|  |
| --- |
| Secure Embedded Computing Systems |
| Hardware Reverse Engineering |
| Student Workbook |

|  |
| --- |
| Russ Bielawski, Bill Hass  6/26/2017 |

1. Introduction

Hardware reverse engineering (HW:RE) is part of a process to achieve a higher goal. From the eyes of an attacker, this process can generally be broken into three stages: HW:RE, software RE (SW:RE), and exploitation.

Different applications require different tools, but there are a core set of basic tools that any hardware hacker should have to be successful:

|  |  |  |  |
| --- | --- | --- | --- |
| * Screwdrivers * Razor blades * Tweezers * Pliers * Strippers * Q-tips * Paper clips | * Soldering iron * Multimeter * DC power supply   + Battery   + Wall wart   + Bench-top | * Jumper wires * Headers * Patch wire * Micro clamp probes * Solder * Desoldering wick or pump * Flux | * Linux computer * USB A, B, mini, micro * UART-to-USB (FTDI) cable * Bus Pirate or JTAGGER |

A more thorough overview of tools can be found in the back of this manual.

* 1. Goals
     1. Primary Goals:

#### Exploitation

Ultimately, as an adversary we want to find a method to exploit the device. There are advanced hardware techniques such as power analysis and glitching (See ChipWhisperer) or cold-boot attacks that can directly recover cryptographic keys or bypass security mechanisms. These are beyond the scope of this class, but keep in mind that the techniques do exist and there are ways to go directly from HW:RE to Exploitation.

More commonly, however, the job of the HW:RE stage is reconnaissance and to extract the firmware for SW:RE.

#### Memory Extraction

Computers use memory to store data. Cryptographic keys, program code, proprietary information, operational logs, and debugging logs are all types of data an attacker is interested in and can be obtained by dumping memory. We consider memory extraction to be the most important job of HW:RE because once the memory is obtained, it would be considered SW:RE to perform further analysis on it.

Manufacturers may take steps to make it more difficult or prevent firmware from being extracted, but in some instances firmware images are available directly from the manufacturer’s website as software updates. Before attempting to extract memory, it’s best to see if the firmware is already available. Nonetheless, memory extraction is still required to obtain cryptographic keys, operational logs, and debugging logs.

#### Interactive Debugging

* + 1. Secondary Goals:

#### Determine capabilities of the module – What can it do?

#### Find and interrogate components from within – What can it tell me?

1. Circuit Identification (~30 minutes)

Involves carefully taking apart the assembled component to get to the circuitry and performing circuitry reconnaissance to identify areas of interest.

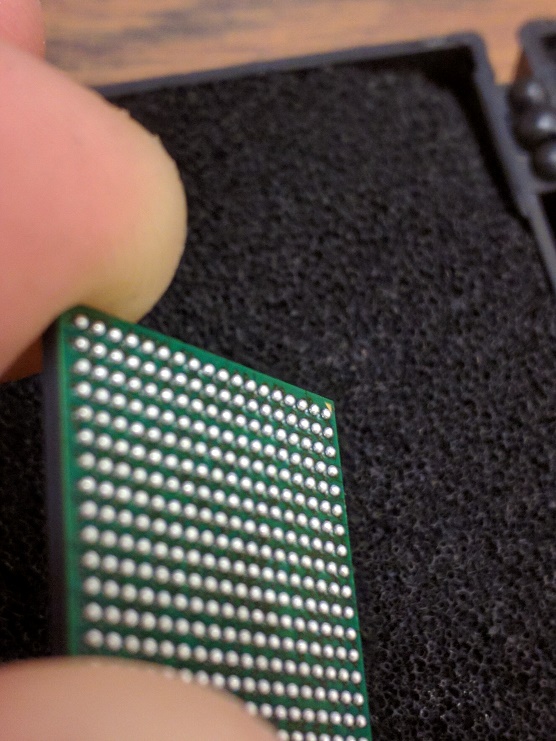
* 1. Visual Analysis

For many

Initial lay-of-the-land inspection.

