 3: Mask Design

Modify masks to add or change behavior of components

- new sets of masks,
- at the far-end of the design process.

Very hard to detect – modifications on very low abstraction level

Very expensive – highly skilled personnel, expensive tools and know-how, preferably also market position

Each phase is realized by a different entity within a company or even by different sub-contractors in case of OEMs!
Talk's Outline

- Motivation
- Work principles and objectives of a HW trojan
- CPU Example (+ Demo of an exemplary attack)
- Where and when can a hardware trojan be introduced?
- Defense methods and their costs
- Summary
Post-Manufacturing Trojan Detection

The process is difficult and expensive!
Based on the „golden chip“ principle - comparison vs. correct chip

Open the case and use the optical reverse-engineering:
• typically plastic package (epoxy), smart-card package etc.

Several methods to achieve this goal:
• mechanical grinding easy and cheap
  ▪ a single scratch can destroy the chip
• chemical acid to dissolve the case
  ▪ requires resources (personnel, laboratory equipment)
Reading Chip

Size of the chip is a barrier
For amateurs chips from 70s, 80s early 90s

Highly optimized
• Due to technical & economic reasons
Works in parallel, legacy functions (x86)

Make photos of each layer, e.g. with a scanning electron microscope
Compare them to the layout to detect metal or polysilicon wires
Comparison Against Golden-Chip

Pros:
• very high reliability of detection
• if you have a golden chip

Cons:
• expensive
• time consuming
• destroys chip under analysis
applicable only to small number of chips ....
• is it enough?
Example Inverter

Where is the trojan?

Optical inspection is not enough!

based on [Paar, 2017]
Dopant-Trojans

Can be used to limit the number of cipher combinations! Especially important in telecommunication, e.g. IEEE 802.11

Example: AES, based on [Paar, 2017]

- random register
  - 010101001010…….000000
- known bits fixed by trojan
- only 32 random bits, testing them will take several seconds
  - 010….001010…….000000
- counter register
- AES
- crypto key

Source: wikipedia commons
Dopant-Trojans (Phase 4 - Wafer)

Requires modification of the dopant mask

Pros:
• no additional transistors or wires
• optical inspection is not enough!
• may be also included in the „golden chip“!

Cons:
• limits certain functionalities, „damaged chip parts“
• functional testing will find error right away!
• necessary anti-test functions for known test inputs!

Source: wikipedia commons
Error or Trojan? Or both?

Once implemented it is hard to change or modify a HW component. Thus, electronic circuits require extensive testing during the design phase.

Selected errors could be used as attack vectors, e.g.:
• certain bugs left on purpose / it’s not a feature, it’s a bug
• interesting in this context Intel Bugs: Meltdown and Spectre

Convenient excuse for the manufacturer
Otherwise high costs and severe consequences!
Talk's Outline

- Motivation
- Work principles and objectives of a HW trojan
- CPU Example (+ Demo of an exemplary attack)
- Where and when can a hardware trojan be introduced?
- Defense methods and their costs
- Summary
Summary

Who should fear HW trojans?
Everyone

Why?
Very difficult to detect
Leverage all software security mechanisms

What to do?
Build skilled force in HW domain (personnel and tools)
Evaluate HW products, will make attacker’s life more difficult
My blog with sources and materials

https://adamkostrzewa.github.io/

and my twitter for people interested in hardware hacking

https://twitter.com/systemWbudowany

Tank you for your attention!
Questions?
<table>
<thead>
<tr>
<th>Equipment</th>
<th>Basic</th>
<th>Advanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microscope</td>
<td>Simple New from 300 euro Used from 50 euro</td>
<td>Stereo New: about 1500 Euro Used: from 250 Euro</td>
</tr>
<tr>
<td>Tools</td>
<td>Mechanical tools About 20 euros</td>
<td>Microscope camera New: about 350 Euro Used: from 150 Euro</td>
</tr>
<tr>
<td>Ultrasonic cleaner</td>
<td>New 50 euro Used from 20 euro</td>
<td></td>
</tr>
<tr>
<td>Solvents and glassware</td>
<td>New: about 20 Euro</td>
<td></td>
</tr>
</tbody>
</table>

Source “Uncaging Microchips Techniques for Chip Preparation” Peter Laackmann Marcus Janke, CCC2014
```c
#include <stdio.h>
#include <stdlib.h>

int main()
{

    printf("Demo Application - Trojan Trigger\n");
    int a = 1024;
    int b = 63;
    int c = a / b;

    // int psr = get_psr();

    printf("a=%d b=%d \n\nc = a / b = %d \n", a, b, c);
    getchar();

    int *x = NULL;
    int y = *x; // null pointer dereference

    return 0;
}
```
Phases of Chip Design

**Phase 1**
Logical Design

**Phase 2**
Layout

**Phase 3**
Mask Design

**Phase 4**
Wafer Production

*Hardware Description Language, HDL*
VHDL, Verilog, systemVerilog

Synthesis of HDL into the electronic circuits

Creation of matrices for serial productions

Chemical process, doping

Based on [Laackmann, Janke:2015]
Where the Processor’s State is Stored?

- **Hardwired bits to identify an implementation or class of implementations of the architecture**
- **Version** (implementation-dependent)
- **Integer Condition Codes**
- **Enable Cooprocessor**
- **Enable Floating-Point Unit**
- **Enable Traps**
- **Processor Interrupts Level**
- **Current Window Pointer**
- **Supervisor Bit**
  - If 1 kernel mode
  - If 0 user mode
- **Processor State before an exception**

<table>
<thead>
<tr>
<th>impl</th>
<th>ver</th>
<th>ic</th>
<th>reserved</th>
<th>EC</th>
<th>EF</th>
<th>PIL</th>
<th>S</th>
<th>PS</th>
<th>ET</th>
<th>CWP</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:28</td>
<td>27:24</td>
<td>23:20</td>
<td>19:14</td>
<td>1</td>
<td>1</td>
<td>11:</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4:</td>
</tr>
</tbody>
</table>

**Reserved bits**