

# **A HANDS-ON SECURITY COURSE**

by

**ANDREW HENNESSY**

Submitted in partial fulfillment of the requirements

for the degree of Master of Science

Department of Electrical Engineering and Computer Science

CASE WESTERN RESERVE UNIVERSITY

May 2017

**CASE WESTERN RESERVE UNIVERSITY**  
**SCHOOL OF GRADUATE STUDIES**

We hereby approve the thesis of

**Andrew Hennessy**

candidate for the degree of **Master of Science\***.

Committee Chair

**Dr. Kenneth Loparo**

Committee Member

**Dr. Francis Merat**

Committee Member

**Dr. Gregory Lee**

Date of Defense

**April 6, 2017**

\*We also certify that written approval has been obtained  
for any proprietary material contained therein.

# Contents

List of Tables	v
List of Figures	vi
Acknowledgments	viii
Abstract	x
<b>1 Introduction</b>	<b>1</b>
1.1 Other Courses at Different Universities . . . . .	2
1.2 Organization of the Thesis . . . . .	3
<b>2 Laboratory Environment</b>	<b>4</b>
2.1 Equipment and Tools Needed . . . . .	5
2.2 Experimental Board . . . . .	6
2.2.1 Purpose-Designed Board . . . . .	6
2.2.2 Off-the-shelf Boards . . . . .	18
<b>3 Syllabus</b>	<b>20</b>
3.1 Prerequisites . . . . .	20
3.2 Credit Hours . . . . .	20
3.3 Course Description . . . . .	20
3.4 Key Concepts . . . . .	21

---

3.5	Course Requirements . . . . .	21
3.6	Student Expectations . . . . .	21
3.7	Lecture Schedule . . . . .	22
3.8	Grading . . . . .	22
3.9	Student Conduct and Academic Integrity . . . . .	23
3.10	Expected Outcomes . . . . .	23
<b>4</b>	<b>Lab #1: Buffer Overflows</b>	<b>24</b>
4.1	Objectives . . . . .	24
4.2	Assignment . . . . .	24
4.2.1	Introduction . . . . .	24
4.2.2	Instructions . . . . .	25
4.3	Assignment Files . . . . .	26
4.4	Solutions . . . . .	29
<b>5</b>	<b>Lab #2: Diffie-Hellman Key Exchange</b>	<b>32</b>
5.1	Objectives . . . . .	32
5.2	Assignment . . . . .	32
5.2.1	Extra Credit Opportunity #1 . . . . .	33
5.2.2	Extra Credit Opportunity #2 . . . . .	33
<b>6</b>	<b>Lab #3: AES Encryption</b>	<b>34</b>
6.1	Objectives . . . . .	34
6.2	Assignment . . . . .	34
6.2.1	Introduction . . . . .	34
6.2.2	Instructions . . . . .	35
6.3	Assignment Files . . . . .	36
6.4	Solution . . . . .	38
6.4.1	Caesar Cipher Exercise . . . . .	38

---

6.4.2	AES Exercise . . . . .	38
<b>7</b>	<b>Lab #4: Bus Snooping</b>	<b>39</b>
7.1	Objectives . . . . .	39
7.2	Assignment . . . . .	39
7.2.1	Introduction . . . . .	39
7.2.2	Instructions . . . . .	40
7.2.3	References and Further Reading . . . . .	42
7.3	Solution . . . . .	42
<b>8</b>	<b>Lab #5: Reverse Engineering</b>	<b>45</b>
8.1	Objectives . . . . .	45
8.2	Assignment . . . . .	45
8.2.1	Introduction . . . . .	45
8.2.2	Instructions . . . . .	46
8.2.3	Deliverables . . . . .	47
8.3	Solution . . . . .	48
<b>9</b>	<b>Lab #6: Physically Unclonable Functions</b>	<b>49</b>
9.1	Objectives . . . . .	49
9.2	Assignment . . . . .	49
9.2.1	Introduction . . . . .	49
9.2.2	Instructions . . . . .	50
9.2.3	References . . . . .	51
<b>10</b>	<b>Final Project &amp; Paper</b>	<b>52</b>
10.1	Objectives . . . . .	52
10.2	Final Paper . . . . .	52
10.3	Final Project . . . . .	53

10.3.1	Introduction . . . . .	53
10.3.2	Part 0: Project Proposal . . . . .	53
10.3.3	Part 1: Building the Project . . . . .	54
10.3.4	Part 2: Attacking the Project . . . . .	54
10.3.5	Part 3: Defending the Project . . . . .	54
<b>11</b>	<b>Results &amp; Conclusion</b>	<b>55</b>
11.1	Results . . . . .	55
11.1.1	Fall 2014 . . . . .	55
11.1.2	Fall 2015 . . . . .	58
11.2	Conclusion . . . . .	61
<b>A</b>	<b>Proposed SAGES Course</b>	<b>62</b>
<b>B</b>	<b>Layout Files</b>	<b>66</b>
	<b>Bibliography</b>	<b>75</b>

# List of Tables

2.1	The Bill of Materials for the purpose-designed board. . . . .	8
2.2	Pin Mapping between the Custom Board, the Arduino Leonardo and the native pin names. . . . .	15
2.3	Pin Mapping between the Custom Board and the native pin names for the CPLD and microcontroller. . . . .	16
7.1	The contents of an $I^2C$ transaction between the microcontroller and the accelerometer. . . . .	43
8.1	Example Bill of Materials for the assignment. . . . .	47
11.1	The sequence of assignments for the Fall 2014 version. . . . .	56
11.2	The sequence of assignments for the Fall 2015 version. . . . .	58
A.1	Tentative, proposed, schedule for the SAGES course. . . . .	63
B.1	The Pick-n-Place file for automated component placement. . . . .	74

# List of Figures

2.1	The block diagram of the purpose-designed board. . . . .	7
2.2	The PCB layout of the purpose-designed board in Altium Designer. .	7
2.3	The Arduino IDE when it is first opened. . . . .	14
2.4	The data the default application sends to the computer. . . . .	16
4.1	The expected output of “simple_buffer_overflow.c” . . . . .	30
4.2	The expected output of “pointer_buffer_overflow.c” . . . . .	31
6.1	The output of “aes_example.c” with three different length arguments.	38
7.1	Block diagram of the demo Printed Circuit Board (PCB). . . . .	40
7.2	The screen of an Agilent MSO 2002A configured to decode I2C data and to trigger on a I2C start bit. It should be noted that the measured frequency of Channel 1 is incorrect. The correct value is 400.0kHz. . .	43
8.1	Block diagram of the demo Printed Circuit Board (PCB). . . . .	46
B.1	The schematic of the purpose-built board. . . . .	67
B.2	The top silk screen graphic. . . . .	68
B.3	The top soldermask graphic. . . . .	69
B.4	The top copper graphic. . . . .	70
B.5	The bottom copper graphic. . . . .	71
B.6	The bottom soldermask graphic. . . . .	72



B.7 The drill locations. . . . .	73
----------------------------------	----

# Acknowledgments

The first person that requires acknowledgment is Dr. Kenneth Loparo, without whom this thesis would likely not exist. He was always willing to drop what he was doing and offer an open chair to listen to and assist with my issue *du jour* as well as offering a gentle nudge every once in a while to keep me on track. He also had enough faith in me to allow me to teach an untested course to twenty-five undergraduate students over two semesters. Secondly, without Dr. Francis Merat's seemingly unending knowledge of Case Western's bureaucracy and history I would have been stuck multiple times. Thirdly, Dr. Gregory Lee, whose guidance and experience enabled me to make my course better than it otherwise would have been.

Additionally, I would like to thank one of my cohorts over at Cleveland State University, Chirayu Shah, for providing the basis of two of the software-based assignments that I built upon.

I would also like to thank the innumerable people who let me bounce ideas off of their shoulders during the brainstorming sessions for this course. Without their open ears this course would have turned out very differently.

This material is based upon work supported by the National Science Foundation under Grants No. 1603480 and 1520306. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Finally, without my family's support and encouragement none of this would have

ever been possible.

# A Hands-On Security Course

Abstract

by

ANDREW HENNESSY

In today's world almost every transaction, whether it's monetary or the exchange of information, passes through a computer system at some point in its journey. These transactions are secured using numerous forms of encryption to both protect the information as well as ensuring the integrity of the parties involved.

These security methods are currently taught in classrooms with chalkboards full of equations without a thought to how the methods are practically implemented, leaving students confused as to how these equations and concepts relate to the physical or virtual worlds.

This thesis provides the basis of a laboratory course that enables students to easily learn about the modern implementations of security in a controlled environment using common tools such as a digital multimeter and an oscilloscope, providing both a theoretical and practical setting for student enrichment.

This course was piloted at Case Western Reserve and the student response was overwhelmingly positive.

# Chapter 1

## Introduction

Almost everything we do on a computer has to be secured in one fashion or another. Historically, communications between two parties were secured by a number of different methods, ranging from wax seals to padlocks and even to primitive cyphers. The direct analogy to physical security in the digital world is encryption. However, like wax seals — which can be opened without detection — and locks — which can be pried open — encryption can be defeated in a number of different ways. This thesis explores how encryption can be broken by attacking, “hacking,” with two different approaches: attacking the implementation of the encryption in software and by attacking the hardware that the encryption algorithms run on.

As it stands now, encryption technologies and methods are rarely taught at the undergraduate level, with often only one or two courses offered at the graduate level. These courses are typically through the Mathematics Department instead of through an engineering department. These courses are usually taught at the theoretical level, ignoring and purposely leaving out implementation details. This leads to an oft-repeated adage that states, “one should never implement their own encryption system, they should let the professionals handle it.” This statement, however, implicitly instructs users to treat an encryption algorithm as a black box, a statement which

runs orthogonal to that of an educational institutions's philosophy.

## 1.1 Other Courses at Different Universities

While researching this class and creating the required coursework for students to preform, a small handful of other courses at different universities were discovered with the same goal as the course described in this thesis. It is difficult to find full details on these courses due to the increasing trend of courses being locked behind a password-protected Learning Management System, such as Blackboard, Moodle and Canvas.

One such course that was found, and not locked behind such a system, was CSCI: 4974 and 6964: Hardware Reverse Engineering taught during the Fall 2014 semester at the Rensselaer Polytechnic Institute (RPI) by Dr. Bülent Yener and his TA, Andrew Zonenberg[11].

Their course focused entirely on attacking the hardware that an encryption algorithm runs on and they were able to go more in-depth with equipment and methods such as decapping an Integrated Circuit (IC), as well as invasive attacks on hardware with tools such as a Focused Ion Beam (FIB) setup to modify IC's and Scanning Electron Microscope (SEM) to reverse engineer the schematics of an IC.

As it's title suggests, this course focuses entirely on the hardware of a system and how it can be attacked, with little regard to potential software vulnerabilities. In addition, while Case Western Reserve has a facility that owns a SEM and FIB machine, budgetary and course-sequence concerns precluded the use of these machines in the way that the RPI course used them. One of the design criteria for the course described in this thesis was that it should be available to the highest number of students possible and not have external prerequisite dependencies. The consequence of this decision is that the advanced labs that the RPI course was able to include,

such as the aforementioned IC reverse engineering lab, could not be included in this course.

## 1.2 Organization of the Thesis

This thesis is divided into four broad parts. The first part contains the introductory material — this introduction, Chapter 1 — as well as Chapter 2, which discusses the equipment that is needed to do the experiments documented in this thesis.

The second part, Chapters 3–10 discuss in detail each of the six experiments that compose the course on Integrated Security. It starts with Chapter 3, which provides a sample syllabus for the class. Then, the chapters for the experiments are typically further subdivided into two sections. The first section is a short description of what the students are expected to have learned following the conclusion of the experiment. The other section in each chapter is the assignment that is given to the students to complete, along with any necessary support files. Solutions to questions that are asked in the assignment are provided, however several assignments ask students to write a program to accomplish the task. Due to the large variety of possible answers, solutions to programming exercises are not provided.

The next part, Chapter 11 discusses the feedback received by students when this course was offered twice, in the Fall of 2014 as well as the Fall of 2015 as well as changes that were made as a consequence of that feedback.

The last part of this thesis is comprised of a two appendices. The first appendix describes an alternate version of the course described in this thesis re-framed as a freshman English class. The second appendix contains all of the necessary technical documentation that is needed to construct more copies of the purpose built board that is described in Chapter 2.

## Chapter 2

# Laboratory Environment

One aspect that sets this curriculum apart from others in the field is that all of the experiments and laboratory exercises are done on an embedded device, as opposed to using either a desktop computer or what is unfortunately becoming the norm, having no hands-on labs at all. The advantages of this approach are numerous, students are able to interact with hardware in situations that they might not otherwise be able to as well as seeing how changes in the firmware affect the fundamental operation of the hardware.

The drawback of this approach is that a certain amount of initial capital investment is needed to teach this course. This course was designed around using a general purpose circuits laboratory without the need for specialized equipment outside of what is needed for an introductory sophomore-level circuits course. For example, at Case Western Reserve University sophomore engineering students take a course titled **ENGR 210: Introduction to Circuits and Instrumentation** to satisfy part of their engineering breadth requirement. This course has a mandatory weekly laboratory session in a purpose-build teaching laboratory, the Sears Undergraduate Circuits Lab, that has the following equipment:

- HP 33120A Function Generator



- HeTest 3005F Dual Rail Power Supply
- HP 34401A or Keithley 2000 Digital Multi Meter
- Agilent MSO 2002A Digital 2 Channel Mixed-Signal Oscilloscope

This is typical of the level of equipment that is available in most engineering schools.

## 2.1 Equipment and Tools Needed

For the described security-centric class this array of expensive test equipment is not necessarily needed. Since the security class is a predominantly digital class and not an analog design class, the function generator and the power supply are not needed. The digital multi meter is only needed for one experiment — the reverse engineering experiment — and a generic multimeter is all that is required.

Of the laboratory experiments elaborated on in later chapters only the experiments for snooping on a bus (Chapter 7) and reverse engineering (Chapter 8) require an oscilloscope. The rest of the experiments only require an embedded software development board, as discussed in the next section.

This course was designed to use a Keysight MSO 2002A oscilloscope, which is one step above the base model 2000 series oscilloscope. It has a bandwidth of 70 MHz, a maximum sample rate of 2 GS/s, a memory depth of one million points and two analog channels with eight digital channels, with a MSRP of \$2,059[15]. Additional to the oscilloscope itself the course is designed to take advantage of the segmented memory upgrade (\$307) as well as the *I<sup>2</sup>C* and *SPI* hardware decoders (\$500). For an institution gearing up to offer an electronics lab for the first time, outfitting a number of workstations with oscilloscopes and PC's might not be possible for some budgets.

A less expensive alternative that has been verified to work with all of the assign-

ments is the Saleae Logic 8. This is a more basic option than a full-blown oscilloscope, yet it has all of the features needed for this class. The bandwidth of the Saleae is 25 MHz with a maximum sample rate of 100 MS/s and a memory depth that is solely based on the attached computer amount of RAM, since the Logic 8 has no display or storage of its own and entirely relies on an attached computer. The Logic 8 has eight dual-purpose digital or analog inputs and the device's software automatically handles the decoding of  $I^2C$  and  $SPI$  bus signals, all for a MSRP of \$219, not that much more than a modern collegiate level textbook (and in some cases less)[21]. It would be reasonable to require students to purchase their own pocket-sized oscilloscope for this class instead of requiring an actual laboratory oscilloscope.

## 2.2 Experimental Board

The majority of the hardware experiments for this course are designed to run on a development board with a cross-compilation environment set up on a computer. For this class, a Printed Circuit Board (PCB) has been custom-designed for ease-of-use and completeness. Alternatively, an off-the-shelf board, such as an Arduino Uno R3, can be used although not all experiments, as they currently exist, will be able to be completed on an off-the-shelf board.

### 2.2.1 Purpose-Designed Board

Once the experiments were sketched out and preliminary designs were finished it became apparent that no currently existing off-the-shelf development board would be suited to the tasks necessary for this course. A new development board would have to be designed and manufactured.