* How are components mounted?

*Through-hole is your friend. Flat pack components expose all pins. BGA is your enemy.*

**

* Are there barriers or protections in place?

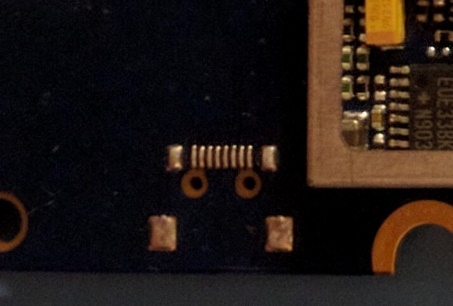
*EMF shielding and robustness coatings can make our job difficult.*

* What are the populated interfaces?

*Things like USB, vehicle connectors, and hidden connectors.*

* Where are interesting areas (depopulated pads, test-points, unsure)?

*Development and debug interfaces are typically removed before production.*

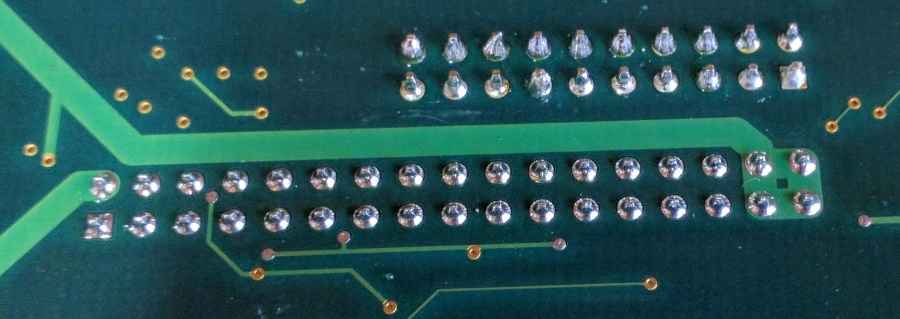
**

* How do components relate to one another?

*Observe general layout, components will be closest to what they interface with.*

* How is the board powered?

*You will need to power the board. Good starting point for tracing. Big traces means big current. Good chance it is a power line.*

**

* 1. Chip Identification

Gather information about each chip on the board for use later. Build a “Bill-of-Materials” (BOM).

* What are each of the chips and what do they do?

*Identify memory, processors, and interface controllers. The more you know the better.*

* 1. Continuity Probing and Tracing

Reverse engineer the circuitry to better understand the board function and zero in on areas of interest. Draw schematics by hand as you develop an understanding during this phase.

* How are inputs and outputs connected to chips?

*Identify passive circuitry, draw it out, reason about it.*

* What chips are connected?

*Buses between memory and cpu or interfaces and cpu could be MitMd.*

* Where do depopulated pads and test-points connect to?
  + *This can help identify JTAG or serial interfaces and areas to be repopulated.*

1. Hardware Modification (~30 minutes)
   1. Solder the JTAG header
2. Active Probing (~90 minutes)

After a thorough assessment with the board unpowered, it is time to attempt to provide power to the board. In industry, this is called “smoke-testing” – give the board juice and watch for smoke. If you see smoke, you’ve failed. Otherwise, you’ve passed.

In automotive, most of the time you will find 12v DC is the proper power supply because a car battery is 12v DC. However, this is not always the case. Look for stickers on the case or power supply voltage in the documentation. If you are completely unsure, start at a lower voltage and gradually work up until the board appears to be functioning properly.

* 1. Multimeter Probing

Start with the power input and use the multimeter to check voltage levels at various points on the board. Focus on voltage regulators.

|  |
| --- |
| File:LM2676.jpgRelated image |
| Examples of regulators |

* What are the voltage domains?

*Know voltage domains to interface with board later without creating smoke.*

* Use to verify chip and connector pin-outs.

*Sanity check what you think you know about the components. Is GND and Vdd where you expect it to be?*

* Use to find UART transmission lines or CAN lines.

