Formal models of hardware peripherals

Introduction

Security and privacy are important concerns. We study the mechanisms that prevent abuse of hardware peripherals. Our goal is to build and proof-check a model of secure hardware peripherals. Our networks, operating systems and applications must interact with secure hardware peripherals: Input/output devices (UART, USB, network controllers), interrupt controllers and coprocessors (GPU, FPGA).

There are formal models to test software and to guarantee data security. Hardware peripherals and their connections are usually not part of such tests.

What's the problem?

Schneier's problem is that everybody can outsmart themselves and consequently you can also intrude on yourself. ("any security expert can invent a cipher that he himself can't break"). See also common data security patterns that use the placeholder actor names Alice, Bob, Carol, Eve, Mallory, Peggy, Victor, Trent etc for different roles and functions. Typically Alice and Bob are the ones sending data and the other guys are actors with different functions.

Methodology

We build, examine and test hardware security models. We use tools that check Hoare contracts.

- We learn proof-checking
- We build a hardware peripheral security model and proof-check it. We test it with Quartus, Promela, LTL, Büchi automata, SPIN or similar tools
- We verify Hoare logic using Hoare contracts
- UART case studies
- We learn the HOL proof-tool Isabelle and how it's better than the check testing framework that we already use
- Decide which hardware to proof-check and secure: UART, USB, GPU, FPGA, RAM, storage, networks
- We learn hardware abstraction layers (graph theory (python/oop?))
- We read the relevant docs about asserting hardware and firmware
- We look how to prove VHDL and/or Verilog

- We see what Quartus can, can’t, won’t or needs
- We analyse, investigate, build and check
- We look at brute forcing software and brute forcing hardware
- We discuss different models for example different architectures
- We look at a specific software problems such as Ken Thompson’s gcc attack (1984). The Unix backdoor was compiled into next version of the compiler and not even visible in the source code.
- We examine hardware Trojans
- We mention common problems that appear and common patterns e.g. flattening
- We study the OSI layers specifically networks (ISO 8348, X.213), connections(ISO 8886, X.212) and the physical layer (ISO 10022, X.211)

References

[1] P. Mishra et al: Hardware IP Security and Trust, Springer 2017
[2] Ross Anderson, “Security Engineering”, https://www.cl.cam.ac.uk/~rja14/book.html
[3] M. Hicks, M. Finnicum, S. King, M. Martin, and J. Smith, “Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically,” in IEEE Symposium on Security and Privacy (SP) 2010, pp. 159–172
[4] Formality, User Guide, http://www.vlsiip.com/formality/ug.pdf, 2007.
[5] 7] B. C¸ akir and S. Malik, “Hardware trojan detection for gate-level ics using signal correlation based clustering,” in Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition. EDA Consortium, 2015, pp. 471–476.
[6] Wishbone bus [Online]. Available: http://opencores.org/opencores,wishbone

USB

- QEMU info page
- QEMU git info docs
- USB drivers
- Communicating with hardware
- Device controller source e.g. OHCI, omap-specific
- OHCI controller specs

Time plan

Step 1 (February-March): Investigate, analyse and build theoretical model (which mathematical model?)
Step 2 (April-May): Program, verify, test and check with real hardware

Methodology

Use hol.

```
$~/proj/HOL$ poly < tools/smart-configure.sml
Poly/ML 5.6 Release

HOL smart configuration.

Determining configuration parameters: holdir OS poly polymllibdir

Looked in poly's sister lib directory /usr/lib
and couldn't find libpolymain.a
Please write file tools-poly/poly-includes.ML to specify it.
This file should include a line of the form

  val polymllibdir = "path-to-dir-containing-libpolymain.a";
$~/proj/HOL$ bin
bin/  bind
$~/proj/HOL$ ./bin/bui^C
$~/proj/HOL$ ls bin/
hol.ML  noninterhol.ML  README
$~/proj/HOL$ more bin/README
This is the "bin" directory for HOL. A collection of executable files (hol,
Holmake, etc.) will be placed here when you build HOL.
$~/proj/HOL$ poly < bin/hol.ML
Poly/ML 5.6 Release
No holstate argument provided
$~/proj/HOL$
```

Apps

- Wireshark. Sniffer and protocol analyzer tcpdump
- Command-line based sniffer netwox
- netcat (nc) - Lots of different tools, can be used for a simple client/server
- Nmap
- Iptables
- ufw (“uncomplicated firewall”)

The verification could be performed using the HOL theorem prover (which? holzero?) and a model of the hardware (which?). We prove isolation of code and data of a hardware peripheral.

We demo on real hardware using a hypervisor that protects a peripheral.