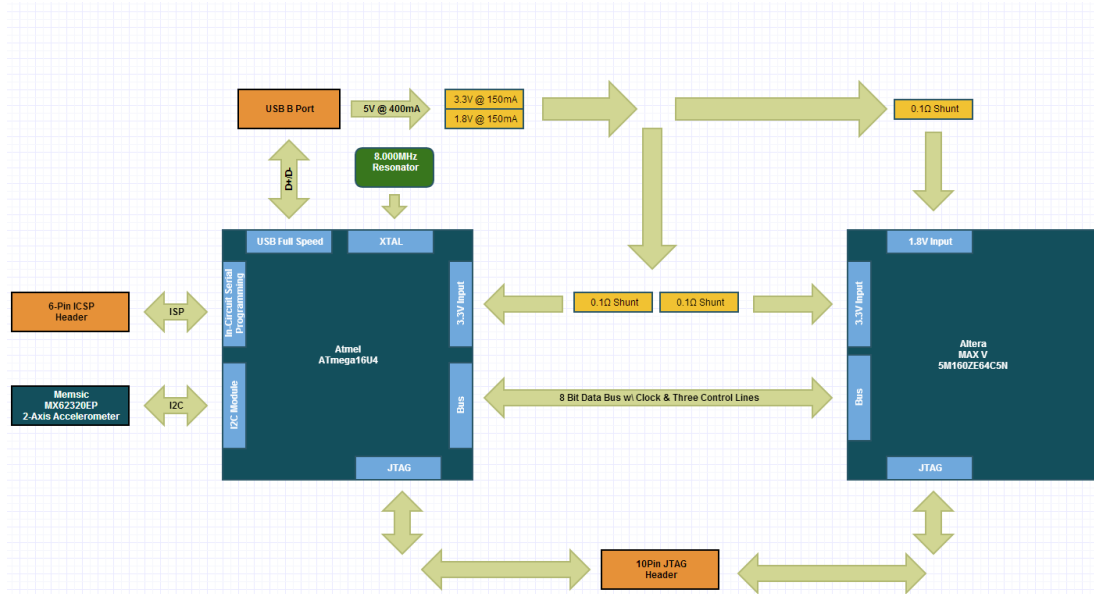


Figure 2.1: The block diagram of the purpose-designed board.

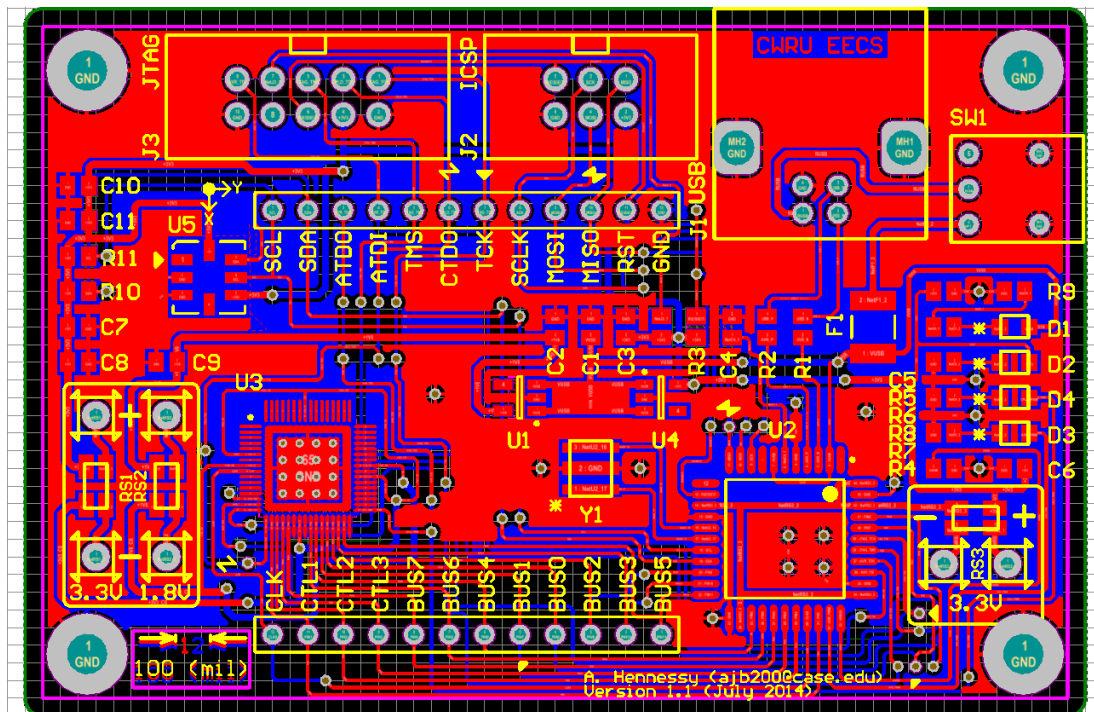


Figure 2.2: The PCB layout of the purpose-designed board in Altium Designer.

RefDes	Manufacturer	Model	Description
C1, C2, C3, C4	Kemet	C0805C105Z4VACTU	Ceramic 0805 1uF Capacitor, Y5V, 16V
C5, C6, C7, C8, C9, C10, C11	Kemet	C0805C104Z4VACTU	Ceramic 0805 0.11uF Capacitor, Y5V, 16V
D1, D2, D3, D4	OSRAM	LG R971-KN-1	0805 Green LED
F1	Bel Fuse	0ZCG0050AF2	500mA PTC Fuse
J1	Kycon	KUSBXHT-BS1N-O-HRF	USB Type B Plug
J2	3M	30306-6002HB	2x3 0.1" Header (ISCP)
J3	3M	30310-6002HB	2x5 0.1" Header (JTAG)
P1, P2	TE Connectivity	9-146285-0-14	1x14 0.1" Header (Debug)
R1, R2	Vishay Dale	CRCW080522R0FKEA	22 Ohm Resistor, 1/8W, 1%, 0805
R3, R4	Vishay Dale	CRCW080510K0FKEA	10 kOhm Resistor, 1/8W, 1%, 0805
R5, R6, R7, R8, R9	Vishay Dale	CRCW0805220RFKEA	220 Ohm Resistor, 1/8W, 1%, 0805
R10, R11	Vishay Dale	CRCW08054K70FKEA	4.7 Ohm Resistor, 1/8W, 1%, 0805
RS1, RS2, RS3	Ohmite	LVK12R100DER	0.1 Ohm Resistor, 1/2W, 0.5%, 1206
SW1	NKK	GW12RHH	SPDT Switch
U1	Texas Instruments	LP3990MF-1.8/NOPB	1.8V LDO Voltage Regulator
U2	Atmel	ATMEGA16U4-AUR	8-Bit Microcontroller
U3	Altera	5M160ZE64C5N	160 Element CPLD
U4	Texas Instruments	LP3990MF-3.3/NOPB	3.3V LDO Voltage Regulator
U5	Memsic	MXC62320EP	2 Axis Accelerometer
Y1	Abrakon	AWSCR-16.00CV-T	16 MHz Resonator

Table 2.1: The Bill of Materials for the purpose-designed board.

## Design of the Board

A block diagram of this board can be seen in Fig. 2.1. The overall architecture of the board includes a master microcontroller (MCU) which receives data from a sensor (an accelerometer in this case) and can perform off-chip Digital Signal Processing (DSP) on a Complex Programmable Logic Device (CPLD) that is attached to the MCU. This design was chosen to be as flexible as possible in order to emulate emerging Internet of Things (IoT) devices. The only element missing from this board is a means of wireless communication, although this can be added later by using the debugging extension headers.

A completed version of the purpose-designed board can be seen in Fig. 2.2. In the figure the layer colors are as follows: Red is Top Copper, Blue is bottom Copper, Yellow is Top Silkscreen and Green/Grey are Holes. The Bill of Materials for the board can be seen in Table 2.1.

The Atmel ATMEGA16U4 was chosen for this board. It is eight bit microcontroller that uses a modified Harvard architecture RISC core. That is to say user and program data are treated as the same by the core, which carries the designation ‘AVR’ by its manufacturer. The ATMEGA16U4 has 16 KB of flash available to store code as well as 1.25 KB SRAM to store program state during execution. The chip also has 512 bytes of Electrically Erasable Programmable Read-Only Memory, which while not used in any of the experiments for this course, is able to store data across a power cycle. The ATMEGA32U4 is a drop in replacement for the ATMEGA16U4 which doubles the amount of flash to 32 KB, the SRAM to 2.5 KB and the EEPROM to 1 KB. The ATMEGA16U4 was chosen for this board over the ATMEGA32U4 primarily due to availability restrictions on the ATMEGA32U4 when the PCB was sent out for fabrication and assembly, but also to save approximately 40% in parts cost[4].

The AVR Core is well supported by several different compilers and, as we will discuss later in this section, is the foundation for the popular Arduino software suite

and development board. In fact, this board can easily emulate the popular Arduino Leonardo development board, which makes programming it by students even easier.

One standout feature of the Atmel ATMEGA16U4 is that it has integrated support for the Universal Serial Bus (USB) standard. The microcontroller supports creating a number of virtual USB endpoints, one of which can be used to emulate a standard serial port. If the ATMEGA16U4 did not support this feature, communication with a host computer would have to go through a more complex translation between the microcontroller’s serial port and an external Integrated Circuit that supports translating between both standards.

The other half of this board, so to speak, is the CPLD. The CPLD that was chosen for this board was the Altera MAX V 5M160ZE64C5N. The 5M160ZE64C5N has a hundred and sixty Logic Elements that can be configured to emulate any basic hardware device, including basic DSP functions[2].

As mentioned previously, many embedded devices collect data with an array of sensors then process that data using a separate DSP chip, all controlled by a microcontroller. A standalone DSP chip was not chosen for this board primarily due to cost and complexity. The Altera 5M160ZE64C5N is \$3.51[7] in quantities and a simple DSP chip starts at \$30[10]. Furthermore, a standalone DSP chip requires multiple power rails and potentially off-chip DRAM and assorted other interfaces. Adding support for those would have exponentially increased the cost to manufacture and assemble this simple board. Furthermore, DSP software stacks are heinously complicated, not to mention expensive and proprietary, causing the core focus of this course — how hardware and software interact to learn about embedded security — to become shifted towards labs in “how to set up the DSP software”.

A similar rationale can be given for why a simpler CPLD was used instead of a more expensive, complex, and powerful Field Programmable Gate Array (FPGA). While a FPGA can be used for more extensive emulation of DSP tasks than a CPLD

can, it carries drawbacks, such as increased power and physical space.

After this board went into production and before this thesis was written, Altera announced the MAX 10 FPGA series which does away with the complicated power trees that FPGAs are known for and replaces it with a single power supply rail[1]. A revision to this board could be made that replaces the MAX V CPLD in favor of the newer MAX 10 FPGA, leaving students able to implement more realistic emulation DSP filters.

One downside to using Altera products is the required expensive, external, programmer, which Altera trademarks as the USB-Blaster series. The official USB-Blaster is \$300[9], while licensed clones are \$50[8]. A counterfeit programmer can be had for \$3 on E-Bay[12], although using a known counterfeit device carries ethical concerns, which would be amplified by the irony of doing so in a security class. It is possible to license the USB-Blaster technology from Altera for a cost of \$5,000 per design[20], although in our case it would require an additional microcontroller and FPGA that are more powerful than the ones on our board, further driving up the cost of the board for the students.

The board has programming connections for both Atmel's In-Circuit Serial Programming (ICSP) interface (Connector J2 in Fig. 2.2) as well as the standard Joint Test Action Group IEEE 1149.1 testing and debug interface (Connector J3 in Fig. 2.2). The Atmel ATMEGA16U4 can be programmed by both interfaces but the Altera 5M160ZE64C5N can only be programmed through the JTAG interface. The JTAG chain in this board is set so that the first device on the chain is the microcontroller than the second and final device on the chain is the CPLD.

To provide feedback to the user visually there are four green LEDs situated on the right-hand side of the board, as can be seen in Fig. 2.2. The top-most LED, D1, lights up any time that there is 5 Volts applied to the USB socket. The second LED down, D2, is illuminated whenever the power switch (SW1) is turned on and

the voltage regulators are providing 3.3V for the rest of the board. The next LED in the stack, D4, is connected to the CPLD to provide user-configurable notification. The final LED, D3, is connected to the microcontroller to serve a similar purpose. By default the two programmable LEDs are designed to light up whenever there is motion in the X- or Y-direction, as sensed by the accelerometer.

The Memsic MXC62320EP accelerometer was chosen for several different reasons. One of the biggest reasons is that the  $I^2C$  communication protocol to read the sensor's data is extremely simple — there are not dozens of registers that need to be precisely programmed to ensure proper operation. This chip simply outputs the current acceleration on two axes when asked[17]. This makes the assignments for bus snooping (Chapter 7) and reverse engineering (Chapter 8) simpler than they would otherwise be while still teaching students valuable lessons. Secondly the physical package of the device is big enough and the pin spacing is large enough not to require special (i.e. expensive) handling for automated assembly. An additional benefit is that the PCB can be finished in a cheaper HASL (Hot Air Solder Leveling) instead of more expensive process, such as ENIG (Electroless Nickel Immersion Gold), which would have been needed if an accelerometer in a QFN package was selected.

One note is that between the production of this board several years ago and the writing of this thesis is that the Memsic MXC62320EP has been discontinued in favor of a replacement that has a higher range and a lower noise floor[18]. Physically and electrically the newer version appears to be a drop in replacement so future revisions of the board could easily switch to the newer product.

The PCB was designed to be as small as possible to minimize manufacturing costs. The end result is a two-layer PCB that is 3" wide and 2" tall with components only on the top. Passive components, such as resistors and capacitors, were arranged in long lines all facing the same direction to make potential hand assembly faster, easier, and the potential for mistakes less. For such a simple design two layers is all that is



needed. A four layer board, adding in a layer for ground and a layer for the power supply, would be completely unnecessary. Additionally, having two layers helps make the reverse engineering assignment (Chapter 8) not a cruelly complicated endeavor. The most difficult to solder package on the board is the Altera 5M160ZE64C5N CPLD which has a pin pitch of 0.4 mm and an exposed pad on the bottom of the device. To solder this package either a reflow oven would need to be used or a hole could be placed to allow a soldering iron to access the exposed pad.

To keep the board from potentially shorting out on a table top four ANSI 4-40 holes were put on the extreme corners of the board to attach metal standoffs that elevate the board off of the surface it is resting on. The holes (and thus standoffs) are grounded to allow the board to be easily grounded, potentially alleviating any Electrostatic Discharge (ESD) issues.

The board was designed to allow easy access to all signals, for both the reverse engineering assignment (Chapter 8) and the bus snooping assignment (Chapter 7). To that end every signal between the microcontroller, the CPLD, the accelerometer as well as both of the programming interfaces were broken out to two standard 0.1" pitch headers for easy connection to an oscilloscope or logic analyzer.

The final feature of this board was put in for a homework assignment that ended up not being implemented in this course. Three 0.1  $\Omega$  sense resistors are located in series with the supply lines for the microcontroller and both of the CPLD's power rails. These resistors are located after any bulk bypass capacitance and as such can be used to perform dynamic power analysis on the microcontroller and CPLD.

## Software Development Experience

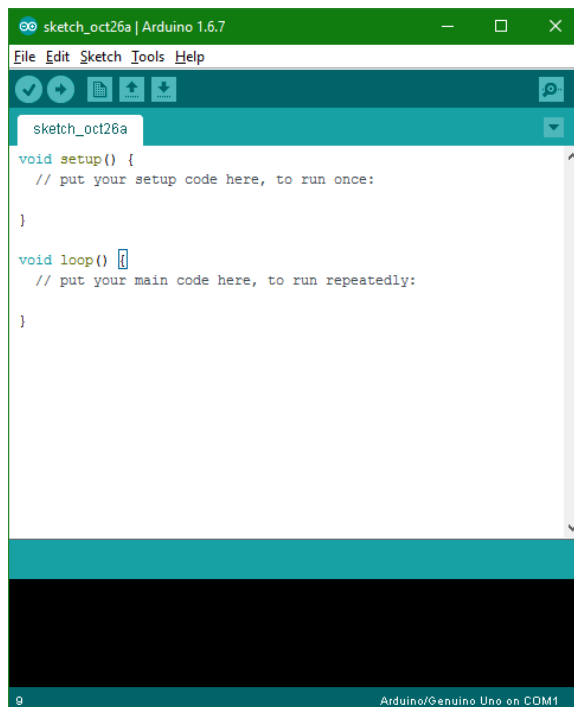


Figure 2.3: The Arduino IDE when it is first opened.

As mentioned previously the purpose-built board by default can easily emulate an Arduino Leonardo and is indistinguishable from other Arduinos to the Arduino IDE. When the board is plugged into a computer the USB-to-Serial Port drivers install automatically and transparently for recent versions of Windows and the board can then be programmed through the Arduino IDE to accomplish the end goal of the user.

One note is that the Arduino Leonardo does not use the pin numbering that the purpose-built board uses. Table 2.2 serves as a map between the two pin numbering schemes as well as the name of the pins that the ATMEGA16U4 uses internally.