*Voltage on UART TX and CAN lines will fluctuate rapidly while data is transmitted.*

* 1. Oscilloscope Probing

Probing with an oscilloscope will give you a better picture of what is happening on the wire.

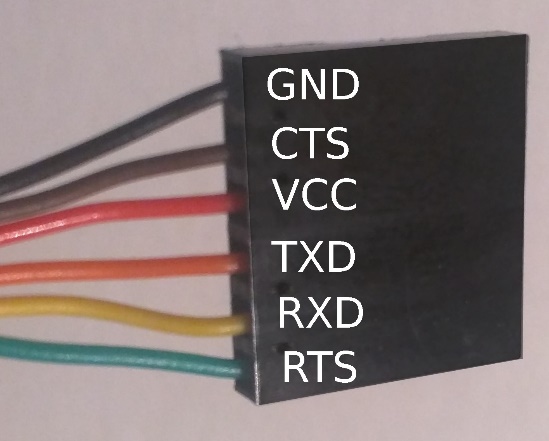
* Find digital buses
* Determine baud rates
* Decode signals
  1. UART Probing

UART stands for “Universal Asynchronous Receiver/Transmitter.” It is a widely used serial communication protocol that HW:RE can be used to:

* Obtain an interactive serial console with a target platform
* Read system and debugging logs
* Communicate with various interfaces

SparkFun has a great write-up here: <https://learn.sparkfun.com/tutorials/serial-communication>

We will be using a pretty ubiquitous “FTDI cable” as shown below.



We will also be using miniature clamp probes to attach to various places on the board to enable us to interface with the components.



* + 1. CLASS GOALS
       1. Find the system boot console

|  |  |
| --- | --- |
| Bootstrap Version |  |
| Memory Size |  |
| NVRAM Init Location |  |
| MFG Vers |  |