There is an eleven bit bidirectional bus that connects the microcontroller to the CPLD along with a dedicated clock line. The bus was designed to allow for an eight bit data bus as well as three out-of-band general purpose signaling lines. The clock line is attached to one of the microcontroller's Pulse Width Modulation (PWM)

ATMEGA16U4	Arduino	Custom Board
PB4	Pin # 8	Pin # 3
PB5	Pin # 9	Pin # 2
PB6	Pin # 10	Pin # CLK
PC6	Pin # 11	Pin # 1
N/A	N/A	CPLD_LED
PC7	Pin # 13	AVR_LED
PD0	Pin # 3	$I^2C$ SCL
PD1	Pin # 2	$I^2C$ SDA
PD2	Pin # 0	Pin # 9
PD3	Pin # 1	Pin # 10
PD4	Pin # 4	Pin # 6
PD5	TXLED	Pin # 11
PD6	Pin # 12	Pin # 5
PD7	Pin # 6	Pin # 4
PF0	Pin # A5	Pin # 8
PF1	Pin # A4	Pin # 7

Table 2.2: Pin Mapping between the Custom Board, the Arduino Leonardo and the native pin names.

outputs, which allows for an easy adjustable clock frequency for the CPLD.

One of the aforementioned disadvantages is that the Altera 5M160ZE64C5N CPLD does not have a plug-and-play programming experience. A separate external programming adapter is required, although once attached the programmer has no issues programming the CPLD with the user's custom application like any other FPGA or CPLD programmed through Altera's Quartus II software. Table 2.3 serves as a guide for which pins on the microcontroller are connected to which pins on the CPLD.

### Default Application

For the laboratory experiments in this course, with special regard to the bus snooping (Chapter 7) assignment, a default application is run on the custom board.

This default application performs the following actions in a loop:

1. Query the accelerometer for the current X- and Y-axis acceleration.
2. Send the data to the CPLD to be filtered through a FIR filter.

ATMEGA16U4	Custom Board	CPLD Bank	CPLD Pin Number
PB4	Pin # 3	1	20
PB5	Pin # 2	1	19
PB6	Pin # CLK	1	9
PC6	Pin # 1	1	18
N/A	CPLD_LED	1	33
PC7	AVR_LED	N/A	N/A
PD0	$I^2C$ SCL	N/A	N/A
PD1	$I^2C$ SDA	N/A	N/A
PD2	Pin # 9	1	29
PD3	Pin # 10	1	30
PD4	Pin # 6	1	26
PD5	Pin # 11	1	31
PD6	Pin # 5	1	25
PD7	Pin # 4	1	27
PF0	Pin # 8	1	28
PF1	Pin # 7	1	27

Table 2.3: Pin Mapping between the Custom Board and the native pin names for the CPLD and microcontroller.

```

COM3 - PuTTY
X: (0x7EF --> 0x7E (126)) 0x90 (144); Y: (0x831 --> 0x83 (131)) 0x91 (145)
X: (0x7E5 --> 0x7E (126)) 0x85 (133); Y: (0x81E --> 0x81 (129)) 0x91 (145)
X: (0x8B5 --> 0x8B (139)) 0x80 (128); Y: (0x839 --> 0x83 (131)) 0x8D (141)
X: (0x906 --> 0x90 (144)) 0x7D (125); Y: (0x83E --> 0x83 (131)) 0x86 (134)
X: (0x970 --> 0x97 (151)) 0x84 (132); Y: (0x815 --> 0x81 (129)) 0x82 (130)
X: (0x7FD --> 0x7F (127)) 0x8C (140); Y: (0x895 --> 0x89 (137)) 0x84 (132)
X: (0x8EF --> 0x8E (142)) 0x8D (141); Y: (0x7C9 --> 0x7C (124)) 0x86 (134)
X: (0x930 --> 0x93 (147)) 0x8D (141); Y: (0x936 --> 0x93 (147)) 0x82 (130)
X: (0x8FB --> 0x8F (143)) 0x8F (143); Y: (0x89A --> 0x89 (137)) 0x88 (136)
X: (0x9C1 --> 0x9C (156)) 0x8B (139); Y: (0x88F --> 0x88 (136)) 0x88 (136)
X: (0x8B9 --> 0x8B (139)) 0x92 (146); Y: (0x78E --> 0x78 (120)) 0x87 (135)
X: (0x9CB --> 0x9C (156)) 0x94 (148); Y: (0x7E3 --> 0x7E (126)) 0x89 (137)
X: (0x904 --> 0x90 (144)) 0x90 (144); Y: (0xB0C --> 0xB0 (176)) 0x7F (127)
X: (0x8C0 --> 0x8C (140)) 0x94 (148); Y: (0x901 --> 0x90 (144)) 0x89 (137)
X: (0x8F3 --> 0x8F (143)) 0x91 (145); Y: (0x7B8 --> 0x7B (123)) 0x8E (142)
X: (0x9BB --> 0x9B (155)) 0x91 (145); Y: (0x5D1 --> 0x5D (093)) 0x8E (142)
X: (0xA25 --> 0xA2 (162)) 0x92 (146); Y: (0xBC5 --> 0xBC (188)) 0x89 (137)
X: (0x90F --> 0x90 (144)) 0x93 (147); Y: (0x89A --> 0x89 (137)) 0x8B (139)
X: (0xAB2 --> 0xAB (171)) 0x96 (150); Y: (0x844 --> 0x84 (132)) 0x85 (133)
X: (0x733 --> 0x73 (115)) 0x9C (156); Y: (0x7F1 --> 0x7F (127)) 0x8A (138)
X: (0x86F --> 0x86 (134)) 0x95 (149); Y: (0x9B9 --> 0x9B (155)) 0x91 (145)
X: (0x828 --> 0x82 (130)) 0x8D (141); Y: (0x77E --> 0x77 (119)) 0x89 (137)
X: (0x88A --> 0x88 (136)) 0x88 (136); Y: (0x88D --> 0x88 (136)) 0x86 (134)

```

Figure 2.4: The data the default application sends to the computer.

3. Read back the filtered data from the CPLD.
4. Send the filtered accelerometer data to the attached PC.

The data that the application sends to the computer can be seen in Fig. 2.4. This application enumerates as a generic USB Serial Port to the computer. A connection can be made to it with the following settings: 9600 baud, 8 Data Bits, No Parity and 1 Stop Bit.

The format that the data is sent is in the following format:

X: (0x88A --> 0x88 (136)) 0x88 (136);

Y: (0x88D --> 0x88 (136)) 0x86 (134)

Both the X- and Y-axis have three hexadecimal output values per iteration. The first value is the raw, unsigned twelve bit output from the accelerometer. The second value is the first value converted to an eight bit unsigned value, simply by throwing away the four least significant bits. In this example, for the X axis the raw value is 0x88A and the truncated value is 0x88. The final value for each axis is the result after the CPLD applied its algorithm. For ease of reading, the truncated and filtered values are repeated in decimal after the hexadecimal value is displayed.

The CPLD smooths the data coming from the accelerometer using a FIR filter, where  $y(n)$  is the output,  $x(n)$  is the current input and  $x(n - \{1, 2, 3\})$  are the three most recent inputs to the CPLD:

$$y(n) = \frac{x(n) + x(n-1) + x(n-2) + x(n-3)}{4} \quad (2.1)$$

This application ensures that there is always activity between the microcontroller, the CPLD, the accelerometer and the serial port.

## Revisions

If this board were to go through another revision while still keeping things basic (i.e. not upgrading to a MAX 10 FPGA), several things would be changed. As discussed previously, the accelerometer would be updated to the newer model.

The two rows of 0.1" headers allow easy access to every signal on the board except for the two power rails. This mistake would be rectified by changing the two twelve pin headers into two thirteen pin headers, with the 1.8V rail going to one header and the 3.3V rail going to the other header.

The ICSP header would be removed to save space and money as it is not strictly necessary. The JTAG header provides a programming interface for both ICs.

The 0.1  $\Omega$  sense resistors would be increased to a value of 0.5  $\Omega$  or even 1  $\Omega$ , as the current draw of the board was lower than expected, causing their voltage drop to be barely noticeable on an oscilloscope.

Finally, the ATMEGA16U4 would be replaced with the ATMEGA32U4 and the main resonator would be replaced with an 8 MHz model from the same line. Both items would be done to enhance compatibility and interchangeability with that of other Arduino models.

### 2.2.2 Off-the-shelf Boards

This course is designed to use the purpose-built board. However, with some care and modification of the assignments an off-the-shelf development board, such as the Arduino Uno R3, can be used.

The Arduino Uno R3 lacks the co-processor in the form of the CPLD and it lacks the integrated sensor, the accelerometer. A variety of sensors can be easily added in the form of ‘shields’, daughter boards that easily plug in to the expansion headers on the Uno. It would be possible to make a shield that contains a CPLD (or a FPGA).

Such a shield would be just as big as the purpose-built board, if not bulkier, with

insignificant cost-savings.

# Chapter 3

## Syllabus

### 3.1 Prerequisites

EECS 281: Digital Logic Design is recommended but not required

### 3.2 Credit Hours

This course is three credit hours.

### 3.3 Course Description

This course focuses on the hands-on learning of computer hardware security. The course will follow a distinctive hands-on teaching approach using a well-designed set of experiments as the learning tool. Students will be able to “hack” a system at different levels and analyze existing countermeasures.



## 3.4 Key Concepts

Introduction to comprehensive coverage on security issues — information, network, software, and hardware security. Understand information security through data encryption and decryption to protect data and systems. Learn buffer overflow attacks: stack overflow, heap overflow, and array indexing errors. Learn bus snooping attacks and protection schemes through different kinds of encryption. Be able to reverse engineer a closed system to figure out how the hardware components functions.

## 3.5 Course Requirements

There is no required textbook. The following books serve as a useful, but optional, references for concepts explored in this course:

- Erickson, Jon. “Hacking: The Art of Exploitation.” No Starch Press, San Francisco, 2008. ISBN: 978-1593271442.[14]
- Huang, Andrew. “Hacking the Xbox: An Introduction to Reverse Engineering.” No Starch Press, San Francisco, 2008. ISBN: 1-59327-029-1[3].
- Petzold, Charles. “Code: The Hidden Language of Computer Hardware and Software.” Microsoft Press, Redmond, 2000. ISBN: 0-7356-1131-9[5].

## 3.6 Student Expectations

The following criteria are essential for your success in this course.

Six group lab assignments will be given throughout the semester. Each assignment will have two weeks to complete. A final project will be assigned concurrent to the other assignments. There will be a research paper that is due on the last day of class. All lab assignment write-ups are to be completed on a computer; handwritten write-ups will not be accepted.

Late assignment write-ups will receive a penalty of ten percent per day unless a valid excuse was given to the instructor and accepted before the assignment was due. No late submissions will be accepted beyond one week after the original due date.

## 3.7 Lecture Schedule

This course meets twice a week for an hour and a half each time. The first day of every two week cycle will be spent in the classroom, with a lecture introducing the concept that the assignment revolves around. The second, third and fourth times the class meet during the cycle will be in the laboratory space for students to work on their assignments with the instructor and TA present to answer questions as needed.

As time permits and as needed, additional lectures will be given by either the instructor or an invited outside guest.

## 3.8 Grading

Your course grade is based upon your overall performance throughout the entire semester. The relative weights for your grade are as follows:

- Lab Assignments: 60%. (10% per lab.)
- Final Paper: 10%.
- Final Project: 30%. (10% per phase of the project)

This class uses the standard grading scale of:

- Less than a 60% is an F.
- Between 60% and 69.9% is a D.
- Between 70% and 79.9% is a C.
- Between 80% and 89.9% is a B.
- Above a 90% is an A.

## 3.9 Student Conduct and Academic Integrity

All forms of academic dishonesty including cheating, plagiarism, misrepresentation, and obstruction are violations of academic integrity standards. Cheating includes copying from another's work, falsifying problem solutions or laboratory reports, or using unauthorized sources, notes or computer programs. Plagiarism includes the presentation, without proper attribution, of another's words or ideas from printed or electronic sources. It is also plagiarism to submit, without the instructor's consent, an assignment in one class previously submitted in another. Misrepresentation includes forgery of official academic documents, the presentation of altered or falsified documents or testimony to a university office or official, taking an exam for another student, or lying about personal circumstances to postpone tests or assignments.

Obstruction occurs when a student engages in unreasonable conduct that interferes with another's ability to conduct scholarly activity. Destroying a student's computer file, stealing a student's notebook, and stealing a book on reserve in the library are examples of obstruction.

## 3.10 Expected Outcomes

Upon the successful completion of this course, a student will be able to:

- Understand the basic concepts of computer system security, including: network, information, software, and hardware security.
- Design new solutions to protect against existing attacks
- Learn how to hack into a system and then come up with a new threat model and the defense mechanisms against it.
- Analyze and validate computer systems for security, build new secure computer systems.

# Chapter 4

## Lab #1: Buffer Overflows

### 4.1 Objectives

- To become familiar with buffer overflows in the C programming language.
- To become familiar with identifying fixing buffer overflows.

### 4.2 Assignment

#### 4.2.1 Introduction

Buffer Overflows are a serious problem in modern systems. A typical buffer overflow is when a segment of memory that is assigned to one variable is erroneously overwritten by another variable's contents, corrupting the original variable. The consequences range from simply crashing the program with a segmentation fault error to potentially executing unwanted, arbitrary and malicious instructions.

In this assignment students will become familiar with finding potential buffer overflows and through executing injected code they will gain insight into exactly how dangerous buffer overflows can be.

### 4.2.2 Instructions

#### Part 1: Three Simple Examples.

Students should write code, in C, to demonstrate the concepts of three different kinds of buffer overflows: the heap overflow, the integer overflow, and the stack overflow.

#### Part 2: Two Specific Exploits

This section of the assignment will demonstrate how a buffer overflow can grant access to a secure system even if an incorrect password is given. This experiment is a perfect example of incorrect programming practices and how they can be exploited by an attacker. Anyone who knows that a program is susceptible to a buffer overflow can then exploit it and this assignment demonstrates how that is possible.

Please note that these programs have only been tested on a 32-Bit Linux installation and may or may not work on Mac OS X and Windows systems.

To compile the program on your system, first locate the archive you downloaded for the previous section and extract the `simple_buffer_overflow.c` file. Run the following command to compile the program:

```
gcc simple_buffer_overflow.c -o simple_buffer_overflow.
```

Run your compiled program with the following command:

```
./simple_buffer_overflow.
```

Try running the program under the following conditions and report the results?

1. What is the output when the correct password is entered? Note that the correct password is *cleveland*.
2. What is the output when an incorrect password that is between ten and twelve characters long?
3. What is the output when an incorrect password that is more than sixteen characters long?

In your writeup, please answer the following question: What is the error in the code that allows this to happen and how can it be fixed?

The next program will demonstrate how a buffer can overflow and how malicious code could be injected into a running program.

From the same archive used for the previous two examples locate the `bo_test.c` file as well as the `run.pl` file. Run the following command to compile the program:

```
gcc pointer_buffer_overflow.c -o pointer_buffer_overflow.
```

Run the newly compiled program with no options:

```
./pointer_buffer_overflow.
```

The program will output two addresses of functions on the call stack: a good function and a bad function. Copy the address of the bad function into the perl script using a text editor. Finally, run the perl code with the command `perl exploit.pl`

A buffer overflow attack should take place and the computer should attempt to shutdown. Take a screenshot of this and write a short conclusion about what happened and how it could be fixed.