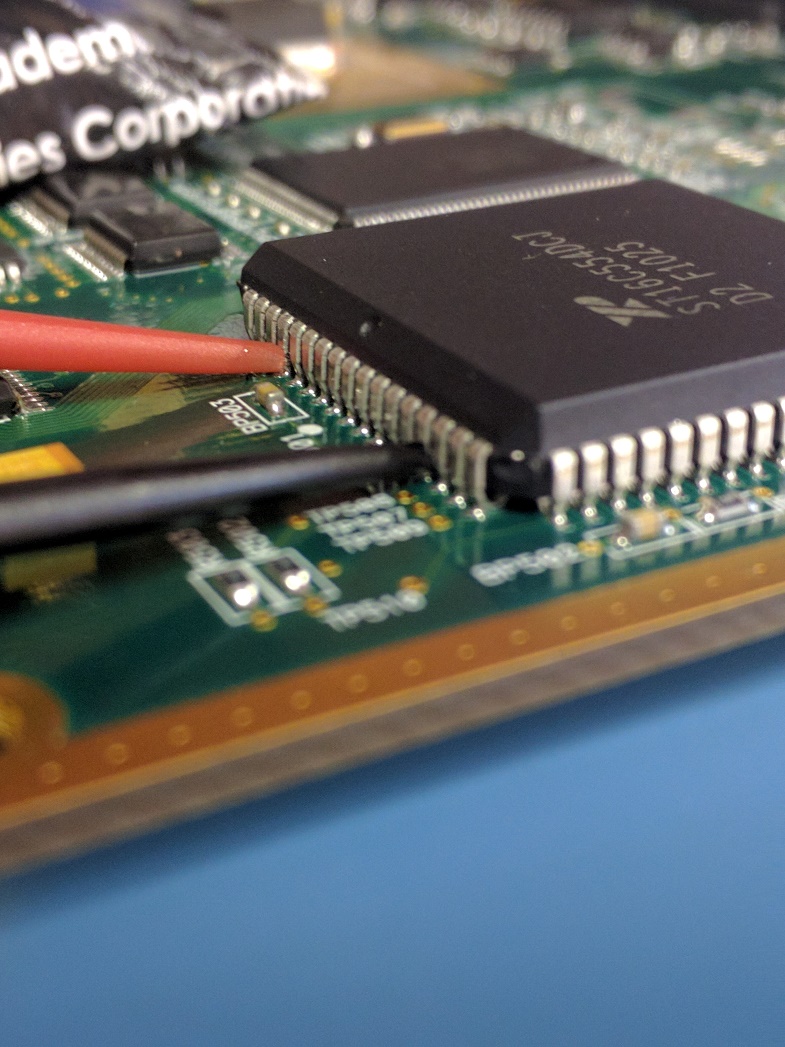
* + - 1. Find system debug logs

|  |  |
| --- | --- |
| PPPC baud\_rate |  |
| PPPC data\_bits |  |
| PPPC stop\_bits |  |
| CMxx default\_cc\_phone |  |

* + 1. Setup

1. Wire UART cable to board. Don’t wire VCC.

*HINT: You are supposed to figure this part out yourself. You should already know where the UART needs to hook up to from the previous stages. If not, check your work and come back ☺*



1. Install minicom:

sudo apt-get install minicom

1. Plug in the FTDI cable and determine which device file descriptor was created using dmesg:

dmesg | grep -i ttyusb

Mine appears as /dev/ttyUSB0, but yours might have a different number. Replace “ttyUSB0” in the following steps with the proper “ttyUSBx” on your machine.

1. Give permissions to /dev/ttyUSB0 and run minicom:

sudo chmod o+rw /dev/ttyUSB0

minicom -D /dev/ttyUSB0

1. Turn off hardware flow control to enable sending characters:

CTRL-A Z

O

“Serial port setup”

F

1. Useful commands

All commands from the main window begin with CTRL-A followed by a single key press.

|  |  |
| --- | --- |
| Help | CTRL-A Z |
| Exit | CTRL-X |
| Configure comm. Parameters  (Baudrate, data bits, stop bits, parity) | CTRL-P |
| Toggle local “echo” on/off | CTRL-E |
| Clear screen | CTRL-C |

Easter Eggs:

0:00:00.000 Trace Header type 0xDEADBEEF Apps/Agents/Trace/Trace.c, (793)

0:00:03.950 CMxx Restarted ...humbly awaiting instructions...

* 1. JTAGing

JTAG is a standard for “boundary scan” testing and in-circuit debugging, among other things. It allows a user to connect to the

* + 1. Setup Bus Pirate, OpenOCD, GDB and Eclipse

The microcontroller on the PeopleNet G3 is an ARM9, which is a relatively old part. In addition, the open-source ecosystem has mature support for the ARM architecture, including the Thumb instruction set architecture (ISA), which seems to be more common in embedded systems due to reduced code size. Therefore, we will use open-source tools to interact with the processor.

We’re going to use an open-source tool chain to attempt to attach to the NXP LH7A400 microcontroller/microprocessor

* + - 1. Upgrade Bus Pirate firmware

Although outdated, I followed this: <https://research.kudelskisecurity.com/2014/05/01/jtag-debugging-made-easy-with-bus-pirate-and-openocd/>

I just used the outdated firmware version BPv3-frimware-v6.1.hex [sic] found here: <https://github.com/BusPirate/Bus_Pirate/tree/master/package/BPv3-firmware/old-versions>

* + - 1. Install OpenOCD

OpenOCD is an open-source project for on-chip debugging.

sudo apt-get install openocd

* + - 1. Wire the Bus Pirate to the JTAG port

ARM has a pin out for the 20-pin JTAG connection here: <http://infocenter.arm.com/help/topic/com.arm.doc.dui0499d/BEHEIHCE.html>

* + - 1. Attempt to connect to board using OpenOCD attached to Bus Pirate

Use dmesg to find which device file was created for the bus pirate (on my machine it was /dev/ttyUSB0, but on yours it may be /dev/ttyUSB1 or another trailing number).