## 4.3 Assignment Files

---

Listing 4.1: `simple_buffer_overflow.c`

---

```
1  /*
2   * A simple buffer overflow example that could
3   * allow access into a secure system.
4   * Written by Andrew Hennessy
5   * Based off of an example provided by Chirayu Shah at CSU
6   */
7
8  #include <stdio.h>
9  #include <string.h>
10
```

```
11 int main(void) {
12     char password[10];
13     int authorized = 0;
14
15     printf("Enter the password to gain access to the system:\r\n");
16     gets(password);
17
18     if(strcmp(password, "cleveland")) {
19         printf("You entered the wrong password :(\r\n");
20     } else {
21         printf("You entered the correct password!\r\n");
22         authorized = 1;
23     }
24
25     if(authorized){
26         printf ("Access Granted!\r\n");
27     }
28
29     return 0;
30 }
```

---

Listing 4.2: pointer\_buffer\_overflow.c

---

```
1 /*
2  * A more complicated example of a buffer overflow
3  * that could allow access into a secure system.
4  * Written by Andrew Hennessy
5  * Based off of an example provided by Chirayu Shah at CSU
6  */
7
8 #pragma check_stack(off)
9
10 #include <string.h>
```

```
11 #include <stdio.h>
12
13 void good_function(){
14     printf("good_function()\n");
15 }
16
17 void bad_function(){
18     printf("bad_function()\n");
19     system("shutdown -P now");
20 }
21
22 int main(int argc, char* argv[]){
23     char data[10];
24
25     printf("Address of good_function(): %p\n", good_function);
26     printf("Address of bad_function(): %p\n", bad_function);
27
28     void (*function_pointer)();
29     function_pointer = &good_function;
30
31     printf("SIZE : %d\n", sizeof(function_pointer));
32
33     // Try to overload the pointer here by coping data from argv[1]
34     strcpy(data, argv[1]);
35
36     printf("Address of the function pointer: %p\n", function_pointer);
37
38     function_pointer();
39
40     printf("The program stack looks like:\n%p\n%p\n%p\n%p\n%p\n%p\n% p\n\n");
41 }
```

---



Listing 4.3: expolit.pl

---

```
1 # A more complicated example of a buffer overflow
2 # that could allow access into a secure system.
3 # Written by Andrew Hennessy
4 # Based off of an example provided by Chirayu Shah at CSU
5
6 $argument = "AAAAAAAA" . "\xC1\x84\x04\x08";
7 $command = "./pointer_buffer_overflow "$argument;
8
9 system($command);
```

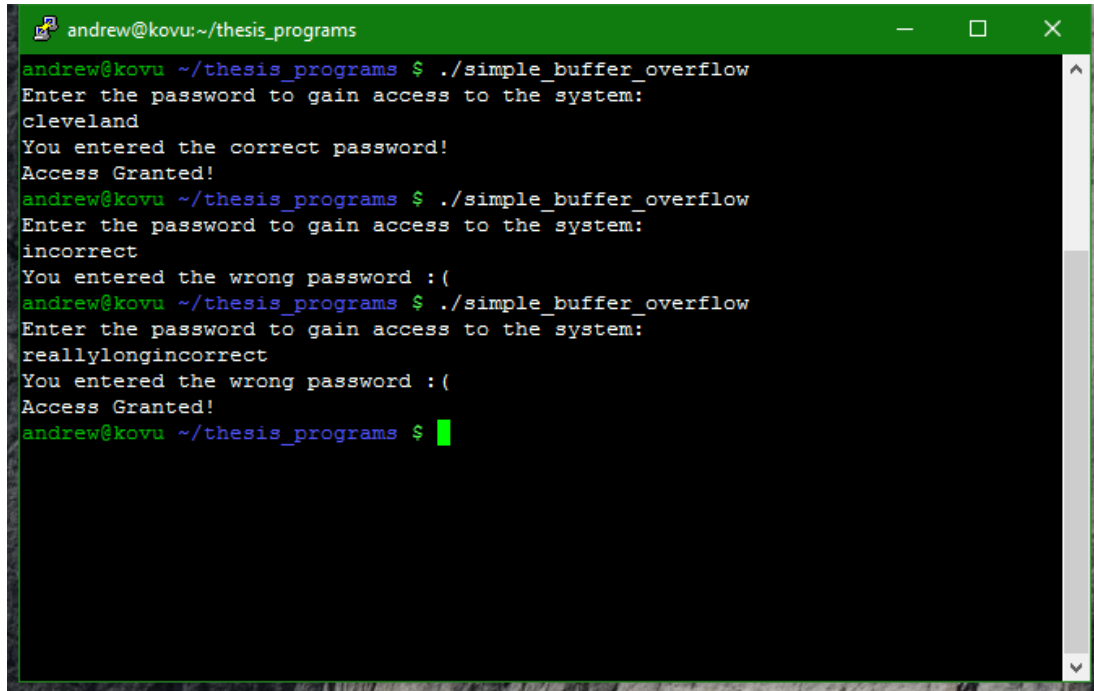
---

## 4.4 Solutions

1. The output when the correct key is entered is: “You entered the correct password! Access Granted!”
2. The output when a short, incorrect key is entered is: “You entered wrong password :(”
3. The output when a long, incorrect key is entered is: “You entered wrong password :( Access Granted!”

It is fairly obvious that a buffer overflow occurred. When too many characters were put into the `password` variable, it overflowed into the “authorized” variable, making it non-zero (e.g. `true`) and access was erroneously granted to the user. One possible way of fixing this issue is to use the function *fgets* instead of the deprecated function *gets*. Another way of fixing this issue is to explicitly check the `password` variable for the correct value of one instead of checking if it isn’t zero.

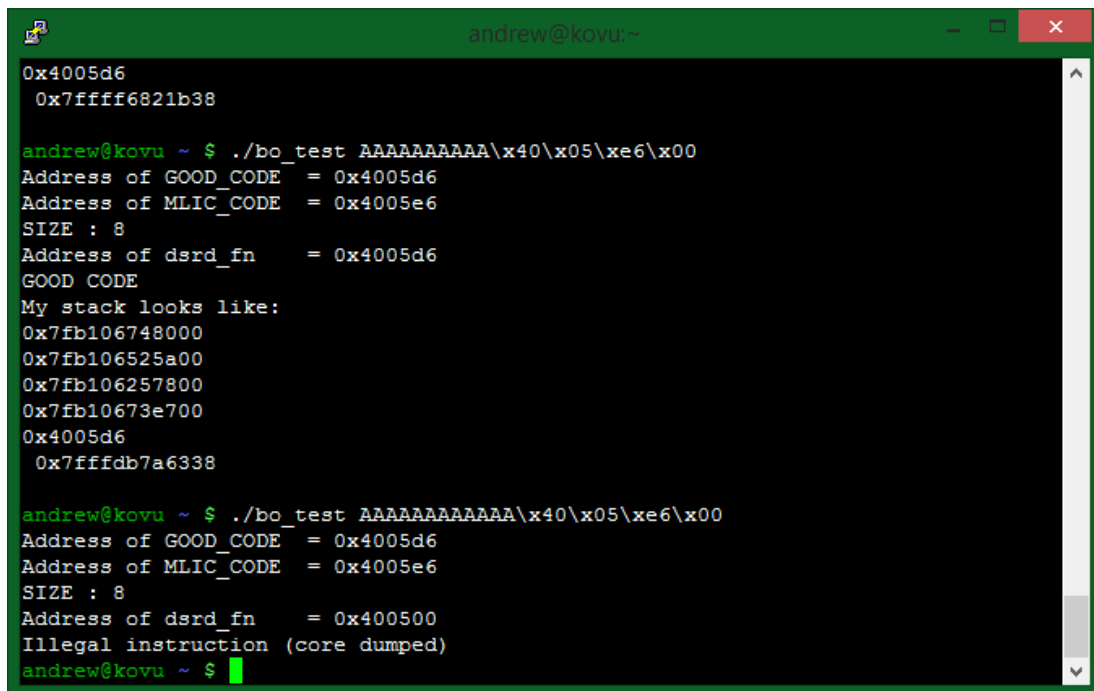
The perl script is written so that once the address of the malicious function is inputted into the script correctly, the script will try to overwrite a function pointer to change the execution path of the program from the correct function to the incorrect

A terminal window titled 'andrew@kovu:~/thesis\_programs' with standard window controls. The terminal shows the execution of a program named 'simple\_buffer\_overflow'. The user enters 'cleveland' as a password, which is correct, leading to 'Access Granted!'. The user then enters 'incorrect' and 'reallylongincorrect' as passwords, both of which are rejected with the message 'You entered the wrong password :('. The terminal ends with the prompt 'andrew@kovu ~/thesis\_programs \$' and a green cursor.

```
andrew@kovu ~/thesis_programs $ ./simple_buffer_overflow
Enter the password to gain access to the system:
cleveland
You entered the correct password!
Access Granted!
andrew@kovu ~/thesis_programs $ ./simple_buffer_overflow
Enter the password to gain access to the system:
incorrect
You entered the wrong password :(
andrew@kovu ~/thesis_programs $ ./simple_buffer_overflow
Enter the password to gain access to the system:
reallylongincorrect
You entered the wrong password :(
Access Granted!
andrew@kovu ~/thesis_programs $
```

Figure 4.1: The expected output of “simple\_buffer\_overflow.c”

function. This is possible because the function pointer is directly after the user input in the stack frame of the program. If the program was not run as the root user than the shutdown command failed to execute, however if the program was run as the root user then the computer would have successfully shut down.



```
andrew@kovu:~  
0x4005d6  
0x7ffff6821b38  
  
andrew@kovu ~ $ ./bo_test AAAAAAAAAA\x40\x05\xe6\x00  
Address of GOOD_CODE = 0x4005d6  
Address of MLIC_CODE = 0x4005e6  
SIZE : 8  
Address of dsrd_fn = 0x4005d6  
GOOD CODE  
My stack looks like:  
0x7fb106748000  
0x7fb106525a00  
0x7fb106257800  
0x7fb10673e700  
0x4005d6  
0x7ffffdb7a6338  
  
andrew@kovu ~ $ ./bo_test AAAAAAAAAA\x40\x05\xe6\x00  
Address of GOOD_CODE = 0x4005d6  
Address of MLIC_CODE = 0x4005e6  
SIZE : 8  
Address of dsrd_fn = 0x400500  
Illegal instruction (core dumped)  
andrew@kovu ~ $
```

Figure 4.2: The expected output of “pointer\_buffer\_overflow.c”

# Chapter 5

## Lab #2: Diffie-Hellman Key Exchange

### 5.1 Objectives

- To gain a basic understanding of public-key cryptography
- To become familiar with the process of key exchange and why it is needed.

### 5.2 Assignment

As discussed in class it is possible for two parties to communicate over an insecure channel and exchange enough information to generate a key known to only themselves. The purpose of this assignment is to implement the Diffie-Hellman Key Exchange protocol in a programming language of your choice. Please note that this is a group homework — your group will only turn in one assignment. Groups are not allowed to work together.

As discussed in class, this is the Diffie-Hellman Key Exchange Protocol, as described in [13]:

1. Alice and Bob agree on two numbers,  $p$ , the modulus (a prime number), and

$g$ , the base. These are public and should be communicated by one party to the other.

2. Alice generates a secret number,  $a$ , and sends Bob  $A$ , which is  $g^a \pmod{p}$ .
3. Bob generates a secret number,  $b$ , and sends Alice  $B$ , which is  $g^b \pmod{p}$ .
4. Alice computes the shared secret,  $s$ , by computing  $B^a \pmod{p}$ .
5. Bob computes the shared secret,  $s$ , by computing  $A^b \pmod{p}$ .

Your group will implement this protocol in your programming language of choice. You should allow one user to choose  $p$  and  $g$  as well as  $a$  and let the other user choose  $b$ . The program should output the shared secret  $s$ . Your program should use thirty-two bit unsigned integers.

### 5.2.1 Extra Credit Opportunity #1

One of the mathematical subtleties involved with this exchange that was not mentioned in class is that  $g$  should be a “primitive root modulo  $p$ ”. Implement a check in the beginning of the program that makes sure that  $g$  is in fact a primitive root modulo  $p$ .

### 5.2.2 Extra Credit Opportunity #2

Split your program into two parts that communicate over an insecure protocol. For example, make a client program and a server program that communicate over TCP/IP, Remote Procedure Calls, or use two microcontrollers that communicate over  $I^2C$ .

# Chapter 6

## Lab #3: AES Encryption

### 6.1 Objectives

- To gain a basic understanding of private-key cryptography.
- To become familiar with the basic Caesar Cipher.
- To become familiar with the AES standard.

### 6.2 Assignment

#### 6.2.1 Introduction

In this assignment students learn how to encrypt and decrypt data with two standard encryption algorithms. The focus of this assignment is to use cryptography to protect sensitive data with a key that is kept secret. This is in opposition to the previous week's assignment, where part of the key is kept public for anyone to use and see. In private key cryptography one way to make the data harder to decrypt without knowledge of the key is to make the bit length of the key longer.

This assignment deals with two different kinds of *block* ciphers, the Caesar Cipher and the Advanced Encryption Standard (AES). A block cipher takes in a certain

amount of data at a time, the “block”, and applies a mathematical algorithm involving the data and the key as inputs and outputs the ciphertext. Contrary to this, a *stream* cipher takes each bit of data, a bit at a time, and applies the algorithm bit-by-bit.

### 6.2.2 Instructions

Like the first week’s assignment, download the archive from Blackboard and extract the files in it to a known location.

#### Caesar Cipher Exercise

Before continuing on this assignment, you and your partner should research terms and concepts such as: the difference between block ciphers and stream ciphers, the difference between a cryptosystem, cryptography and encryption as well as the difference between asymmetric encryption and symmetric encryption.

You should write a short program in a language of your choice to demonstrate the operation of a Caesar Cipher. At the top of your submission please write a short paragraph on any improvements possible to the Caesar Cipher.

#### AES Exercise

Locate the `aes_example.c`, `TI_aes.c`, and `TI_aes.h` files from the archive and run the following command to compile the code:

```
gcc aes_example.c TI_aes.c -o aes_example
```

Run the compiled code with the following command:

```
./aes_example 16charargument_
```

Take a screen shot of the output. In your report answer the following question: What is the difference between the plain text input and the decrypted cipher text output?

Next, run the program again, except this time put in an argument that is less than sixteen characters long. Take a screenshot of the output. What changed?

After that, run the program a final time, except put in an argument that is at least twenty characters in length. Take a screenshot of the output. What changed?

Finally, can you modify the program to have a key length of either 192 bits or 256 bits? (Hint: the number of rounds of encryption will need to be changed)

## 6.3 Assignment Files

The files `TI_aes.c` and `TI_aes.h` are from an open-source implementation of the AES standard and can be found on GitHub[23].

---

Listing 6.1: `aes_example.c`

---

```
1  /*
2   * A simple program that runs text through the AES algorithm
3   * Written by Andrew Hennessy
4   * Based off of an example provided by Chirayu Shah at CSU
5   */
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <string.h>
9  #include "TI_aes.h"
10 #define LENGTH 16
11
12 void print_hex(unsigned char* message) {
13     unsigned short i = 0;
14
15     for (; i < LENGTH; i++) {
16         printf("0x%02X", message[i]);
17         if (i != (LENGTH - 1)) {
18             printf(", ");
```



```
19     }
20 }
21     printf("\r\n");
22 }
23
24 void main(int argc, char *argv[]) {
25     const unsigned char aes_key[] = {0xA1, 0xA2, 0xA3, 0xB4, 0xA5, 0xA6,
26     0xA7, 0xA8, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7, 0xB8};
27
28     unsigned char message[LENGTH];
29     strcpy(message, argv[1]);
30
31     char cipher_text[2 * LENGTH];
32
33     printf("ASCII message before encryption: '%s'\r\n", message);
34     printf("Hex message before encryption:      ");
35     print_hex(message);
36
37     aes_encrypt(message, aes_key);
38     memcpy(cipher_text, message, LENGTH);
39     cipher_text[17] = '\0';
40
41     printf("ASCII message after encryption:      '%s'\r\n", cipher_text);
42     printf("Hex message after encryption:          ");
43     print_hex(cipher_text);
44
45     aes_decrypt(cipher_text, aes_key);
46     cipher_text[17] = '\0';
47
48     printf("ASCII message after decryption:      '%s'\r\n", cipher_text);
49     printf("Hex message adter decryption:              ");
50     print_hex(cipher_text);
51 }
```

```

andrew@kovu: ~/thesis_programs
andrew@kovu: ~/thesis_programs $ ./aes_example 16charargument
ASCII message before encryption: '16charargument'
Hex message before encryption: 0x31, 0x36, 0x63, 0x68, 0x61, 0x72, 0x61, 0x72, 0x67, 0x75, 0x6D, 0x65, 0x6E, 0x74, 0x00, 0x00
ASCII message after encryption: 'G2:3\p'
Hex message after encryption: 0xE8, 0xCA, 0x1C, 0xEB, 0xB6, 0xB1, 0x47, 0x81, 0x5A, 0x3B, 0x33, 0x5F, 0x5C, 0x87, 0x01, 0x46
ASCII message after decryption: '16charargument'
Hex message after decryption: 0x31, 0x36, 0x63, 0x68, 0x61, 0x72, 0x61, 0x72, 0x67, 0x75, 0x6D, 0x65, 0x6E, 0x74, 0x00, 0x00
andrew@kovu: ~/thesis_programs $ ./aes_example short
ASCII message before encryption: 'short'
Hex message before encryption: 0x73, 0x68, 0x6F, 0x72, 0x74, 0x00, 0x00, 0x00, 0x90, 0x04, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00
ASCII message after encryption: 'ub/#-H*CA'
Hex message after encryption: 0x8B, 0x05, 0x75, 0xD0, 0xAC, 0x2F, 0x15, 0x23, 0xA3, 0xC4, 0x3D, 0x48, 0x2A, 0x43, 0x41, 0x90
ASCII message after decryption: 'short'
Hex message after decryption: 0x73, 0x68, 0x6F, 0x72, 0x74, 0x00, 0x00, 0x00, 0x90, 0x04, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00
andrew@kovu: ~/thesis_programs $ ./aes_example reallylongargument
ASCII message before encryption: 'reallylongargument'
Hex message before encryption: 0x72, 0x65, 0x61, 0x6C, 0x6C, 0x79, 0x6C, 0x6F, 0x6E, 0x67, 0x61, 0x72, 0x67, 0x75, 0x6D, 0x65
ASCII message after encryption: '$-UAb=3gD'
Hex message after encryption: 0x24, 0x04, 0x16, 0xA5, 0xCF, 0x2D, 0x55, 0x41, 0xD3, 0xB9, 0x62, 0x3D, 0x33, 0x67, 0x44, 0xA0
ASCII message after decryption: 'reallylongargume'
Hex message after decryption: 0x72, 0x65, 0x61, 0x6C, 0x6C, 0x79, 0x6C, 0x6F, 0x6E, 0x67, 0x61, 0x72, 0x67, 0x75, 0x6D, 0x65
andrew@kovu: ~/thesis_programs $

```

Figure 6.1: The output of “aes\_example.c” with three different length arguments.

## 6.4 Solution

### 6.4.1 Caesar Cipher Exercise

The Caesar Cipher is taught as one of the most basic and easily broken ciphers used. The cipher is inherently vulnerable to frequency analysis attacks and using any other cipher would be an improvement.

### 6.4.2 AES Exercise

The major difference between the plain text and the decrypted cipher text is that the cipher text is always only sixteen characters long. When the input is exactly sixteen characters long this isn’t noticeable. However, with a shorter input the output has random data inserted to pad the length up to sixteen characters and when the input is longer than sixteen characters the output is cutoff after the sixteen character mark.

The number of rounds required for a 192 bit key is 12 and for a 256 bit key is 14.

# Chapter 7

## Lab #4: Bus Snooping

### 7.1 Objectives

- To be able to approach an unknown system and derive information about its functionality
- To be able to decode an unknown  $I^2C$  bus's data.
- To gain proficiency with using oscilloscopes and logic analyzers.

### 7.2 Assignment

#### 7.2.1 Introduction

Many times in an embedded system all the effort to secure the system is focused on the software end of the spectrum. After all, the best attacks against any system are ones that attack the biggest vector: the software stack. That being said, one of the oldest adages relating to security is: “If an attacker has physical access to the system all bets are off.[22]” If there is no effort taken to secure the physical link between components in the system, such as the link between the main processor and its memory, then the data transferred between the two components is susceptible to

being eavesdropped upon, or even worse, modification. As we discussed in class, this is how the original Microsoft Xbox was hacked, the link between the Southbridge IC and an EEPROM on the Xbox was observed during bootup and subsequently modified to allow the hackers access to the protected region of the Xbox’s hard drive.

## 7.2.2 Instructions

In this experiment we will be looking at how the various components of the demo board communicate with each other and how the system as a whole communicates with an attached computer. To do this you will need the demo board given to you at the start of the class, a USB A to B cable and two sets of oscilloscope probes (and the corresponding oscilloscope!).

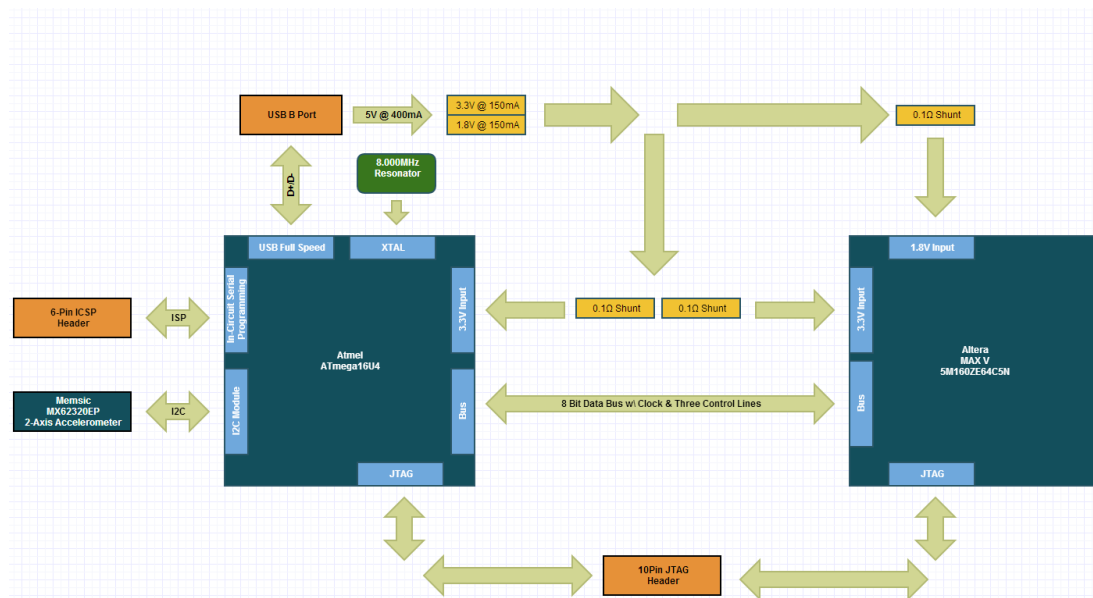


Figure 7.1: Block diagram of the demo Printed Circuit Board (PCB).

To start with, make sure that the USB-B end of the cable is unplugged from the PCB while the USB-A end of the cable is plugged into a power source (e.g. a computer or the oscilloscope’s USB port). Clip the oscilloscope probes to the pins labeled “SCL” and “SDA” on the top-left of the PCB while connecting both ground leads of the probes to the pin labeled “GND”.

Plug the USB-B end of the cable into the board and observe the pattern of traffic on the two lines. You will need to adjust the trigger mode and levels on the oscilloscope to get the best results. Tilt the PCB around and see how the traffic changes as well. Answer the following questions based upon your observations:

1. Which probe would you say is the attached to the clock line?
  - (a) How fast is the clock running?
2. Which probe would you say is the attached to the data line?
3. It is possible to set up the Agilent MSO 2000 series scope to both trigger and decode this specific bus. Please submit a screenshot of the scope triggering on a start condition and successfully decoding the transmitted data.
4. Can you describe the general data packet format? What is the purpose of each bit in the transaction?
  - (a) If you tilt the PCB in one direction how does the data packet change?  
How does it stay the same?
5. If you didn't have conveniently labeled traces to probe how would you figure out which traces to probe if you needed to understand how the bus worked?
6. How would you prevent attacks such as this? If the data transmitted over this bus was critically important and could not be intercepted how would you secure the communications?
  - (a) How would you then debug any issues that arose because of the proposed fix(es)?

### **Optional Followup (Extra Credit)**

The previous bus isn't the only data communication line on the PCB. There is another, much larger bus that enables the Atmel IC and the Altera IC to communicate with each other. Using the HP/Agilent logic analyzer cable attach the probe ends to all of the pins at the bottom of the PCB as well as the pin labeled GND at the top.

Repeating the same process as above (moving the board around and observing the data) answer the following questions:

1. How does this data differ from that of the other bus?
2. How is this bus laid out? What is the data packet like?
3. What is the advantage of a faster, smaller bus over a larger, slower bus? Are there any disadvantages? Provide a real-world example of this difference.

### 7.2.3 References and Further Reading

- Huang, A. “Hacking the Xbox: An Introduction to Reverse Engineering,” No Starch Press, 2003.
- Steil, M. “17 Mistakes Microsoft Made in the Xbox Security System,” presented at the 22<sup>nd</sup> Chaos Communication Congress, Berlin, 2005.
- NXP Semiconductors, “I2C-bus specification and user manual,” 4 April 2014.

## 7.3 Solution

1. SCL is the clock line and it has a frequency of 400kHz.
2. SDA is the data line.
3. See Fig. 7.2
4. The format of the data can be seen in Table 7.1.
5. The first step would be to find the clock line. Almost every digital signal has a repeating clock at a constant frequency. You’d probe around on the board until you find a square wave at a constant frequency. Once the clock is found other pins would be probed one at a time until a more complete picture is formed.
6. One way to passively prevent probing is to hide the signals in an inner-layer of the board and use IC packages such as a Ball Grid Array (BGA) that allow a direct connection to the inner-layer so no probe points are available. Of course

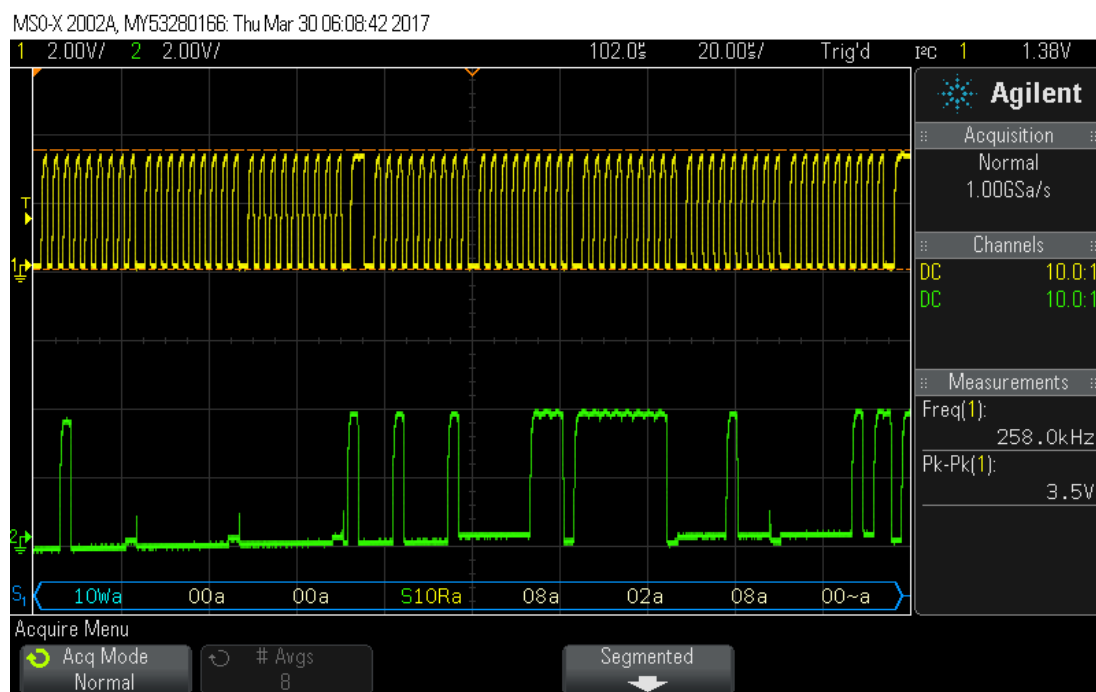


Figure 7.2: The screen of an Agilent MSO 2002A configured to decode I2C data and to trigger on a I2C start bit. It should be noted that the measured frequency of Channel 1 is incorrect. The correct value is 400.0kHz.

Description	Value
Start Bit	
Write Address	0b00100000 (0x10 + Write)
Register Address	0x00
Register Contents	0x00
Start Bit	
Read Address	0b00100001 (0x10 + Read)
Register Contents	X Axis Upper Byte
ACK	
Register Contents	X Axis Lower Byte
ACK	
Register Contents	Y Axis Upper Byte
ACK	
Register Contents	Y Axis Lower Byte
NACK	
Stop Bit	

Table 7.1: The contents of an  $I^2C$  transaction between the microcontroller and the accelerometer.

this also prevents easy debugging of the product so test points are often brought to the surface. It is a constant battle — products that are easy to debug are also easier to hack into.



# Chapter 8

## Lab #5: Reverse Engineering

### 8.1 Objectives

- To be able to approach an unknown system and derive information about its functionality.
- To be able to create a schematic of a board from visual analysis and continuity probing.
- To be able to create a Bill of Materials of a board from visual analysis.
- To be able to create a software interface with a board with zero provided documentation.

### 8.2 Assignment

#### 8.2.1 Introduction

One of the fundamental issues in digital content protection is known as the “Analog Hole”. Simply stated it means that if a person can view content then they can make a copy of it. This is true for almost all aspects of life: from making mix tapes by recording radio stations to copying a movie by pointing a camcorder at a television

screen. Another important area of life that is subject to this hole is in Intellectual Property (IP) Protection. You can have all the IP Protection you want but if your adversary has physical access to the protected IP then you cannot guarantee that they will not be able to copy it. For example, an IC can be decapped and a transistor level schematic can be easily made with a microscope.

## 8.2.2 Instructions

In order to copy IP well one first needs to understand how it works. One way of doing this is to build up a schematic and a Bill of Materials. Most of this can be done by either visual inspection of probing points on the board with a Digital Multi-Meter set to Continuity Mode. This will cause a loud beeping sound whenever the probes are making electrical contact together.

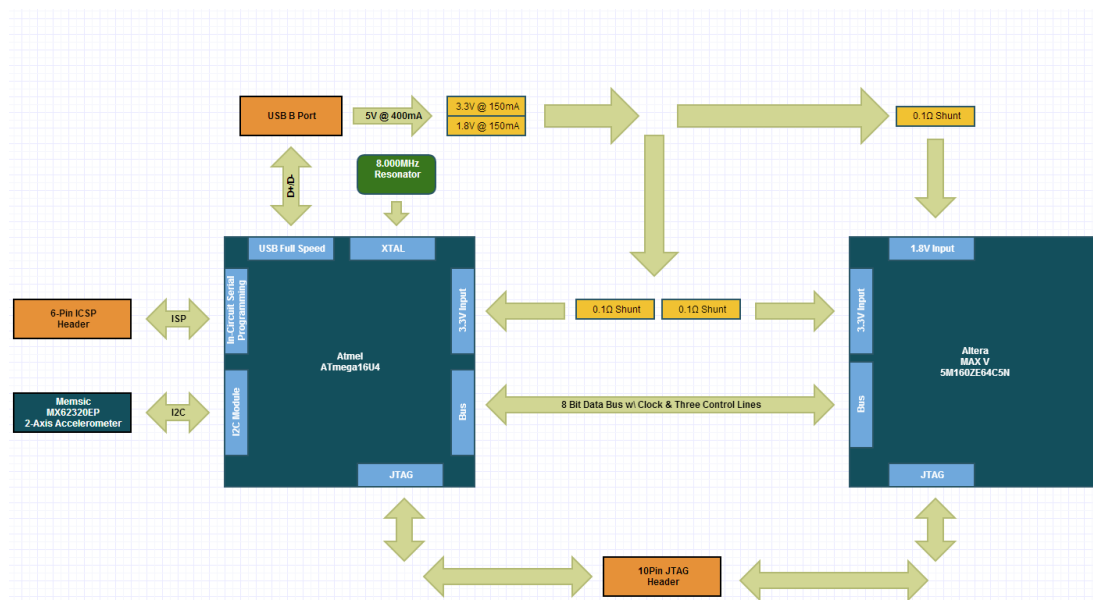


Figure 8.1: Block diagram of the demo Printed Circuit Board (PCB).

For a simple two-layer PCB (which our demo board is), this is typically all that is needed. One step that might be helpful is to look up datasheets for all known components to see how the component needs to be connected with other components on the board. Where are the known power pins: where are the known ground pins?

Are there any pins that have one specific functionality? Does the IC implement any known functionality (e.g. industry standards) that have to be attached a certain way? (Hint: Remember  $I^2C$  from last week?)

The final part of this assignment is to write a computer program that reads the data the boards sends and graphs it. This will involve learning how to open and read information from a serial port in a programming language of your choice. Remember that the serial port on this device is configured as a 9600 baud port with “8-N-1” settings. Once the program can read the data being transmitted, it should graph all six values the board continuously transmits.

### 8.2.3 Deliverables

For the hardware-based reverse engineering part of this assignment you are required to turn in two things for this experiment. One of them is a Bill of Materials. It should be structured as such:

RefDes(s)	Mfg.	Model	Description	Qty
U2	Atmel	MEGA16U4-AU	8 Bit Microcontroller	1
U3	Altera	5M160ZE64C5N	160 Logic Element CPLD	1
D1, D2, D3, D4	???	???	Green LED	4
J1	???	???	Bright Orange USB-B	1

Table 8.1: Example Bill of Materials for the assignment.

Please note that every element on the PCB will require a line-item in the Bill of Materials. Some components show up more than once; others will only show up once. Every component that you see should show up on the Bill of Materials at least once. Some components that look identical at first glance might be subtly different in some aspect, for example, the color of the component or a single digit in the part number. Color is important, for example gold plating looks different than tin plating and the difference is relevant to electronics. You will be expected to figure out the values of all components with the exception of the capacitors. There are two different

value capacitors used. It is not important to figure out the values (which can be surprisingly difficult; however, you should be able to differentiate between them.

The second thing that you need to turn in is as complete of a schematic as your group can generate.

For the graphing software, you should turn in your complete source code (over email) and answer the following questions:

- What do each of the three values reported for each axis represent?
- What is the mathematical relationship between the three values? (Hint: FIR or IIR?)
- Using the figure you turned in for the bus snooping assignment as reference which values were you observing on the  $I^2C$  bus?