First, ensure that you aren’t connected to /dev/ttypUSB0 with any other program, such as screen or minicom. We’ll need to add permissions to the file /dev/ttyUSB0:

sudo chmod o+rw /dev/ttyUSB0

Create the simple configuration file from our walkthrough site and name it buspirate-simple.cfg.

Connect your bus pirate to your machine via USB.

Power on the board and wait two seconds. Now, try to attach:

openocd -f buspirate-simple.cfg –f lh7a400.cfg

If everything is successful you will see:

TODO

* + - * 1. Connect to ‘telnet localhost 4444’
      1. Install GDB (and GCC)

ower on the board and wait two seconds. Now, try to attach:

apt-get install gdb-arm-none-eabi

apt-get install gcc-arm-none-eabi

TODO

* + - 1. Diagnose the problem (free exploration)

Ideally, the students would have a little bit of time (15-20 min) to explore the problem of why the JTAG port isn’t working

* + - 1. Install and configure Eclipse
         1. Install Eclipse

Download the eclipse installer (TODO: insert link).

Ins

* + - * 1. Point Eclipse’s workspace to $/tools/eclipse
        2. Install the GDB Hardware Debugging plug-in

1. Go to “**Help**” -> “**Install New Software...**”
2. Choose the “**CDT**”... below “**—All Available Sites—**”
3. Search for “GDB”
4. Check the box to the left of “C/C++ GDB Hardware Debugging”
5. Click “**Next >**” twice
6. Select “**I accept the terms of the license agreement**”
7. Click the “**Finish**” box
8. Restart Eclipse when prompted
   * + 1. Use eclipse a little bit

We will need to change the path null.elf (because I couldn’t figure out how to do relative paths).

TODO

1. Tools Overview

**This might become an appendix**

* 1. Basic Hardware Tools
     1. Multimeter

Plan to spend at least $50 on a multimeter. Look for capacitance, resistance, continuity, current, and voltage measurements with high accuracy and resolution.

* + 1. Oscilloscope
    2. Logic Analyzer
    3. UART-to-USB Adapter (a.k.a. FTDI Cable)
    4. Bus Pirate
    5. In-Circuit Debugger (ICD)
    6. Soldering Iron

You’ll also need solder, flux, and a desoldering wick or pump.

Flux prevents beading of solder and helps it flow to the circuit board.

* 1. Search Engine, Encyclopedia
  2. Advanced Hardware Tools
     1. Heat Gun

Used for soldering and desoldering large areas; especially useful for

* + 1. Clamp Meter

Allows for measurement of AC current and frequency and DC current without physically contacting wire. Uses hall effect

* + 1. JTAGulator
    2. Chip Whisperer
    3. Flyswatter
    4. ROM Reader
    5. X-Ray Imager

Cost: Expensive

<https://uvicrec.blogspot.com/2015/08/xy-ray-x-ray-scanner.html>

* + 1. Scanning Electron Microscope (SEM) and Transmission Electron Microscope (TEM)

Cost: Expensive

Can be used to take high precision imagery on surface as well as internally.

* + 1. Lazers

Cost: Expensive

Can be used to decapsulate chips or for fault injection attacks.

* 1. Glossary

|  |  |
| --- | --- |
| EABI | Embedded-application binary interface (EABI) |
| ICD/OCD/ICE | The terms in-circuit debugging (ICD), on-chip debugging (OCD) and in-circuit emulation (ICE) are used interchangeably. |
| ISA | Instruction set architecture |
| JTAG |  |
| Microcontroller |  |
| Microprocessor |  |
| PCB | Printed Circuit Board |
|  |  |

1. References

R. Torrance and D. James, “The State-of-the-Art in IC Reverse Engineering,” <https://www.iacr.org/archive/ches2009/57470361/57470361.pdf>

P. Laackmann and M. Janke, “Uncaging Microchips,” https://www.youtube.com/watch?v=pIpxawdUb4I