There is no extra credit for this experiment. Please note that **physical modification of the PCB is strictly prohibited!** Any group found to have modified their PCB in any way will have points taken off.

## 8.3 Solution

The solution to this assignment is the Bill of Materials in Table 2.1 and the schematic depicted in Fig. B.1.

# Chapter 9

## Lab #6: Physically Unclonable Functions

### 9.1 Objectives

- To become familiar with Physically Unclonable Functions (PUF) in theory and practice.
- To write a SRAM-based PUF for a microcontroller.
- To be able to reliably distinguish between two different Arduino's running the same firmware using a PUF.

### 9.2 Assignment

#### 9.2.1 Introduction

One of the biggest issues in any supply chain management situation is being able to verify that the product ordered is actually the product being received. The electronics industry makes great strides to verify the integrity of their product lines. For example, the distributor that your group likely used to complete the Bill of Materials for the

reverse engineering assignment, Digi-Key, is a member of the trade association, the Electronic Component Industry Association, or ECIA, which strives to ensure that no counterfeit devices enter the supply chain.

However, one cannot always order from verified suppliers. In this case it is necessary to order components from unauthorized resellers, increasing the risk of acquiring counterfeit goods. One emerging research area to counteract this problem is in the area of Physically Unclonable Functions, or PUFs. The gist of the idea of a PUF is that the manufacture of the goods will use some sort of intrinsic process variation to generate a unique signature for every device they make. This signature is then entered into a database so that the end user can, using the same signature generation process, generate and verify the signature of their device.

### 9.2.2 Instructions

There have been several different kinds of PUFs created for use on different kinds of integrated circuits. The type of PUF we will focus on in this experiment is the so-called SRAM PUF. Using the distributed Arduino UNO R3 boards, write a small program to run the SRAM PUF algorithm on the built-in two kilobytes of SRAM on the Arduino's Atmel ATMEGA328P microcontroller.

Finally, using a MATLAB (or similar) script, generate the actual signature using a method of your choice as well as the inter- and intra-Hamming distances for the generated signatures. The goal of this experiment is to run the exact same code on each of the two Arduino UNO's and receive the same result when ran on the same board but a different result when run on a different board.

Please turn in all the code that you have written for this experiment. Please note that this process does not have to be automated (that is to say that your MATLAB script does not need to directly communicate with the Arduinos). In order to have good statistical significance for this experiment run the PUF on the Arduino multiple

times with a full power off cycle between runs.

### 9.2.3 References

This assignment is based upon the masters thesis of a student in the Department of Computer Science at the Czech Technical University in Prague[19].

# Chapter 10

## Final Project & Paper

### 10.1 Objectives

- To research and write about a security incident in the real world involving a hardware hack.
- To build a small embedded system.
- To see how a small embedded system can be hacked.
- To see if modifications to an embedded system can be detected.

### 10.2 Final Paper

This class has discussed both software and hardware attacks and how to make systems more secure by defending against a broad range of attacks. However, in the wild there are so many systems with so many different kinds of vulnerabilities getting attacked it is hard to keep track of all of the attacks.

While there are many examples of software-only attacks, attacks that use a combination of hardware and software (or of hardware-only) are much more rare and interesting. You are to find an example of a system that has been compromised by attacking a component of hardware means sometime in the past twenty years and



write a paper with the following sections:

- A general overview of the system
- How the maker of the system tried to prevent attacks.
- How the attacker attacked the system.
- The effects of the system being compromised.

Your paper will be submitted as both a PDF to Blackboard as well as a physical copy on the due date. It must adhere to the IEEE Conference Proceedings template. It should be at least four pages long and no longer than six pages. Please note that this length includes the references section — which must be in IEEE format.

This paper will be due on the last day of class.

Please note that this is an individual assignment, not a group assignment. Submitted papers will be checked using automated software to detect plagiarism.

## **10.3 Final Project**

### **10.3.1 Introduction**

Today's world contains many, many of embedded systems. These systems are designed for one specific task and only execute that task. Often security is an afterthought after the design phase, if it thought about at all. For this experiment your group will design an embedded system and then see if it can be hacked. This experiment is divided into three parts.

### **10.3.2 Part 0: Project Proposal**

The class should split into as many groups of three as possible and each group should have experience with both hardware and software development. From here the group should come up with a simple embedded system that can be developed over the course of a month and submit their proposal to the instructor.

This part of the project should happen as quickly as possible once the semester starts in order to give students enough time to build their project.

### **10.3.3 Part 1: Building the Project**

Once the instructor approves the project and gives a budget for parts, the group has most of the semester to develop the hardware and write the software for the project. Examples of projects done in this class include a wireless motor controller, a wireless authentication module, a small robot and a small keypad.

This part of the project should have a due date roughly a month before the semester ends in order to give enough time for Parts 2 and 3.

### **10.3.4 Part 2: Attacking the Project**

Once the projects are completed, groups will swap embedded systems as well as exchange source code repositories. Your group will have two weeks to modify the other group's embedded system to insert malicious functionality and try to keep it undetected.

This part of the project should have a due date roughly two weeks before the semester ends in order to give enough time for the final part.

### **10.3.5 Part 3: Defending the Project**

Once the hacks are completed, groups will swap embedded systems back; however, they will keep their "hacks" secret. Your group will have two weeks to figure out what malicious functionality was inserted into the system by the other group.

This part of the project should be due on the last day of class and students should demonstrate how they discovered the hack(s) done on their project and how they could possibly be fixed.

# Chapter 11

## Results & Conclusion

### 11.1 Results

This course was taught twice at Case Western Reserve University: in the Fall 2014 semester with twelve students and in the Fall 2015 semester with thirteen students. Both times the class was taught in the Electrical Engineering and Computer Science department with class designation of EECS 397: **Special Topics in Hardware Security**. The course was not substantially changed between iterations, although student feedback was taken into account between the offerings and modifications were made.

#### 11.1.1 Fall 2014

The first time the course was offered was during the Fall 2014 semester. It was offered as a three credit hour upperclassman special topics elective. Courses in the department that are only going to be taught once, or in our case, being taught for the first time, are given the class number of 397.

Of the twelve students enrolled in the class, ten of them had declared engineering majors. Five of the students were declared as Computer Science majors, three

Assignment	Assigned	Due Date	% of Grade
Lab #1: Buffer Overflows	8/27/2014	9/3/2014	10%
Lab #2: Encryption (AES)	9/3/2014	9/17/2014	10%
Lab #3: Bus Snooping	9/17/2014	10/1/2014	10%
Lab #4: Hardware Trojans	10/1/2014	10/15/2014	10%
Lab #5: Reverse Engineering	10/15/2014	10/29/2014	10%
Lab #6: PUFs	10/29/2014	11/10/2014	10%
Lab #7: Final Project Part #1	11/10/2014	11/17/2014	10%
Lab #7: Final Project Part #2	11/17/2014	11/24/2014	10%
Lab #7: Final Project Part #3	11/24/2014	12/3/2014	10%
Final Paper	11/10/2014	12/3/2014	10%

Table 11.1: The sequence of assignments for the Fall 2014 version.

as Computer Engineering majors and two as Electrical Engineering majors. Two students were enrolled in the School of Arts & Sciences, with one in the Economics department and the final student's major being International Studies.

Initially the syllabus for the course had two quizzes scheduled instead of a final paper, however it was decided that a paper was a more appropriate use of time and resources.

Towards the end of the semester the students were solicited for in-person feedback separately from the system that Case Western Reserve uses for official course evaluations.

The two groups for the final project were randomly assigned, for a total of six students in each. This had the side effect that one group had more Computer Science students in it than Electrical Engineering students. Some students stated that it would be better if the groups were systematically assigned by the instructor so that there would be equal numbers of all majors per group. This way the group could play better to the individual strengths of each of the members.

Students also said that while they appreciated the lack of guidelines for the final project they stated that a small number of guidelines, such as recommended projects or an example project would make the project experience work better. Another

common theme was that everyone wanted more time to work on their final project, as well as accountability checkpoints throughout the project. Finally students asked that the final project be integrated somehow into the experiments during the earlier parts of the semester.

Students also wished that the lab assignments had more of a background section in them for those who are unfamiliar with the subject area and are learning about the topic(s) for the first time.

One topic that was lacking in the class is that when students wrote code, it was almost always in C. One student expressed interest in the possibility of adding a Hardware Descriptor Language (HDL) such as Verilog or VHDL to the course. Other students disagreed with this and were glad there was no HDL requirement for the class.

As mentioned previously, Case Western Reserve conducts official course evaluations at the end of the semester. Out of the twelve students enrolled in the class, six responded.

When asked how this course fit into their academic program, 100% responded by saying that this was an optional or technical elective in their major. 50% of the class were Juniors and 50% of the class were Seniors when they took it.

When asked about the pace of the course, 17% said that it was rather fast while 83% said that it was moderate. Likewise, when asked about the workload of the course 67% said that it had a moderate workload while 33% said that it had a rather light workload.

Next, students were asked to rate the class on a scale of 1–5, where a score of one indicates that the student “strongly disagrees” with the statement while a score of five indicates that the student “strongly agrees” with the statement. When asked if the instructor has an effective command of the subject, the average response was a 4.83. When asked if the instructor speaks and writes clearly the average

Assignment	Assigned	Due Date	% of Grade
Lab #1: Encryption (DH Key)	8/26/2015	9/9/2015	10%
Lab #2: Buffer Overflows	9/9/2015	9/30/2015	10%
Lab #3: Bus Snooping	9/30/2015	10/21/2015	10%
Lab #4: Hardware Trojans	10/21/2015	11/04/2015	10%
Lab #5: Reverse Engineering	11/04/2015	11/11/2014	10%
Lab #6: Final Project Part #1	10/7/2015	11/16/2015	10%
Lab #6: Final Project Part #2	11/16/2015	11/30/2014	10%
Lab #6: Final Project Part #3	11/30/2015	12/7/2015	10%
Final Paper	10/7/2015	12/7/2015	20%

Table 11.2: The sequence of assignments for the Fall 2015 version.

response was a 4.33. When asked if the instructor's expectations of what the students should be learning were clear the average response was a 4.5. When asked if course procedures were clearly explained the average response was a 4.0. When asked if the instructor was able to motivate students' learning of the subject the average response was a 4.50. When asked if the instructor encouraged questions and/or appropriate discussions about the subject the average response was a 4.67. When asked if the course stimulated critical thinking the average response was a 4.83. When asked if the course was conducted in an atmosphere of mutual tolerance, courtesy and respect the average response was a 4.67. When asked if they thought they were informed of their progress in a timely manner the average response was a 4.83. When asked if grading was done fairly the average response was a 4.33. When asked if adequate assistance was available outside of class time the average response was a 4.50.

When asked to rate the course overall the average response was a 4.50. When asked to rate the instructor overall the average response was a 4.67. When asked to rate the labs overall the average response was a 4.75.

### 11.1.2 Fall 2015

In the Fall 2015 semester the course was again offered under the title of EECS 397: Hardware Security as a three-credit hour upperclassman technical elective. The

feedback mentioned in Section 11.1.1 was taken into account and some small modifications were made. Both the final project and the final paper were earlier in the course calendar relative to when they were assigned in the Fall of 2014, resulting in approximately an additional month to work on the assignments.

Of the thirteen students enrolled in the class, twelve had declared engineering majors. Six of those students were declared with Computer Science as their major, with the other six being split equally as Computer Engineering and Electrical Engineering majors. The last student was enrolled in the School of Arts & Sciences as a Music major.

The order of the first two labs was reversed and the encryption lab was changed from covering AES to covering Diffie-Hellman Key Exchange. The Physically Uncloseable Function lab was slated to be assigned as Lab #6 but was cut when the course progressed slower than expected and there was not enough time to cover all of the labs while having an expanded final project, which was deemed to be more important than the canceled assignment.

As with before, towards the end of the semester in-person feedback was solicited from the students. Overall students wanted more lectures about security and fewer class periods dedicated to completing projects and lab assignments. They also yearned for more hardware hacks, due to their novelty and consequently wanted less time dedicated to software hacks.

Students expressed an appreciation for the variety of skills and experiments that the class exposed them to, although several students wanted more focus on how to “package” hacks for delivery and exploitation — this class is setup for teaching students *how* to hack into devices and *why* hacks are possible, not necessarily the best way to actually exploit a device.

The second time the class was taught deadlines were more flexible and students responded by requesting more fixed deadlines so they could plan their schedules a bit

better. One consequence of this flexibility was that the aforementioned PUF project was canceled in favor of expanding the time available to work on the final project. Two students expressed regret they were not able to work on the PUF project.

One oddity in the scheduling of this class was due to how Thanksgiving Break broke up the penultimate week of class, which resulted in Phase #2 of the final project being concurrent with the Thanksgiving holiday, robbing students of possible time spent in the lab hacking into their opponents' project.

Even though the second time the course was offered the Final Project was given an additional month of time and checkpoints were implemented, several students expressed the regret that there were not enough checkpoints to keep them accountable during the additional month of time.

One overall consensus that the class reached is that even though the real-world examples were great, they were not “current topics” and students wanted more exposure to current trends in security.

There were two administrative issues that different students expressed. The first was that there was no official budget for the final project. Students were expected to pay for the projects themselves or through various other sources offered by Case Western Reserve. The second is that the class time slot conflicted with two labs for the introductory circuits course, which resulted in a lack of available lab space when the security class was scheduled to work in the lab.

A final note is that several students not in the class expressed regret at not knowing the class existed and wished that the class was more heavily advertised so that they could have taken it

For an unknown reason a detailed course evaluation was not performed for the Fall 2015 semester of this course. However, two students (out of thirteen enrolled) did respond and both of them rated the course as **Very Good**, or a 4.0 on the 1–5 scale used earlier.



## **11.2 Conclusion**

As elaborated on earlier, the two times the class ran it was a phenomenal success. The students were responsive, engaging and provided invaluable feedback towards the end of the semesters. As it stands the class is inexpensive to teach and is flexible enough to include in any engineering program. The unconventional mix of Electrical Engineering and Computer Science concepts together in a single class was well received and would make it possible to cross-list the class between departments for advanced technical elective credit.

This course slots well into the emerging Institute for Smart, Secure and Connected Systems (ISSACS) at Case Western Reserve University and can be expanded on with additional experiments to further integrate Internet of Things (IoT) concepts into the course's labs.

Further work with the concept could include a follow-up study with the twenty-five students who took the class to assess the amount of knowledge retained years after the semester concluded.

# Appendix A

## Proposed SAGES Course

It was thought that at one point, instead of teaching a second round of the class, a revamped and modified version of the class would be taught as a freshman SAGES class. The Seminar Approach to General Education and Scholarship (SAGES) program at Case Western Reserve replaced the stereotypical composition and writing classes that other universities offer with “themed” classes, pairing an instructor who is passionate about a subject with a skilled English instructor who provides a scaffold around which to promote discussion as well as compose essays and a final research papers. All students at Case Western Reserve are required to take three SAGES classes, starting with the first semester of their freshman year.

A recent push from the school’s administration has been to promote engineering-related SAGES classes, especially for first semester freshman in order to attract them to various engineering programs. A modified version of the class, tentatively titled “FSNA: Cybersecurity”, was proposed as an “engineering” themed freshman SAGES class. The freshman class was ultimately not offered and was abandoned in favor of teaching a slightly revised upperclassman technical elective a second time.

The freshman class would have dropped most of the detailed technical discussion and laboratory exercises in favor of looking at historical cybersecurity incidents and

Week	Topic
Week #1	Definition of Terms
Week #2	Buffer Overflows
Week #3	Wiretapping
Week #4	Social Engineering
Week #5	Class Presentations on Essay #1
Week #6	American Airlines Flight 587 and Uranium Processing
Week #7	Denial-of-Service Attacks
Week #8	Spoofing and Tampering
Week #9	Privilege Escalation
Week #10	Diffie-Hellman, RSA and Dual_EC_DRBG
Week #11	Original Xbox Issues
Week #12	Class Presentations on Essay #2

Table A.1: Tentative, proposed, schedule for the SAGES course.

their impacts on both the technology and the political landscapes. The class would have been split into thirds, with each third of a semester focusing on a different kind of attack.

The class would have started the semester off by taking a look at how an attack could be executed against a piece of software installed across tens of thousands, if not millions of systems. These kind of attacks can be disastrous due to their potential to affect software packages with huge install bases, therefore these attacks can generate hundreds of thousands of dollars in profit for their attackers. The text that would have underpinned this segment of the semester was *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage* by Clifford Stoll[6], the fascinating story of how an astronomer-turned-system-administrator uncovered a KGB spy ring running rampant across America through a seventy five cent internal accounting error.

After software-only attacks were discussed, attacks that focus on the interplay between software and hardware were going to be discussed. Ultimately the software running on a system controls the underlying hardware. If there is an issue in the control software then it is entirely possible to destroy the system, either inadvertently through mistakes made by the operator or on purpose, in the case of espionage. By

leveraging mistakes made in the design of the software it is possible to destroy the underlying hardware invisibly and undetectably until it is too late. The text that would have underpinned this segment of the semester was *Countdown to Zero Day: Stuxnet and the Launch of the World's First Digital Weapon* by Kim Zetter[16], the story of how part of the Islamic Republic of Iran's nuclear program was damaged and delayed through a computer virus that caused uranium purification centrifuges to self-destruct .

Finally, for the last third of the semester, the hardest kind of attacks would have been discussed: attacks attacking the hardware of a device, without touching the software running on that device. Modern days secure systems are constructed using a principle of a chain of trust: every link on the chain verifies the integrity of the next chain. If the verification fails then the system halts the process. However, if the authenticity of the underlying hardware can't be guaranteed then the entire chain is broken from the beginning. An attacker can modify parts of the hardware so that the chain will execute their code instead of the legitimate code transparently. The results of these attacks can have implications running into the millions of dollars. The text that would have underpinned this segment of the semester was *Hacking the Xbox: An Introduction to Reverse Engineering* by Andrew Huang[3], the story of how an MIT student cracked open the original Xbox's security system through reverse engineering part of the motherboard.

The goal of this course would have been to have students apply critical thinking skills to the modern day network of computers, and to explore how the systems normally work together to archive legitimate goals, but how along the way they can be hijacked to preform illicit actions. Each section of the course would have an overarching reading that drives the weekly readings, which describe different types of cybersecurity attacks in greater detail, along with a unit essay that wraps up the themes discussed in class during that section with each student researching an attack

that fits that particular theme.

Each attack discussed would have been demonstrated to the class through journal articles that describe how the attack works on a technical level, as well as examples taken from the media of the attacks being used outside of a laboratory setting with production systems. Selected attacks would have been demonstrated to the students in a live setting, which would have been designed to reinforce the readings by showing how exploits actually attack a system in practice.

# Appendix B

## Layout Files

This section contains all of the files needed to reproduce the custom-built board for this class, along with the Bill of Materials located in Table 2.1 in Chapter 2. As mentioned previously this is a two-layer Printed Circuit Board with soldermask on both sides and silk screen only on the top side.

The copper and silk screen images are positives — that is to say a black line represents where the copper should be left on the board or the silk screen applied. This is in contrast to the soldermask image which is a negative — that is to say a black line represents where there should *not* be soldermask applied.

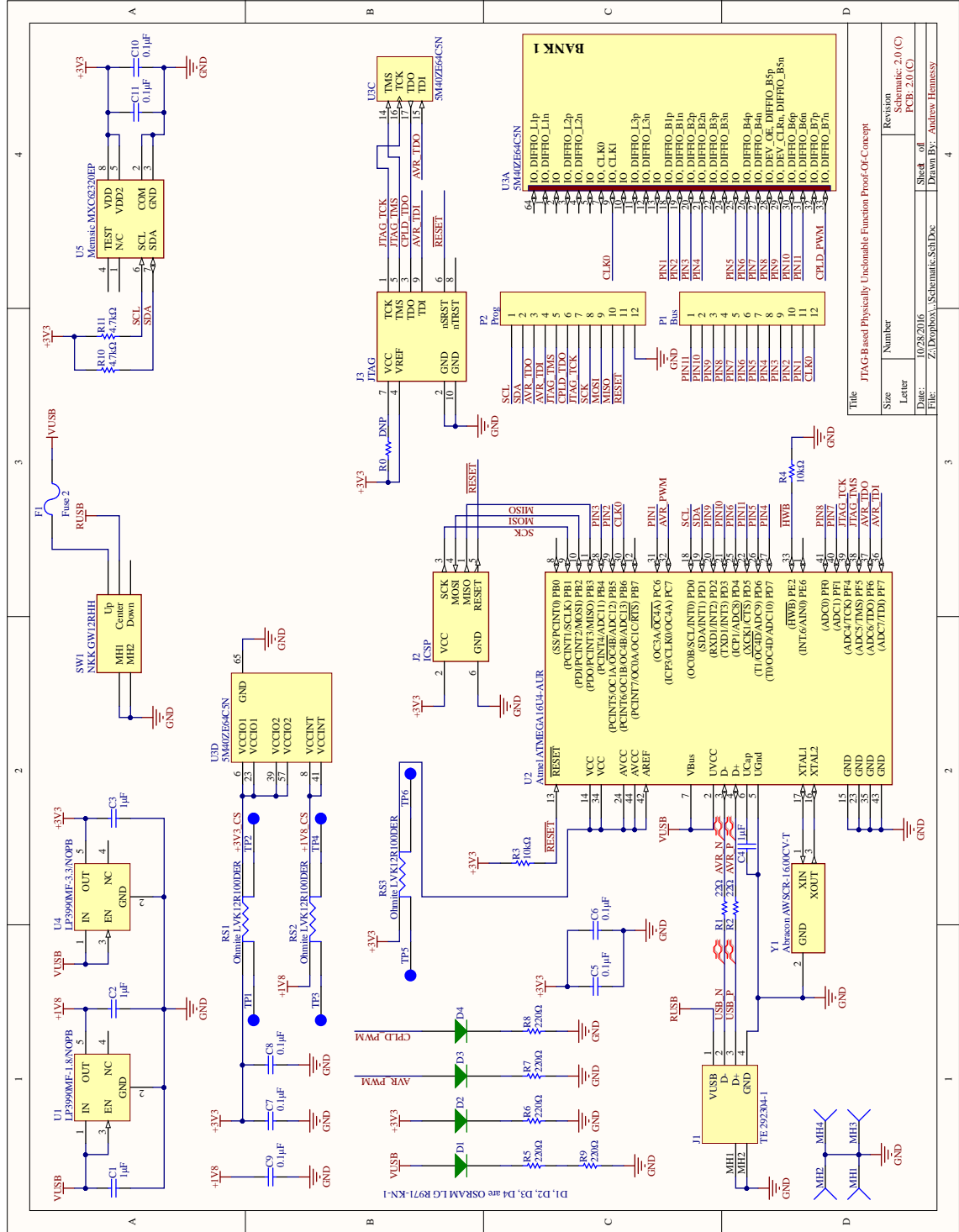


Figure B.1: The schematic of the purpose-built board.





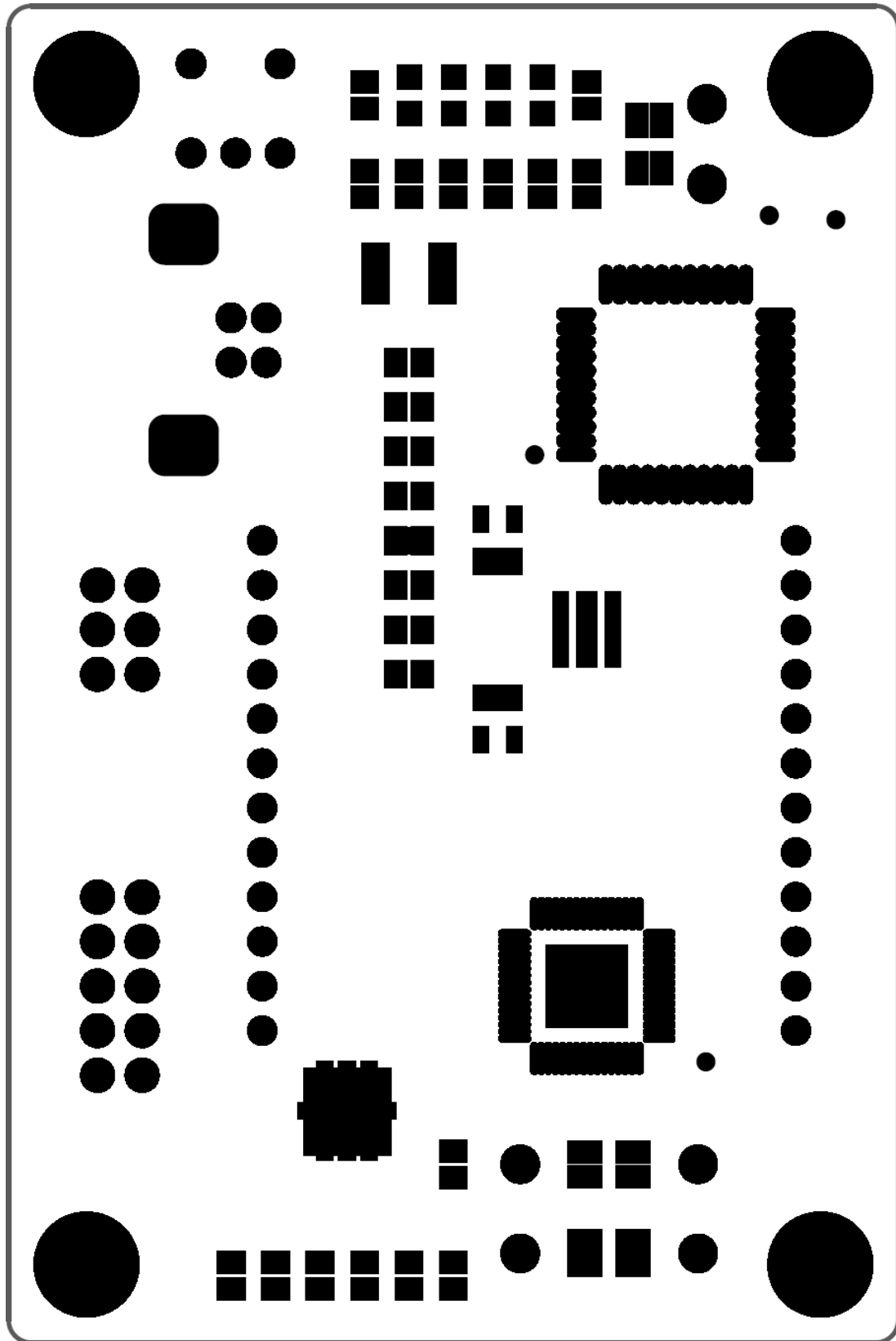


Figure B.3: The top soldermask graphic.

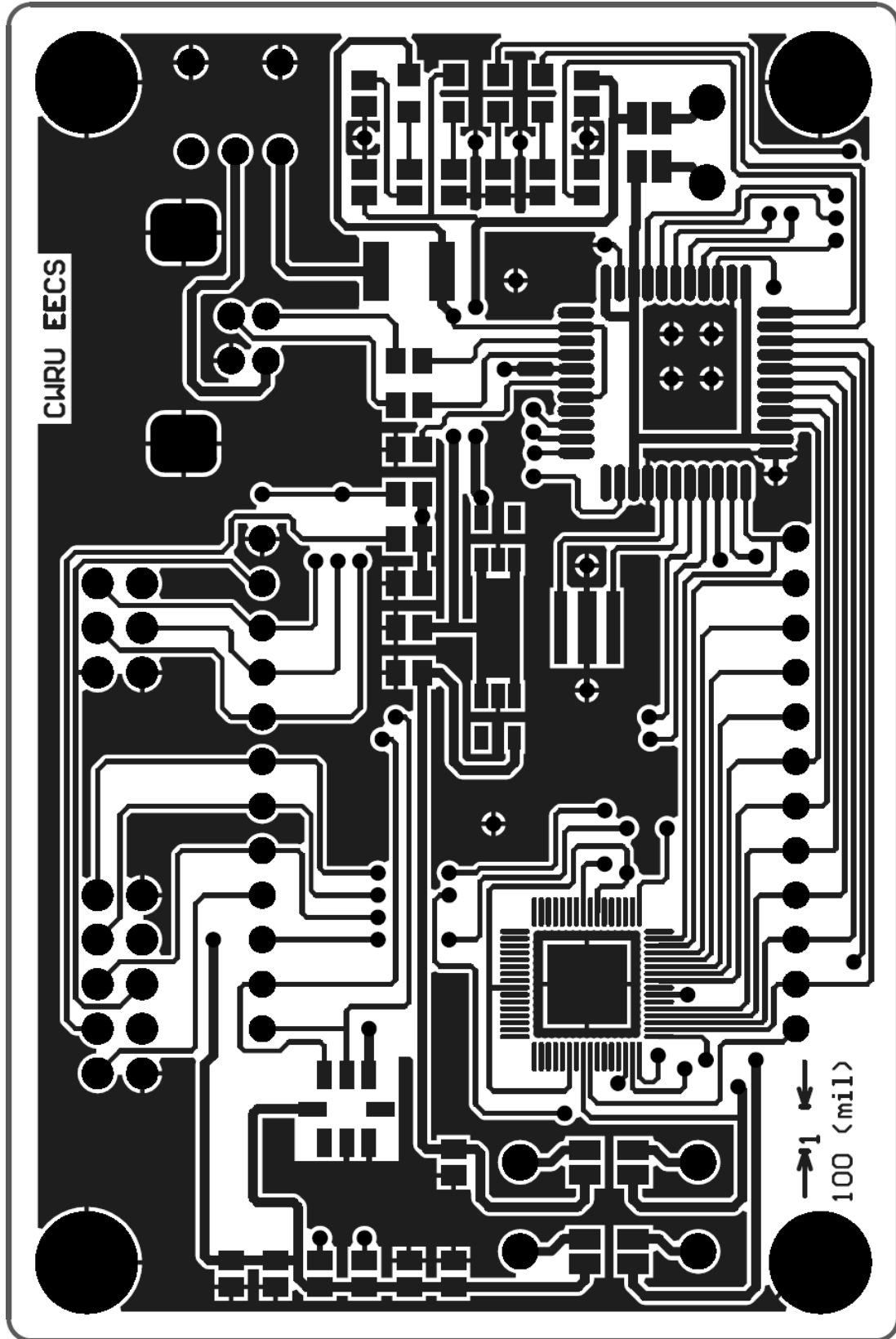


Figure B.4: The top copper graphic.

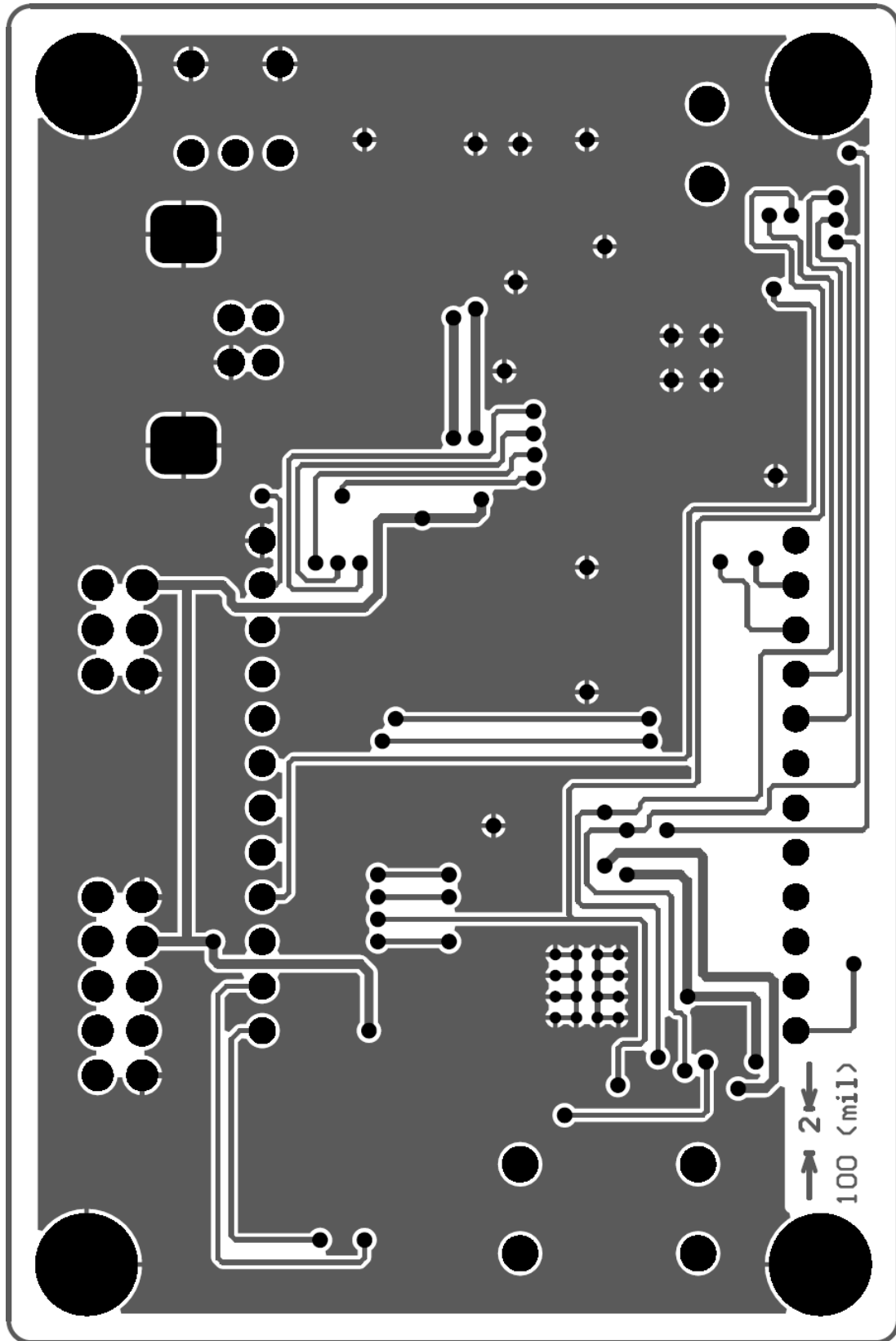


Figure B.5: The bottom copper graphic.

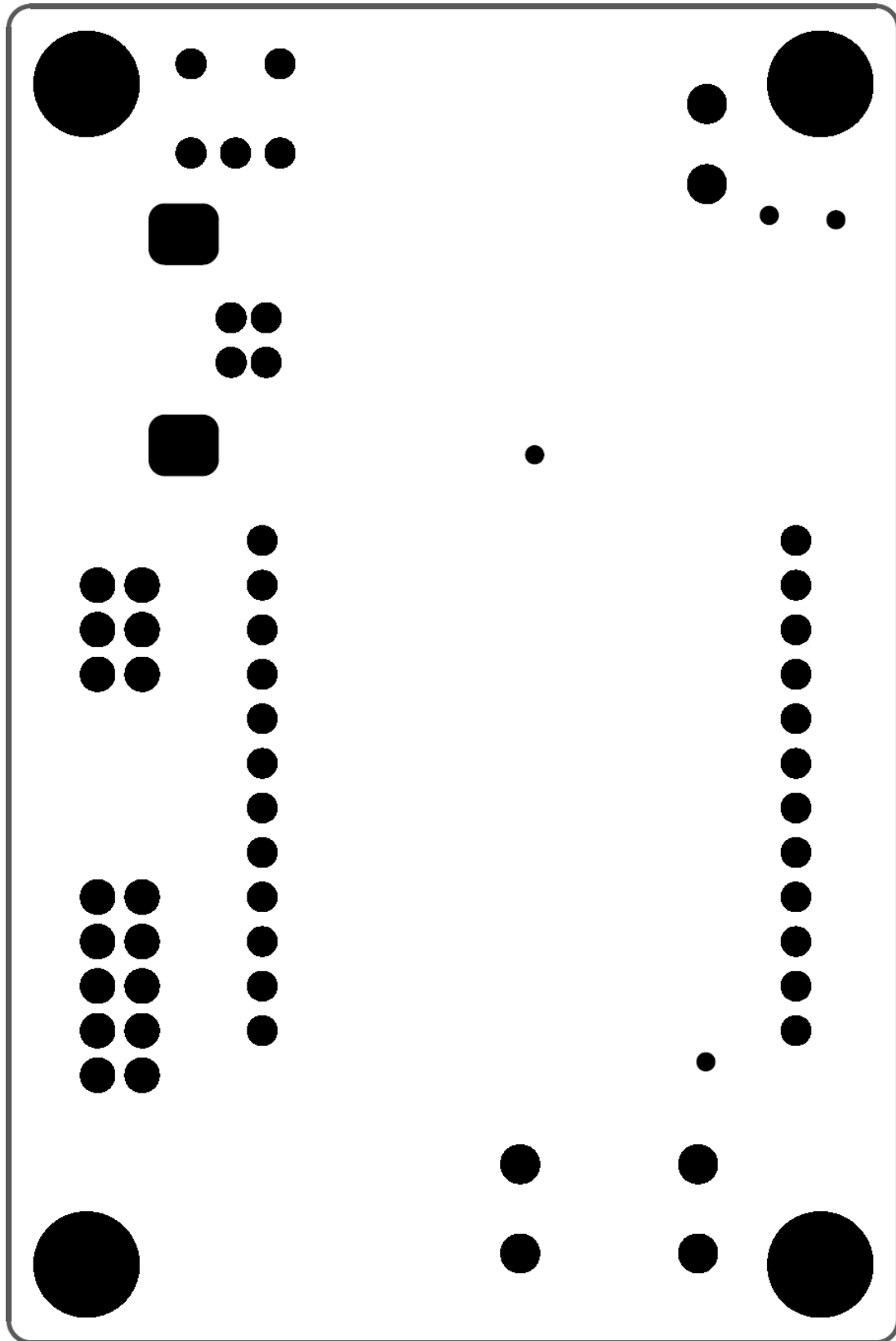


Figure B.6: The bottom soldermask graphic.

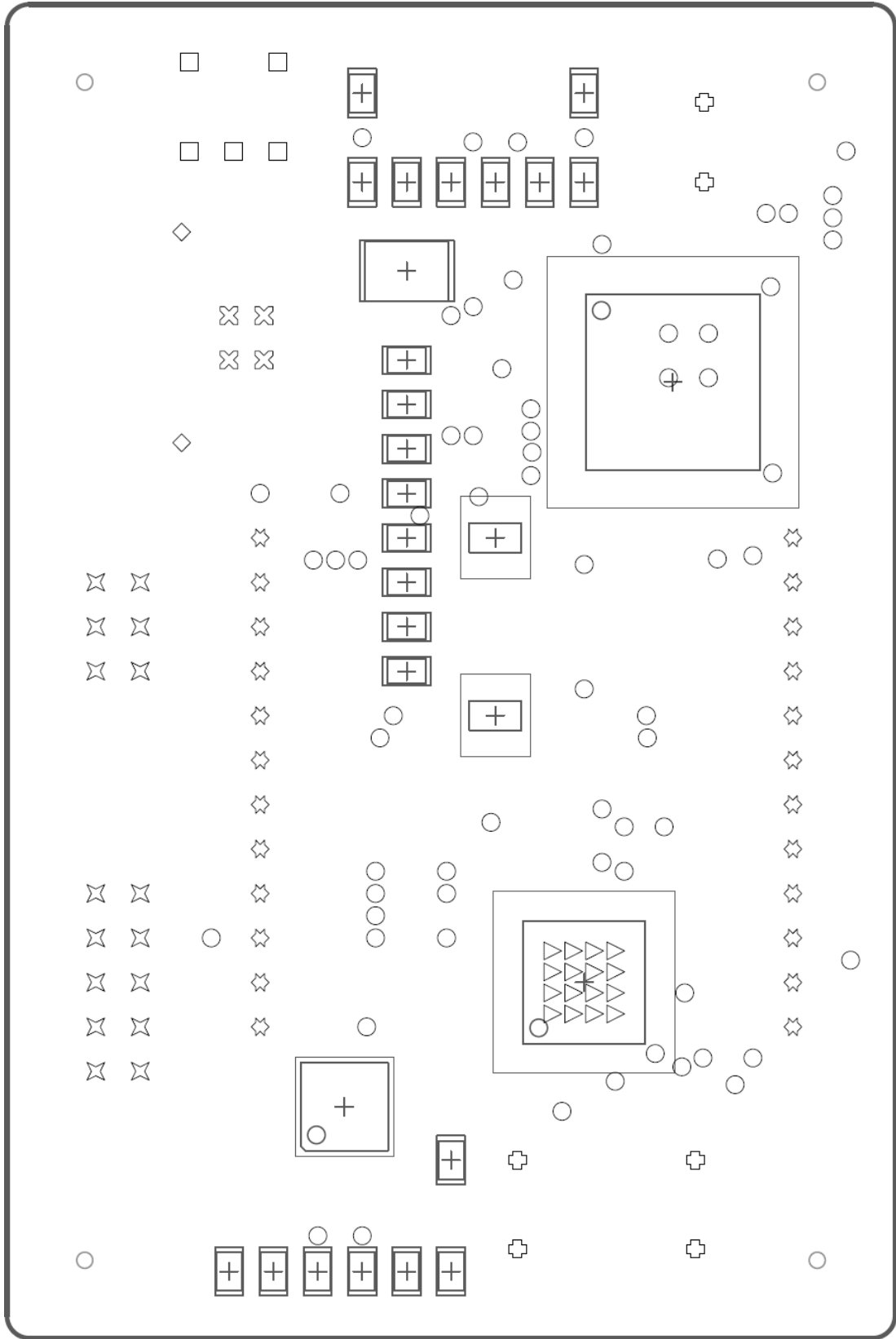


Figure B.7: The drill locations.

Designator	Mid & Ref X	Mid & Ref Y	Pad X	Pad Y	Rotation
C1	2600mil	4100mil	2600mil	4129.528mil	270
C2	2500mil	4100mil	2500mil	4129.528mil	270
C3	2700mil	4100mil	2700mil	4129.528mil	270
C4	3000mil	4100mil	3000mil	4070.472mil	90
C5	3600mil	4200mil	3629.528mil	4200mil	180
C6	3800mil	3700mil	3770.472mil	3700mil	0
C7	1150mil	4100mil	1179.527mil	4100mil	180
C8	1150mil	4000mil	1179.527mil	4000mil	180
C9	1400mil	4000mil	1370.473mil	4000mil	0
C8	1150mil	4000mil	1179.527mil	4000mil	180
C9	1400mil	4000mil	1370.473mil	4000mil	0
C10	1150mil	4500mil	1120.473mil	4500mil	0
C11	1150mil	4400mil	1120.473mil	4400mil	0
D1	3800mil	4100mil	3841.339mil	4100mil	180
D2	3800mil	4000mil	3841.339mil	4000mil	180
D3	3800mil	3800mil	3841.339mil	3800mil	180
D4	3800mil	3900mil	3841.339mil	3900mil	180
F1	3400mil	4100mil	3400mil	4025.197mil	90
R1	3200mil	4100mil	3200mil	4129.528mil	270
R2	3100mil	4100mil	3100mil	4129.528mil	270
R3	2900mil	4100mil	2900mil	4129.528mil	270
R4	3600mil	3700mil	3629.528mil	3700mil	180
R5	3600mil	4100mil	3570.472mil	4100mil	0
R6	3600mil	4000mil	3570.472mil	4000mil	0
R7	3600mil	3800mil	3570.472mil	3800mil	0
R8	3600mil	3900mil	3570.472mil	3900mil	0
R9	3800mil	4200mil	3770.472mil	4200mil	0
R10	1150mil	4200mil	1179.527mil	4200mil	180
R11	1150mil	4300mil	1179.527mil	4300mil	180
RS1	1200mil	3650mil	1227.559mil	3704.134mil	270
RS2	1400mil	3650mil	1427.559mil	3704.134mil	270
RS3	3690mil	3560mil	3744.134mil	3532.441mil	180
U1	2400mil	3900mil	2447.244mil	3862.598mil	180
U2	3150mil	3500mil	3307.48mil3	724.409mil	270
U3	1800mil	3700mil	1637.599mil	3818.11mil	0
U4	2800mil	3900mil	2752.756mil	3937.402mil	0
U5	1520mil	4240mil	1443.622mil	4290mil	0
Y1	2600mil	3700mil	2600mil	3640.945mil	90

Table B.1: The Pick-n-Place file for automated component placement.

# Bibliography

- [1] Altera Corporation. MAX 10 Device Handbook. [https://www.altera.com/content/dam/altera-www/global/en\\_US/pdfs/literature/hb/max-10/m10\\_handbook.pdf](https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/max-10/m10_handbook.pdf). Accessed: 2016-10-28.
- [2] Altera Corporation. MAX V Device Handbook. [https://www.altera.com/en\\_US/pdfs/literature/hb/max-v/max5\\_handbook.pdf](https://www.altera.com/en_US/pdfs/literature/hb/max-v/max5_handbook.pdf). Accessed: 2016-10-28.
- [3] Andrew Huang. *Hacking the Xbox: An Introduction to Reverse Engineering*. No Starch Press, 2003.
- [4] Atmel Corporation. ATmega16U4/ATmega32U4: 8-bit Microcontroller with 16/32K bytes of ISP Flash and USB Controller: Datasheet. [http://www.atmel.com/Images/Atmel-7766-8-bit-AVR-ATmega16U4-32U4\\_Datasheet.pdf](http://www.atmel.com/Images/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf). Accessed: 2016-10-28.
- [5] Charles Petzold. *Code: The Hidden Language of Computer Hardware and Software*. Microsoft Press, 2000.
- [6] Clifford Stoll. *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage*. Doubleday, 1989.
- [7] Digi-Key Electronics. 5M160ZE64C5N Altera — Integrated Circuits (ICs) — DigiKey. <http://www.digikey.com/product-search/en?keywords=5M160ZE64C5N>. Accessed: 2016-10-28.

- [8] Digi-Key Electronics. P0302 Terasic Inc. — Programmers, Development Systems — DigiKey. <http://www.digikey.com/product-detail/en/terasic-inc/P0302/P0302-ND/2003484>. Accessed: 2016-10-28.
- [9] Digi-Key Electronics. PPL-USB-BLASTER-RCN Altera — Programmers, Development Systems — DigiKey. <http://www.digikey.com/product-detail/en/altera/PL-USB-BLASTER-RCN/544-1775-ND/1212940>. Accessed: 2016-10-28.
- [10] Digi-Key Electronics. TMS320VC5509AZHHR Texas Instruments — Integrated Circuits (ICs) — DigiKey. <http://www.digikey.com/product-search/en?keywords=296-42647-1-ND>. Accessed: 2016-10-28.
- [11] Dr. Bulent Yener and Andrew Zonenberg. CSCI 4974 / 6974 Hardware Reverse Engineering. <http://security.cs.rpi.edu/courses/hwre-spring2014/>. Accessed: 2017-03-04.
- [12] ebay Inc. Altera Mini Usb Blaster Cable For CPLD FPGA NIOS JTAG Altera Programmer — eBay. <http://www.ebay.com/itm/200943750380>. Accessed: 2016-10-28.
- [13] Jeffrey Hoffstein and Jill Pipher and Joseph H. Silverman. *An Introduction to Mathematical Cryptography (Undergraduate Texts in Mathematics)*. Springer, 2014.
- [14] Jon Erickson. *Hacking: The Art of Exploitation: The Art of Exploitation*. No Starch Press, 2008.
- [15] Keysight Technologies. MSOX2002A Mixed Signal Oscilloscope: 70 MHz, 2 Analog Plus 8 Digital Channels. <http://www.keysight.com/en/pdx-x201828-pn-MSOX2002A/>. Accessed: 2016-10-28.



- [16] Kim Zetter. *Countdown to Zero Day: Stuxnet and the Launch of the World's First Digital Weapon*. Broadway Books, 2015.
- [17] MEMSIC Inc. Low Power, Low Profile  $\pm 1.5g$  Dual Axis Accelerometer with  $I^2C$  Interface: MXC6232xE/F. [http://www.memsic.com/userfiles/files/Datasheets/Accelerometer-Datasheets/MXC6232xEF\\_Rev\\_A.pdf](http://www.memsic.com/userfiles/files/Datasheets/Accelerometer-Datasheets/MXC6232xEF_Rev_A.pdf). Accessed: 2016-10-28.
- [18] MEMSIC Inc. Low Power, Low Profile  $\pm 2g$  Dual Axis Accelerometer with  $I^2C$  Interface: MXC6232xM. [http://www.memsic.com/userfiles/files/Datasheets/Accelerometer-Datasheets/MXC6232xMP\\_Rev\\_B.pdf](http://www.memsic.com/userfiles/files/Datasheets/Accelerometer-Datasheets/MXC6232xMP_Rev_B.pdf). Accessed: 2016-10-28.
- [19] Mikhail Platonov. SRAM-Based Physical Unclonable Function on an Atmel ATmega Microcontroller, 2013.
- [20] Mouser Electronics. IP-USB2S Altera — Mouser. <http://www.mouser.com/ProductDetail/Altera/IP-USB2S>. Accessed: 2016-10-28.
- [21] Saleae Inc. Saleae Logic. The logic analyzer you'll love to use. <https://www.saleae.com/#DatasheetTile>. Accessed: 2016-10-28.
- [22] Scott Culp. Ten Immutable Laws Of Security (Version 2.0). <https://technet.microsoft.com/en-us/library/hh278941.aspx>. Accessed: 2017-04-03.
- [23] Uli Kretschmar. AES software support for encryption and decryption. <https://github.com/errordeveloper/mist/blob/master/apps/aes/aes.c>. Accessed: 2017-03-31.