The S4 Infrastructure Management System

by

Rodrigo Toste Gomes

B.S., Massachusetts Institute of Technology (2015)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2016

Author .....
Department of Electrical Engineering and Computer Science
May 20, 2016

Certified by .....
Tomás Lozano-Pérez
School of Engineering Professor in Teaching Excellence
Thesis Supervisor

Certified by .....
Adam Hartz
Lecturer
Thesis Supervisor

Accepted by .....
Dr. Christopher Terman
Chairman, Masters of Engineering Thesis Committee

The S4 Infrastructure Management System

by

Rodrigo Toste Gomes

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2016, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis describes the design and implementation of a system for managing the infrastructure supporting a course relying on a large computer lab. Students' information privacy and security was an important focus in the design of this system, as well as integration with existing software systems to facilitate its deployment. Particularly, the design was informed by the needs of the MIT class 6.01 (Introduction to Electrical Engineering and Computer Science).

Thesis Supervisor: Tomás Lozano-Pérez
Title: School of Engineering Professor in Teaching Excellence

Thesis Supervisor: Adam Hartz
Title: Lecturer

Acknowledgments

I would like to thank all of the people—students and staff alike—that I have worked with as part of MIT's *6.01: Introduction to EECS* class. I learned a lot every semester I worked as a Teaching Assistant and, were it not for what I learned there, I would have written a very different thesis.

In particular, I would like to thank the Professors Leslie Kaelbling and Tomás Lozano-Pérez, and the Lecturer Adam Hartz, who have been amazing mentors throughout my career at MIT, and were always available to listen to my ideas, and brainstorm with me.

I would also like to thank all those who have directly contributed to the works presented in this thesis. In particular, Geronimo Mirano, a fellow TA of the 6.01 course, contributed code for the software presented in section C.1, and Alexander Couzens talked to me in the Coreboot IRC, guiding me on how to get the BIOS installed in the 6.01 laptops (specifically contributing to the implementation of the code in section C.2).

Additional thanks go to William Yashar, Jeremy Kaplan, and Shen Gao for their willingness to brainstorm with me, listen to my ideas, and help in multiple projects over the years.

Finally, I would like to thank my family for their continued support of all my endeavors, and unconditional love. I would not be where I am without them.

Contents

1 Introduction **13**

1.1 Motivation 14

1.1.1 Hardware Upgrades 14

1.1.2 Debathena without stable Internet 15

1.2 Kerberos Authentication 16

1.3 Outline 17

2 Issues and Initial Solutions **19**

2.1 Upgrading to Solid State Drives 20

2.2 Six-Oh Caster 23

2.3 Upgrading Wireless Cards 25

2.3.1 Installing the Coreboot BIOS 26

2.3.2 Results of the Wireless Upgrade 29

3 Replacing Debathena **33**

3.1 Issues with Debathena 33

3.2 The Six-Oh File Transfer System 35

3.3 The Six-Oh Authentication Program 39

4 Conclusions and Future Work **43**

A Attempted Software Installation of Coreboot **45**

B Historical Reasons for Choosing Debathena **47**

C Code **51**

C.1 certfix 51

C.2 vga_extract.sh 53

C.3 setup.sh 56

C.4 clone.sh 57

C.5 client.py 59

C.6 server.py 65

C.7 catsoop_login.py 72

C.8 firefox_sid_cookies.py 74

C.9 catsoop_login_pam.sh 76

D Figures **79**

List of Figures

D-1 Integrated Circuit Clip 80
D-2 SOIC-8 chip on T410 Motherboard 81

List of Tables

2.1 Metrics comparing new SSDs to old mechanical hard drives; read/write rates were done with 100 samples, access time was done with 1000 samples 22

2.2 Results of the Wireless Card Experiment, Packet Loss and Round Trip Time Metrics (in milliseconds) 30

Chapter 1

Introduction

This thesis focuses on the work done to improve the infrastructure for the 6.01[1] laptop lab. This work was done in the context of my work as a Teaching Assistant for the class, and looking to fix issues in the hardware and software infrastructure that got in the way of students' learning during lab times.

The issues we focused on were: blocking of software by the AFS[14] filesystem, the difficulty of maintaining a stable wireless connection by the laptops, and the delay when logging in and loading software. The solutions consisted of a comprehensive hardware upgrade to the lab laptops, and the design and partial development of a new system for managing the lab.

This new system consists of four main components. The Six-Oh File Transfer system facilitates students' management of files, keeping them synchronized and available across machines and sessions, and maintaining the students' privacy via encryption. The Six-Oh Auth Program unifies the student's authentication to the class's online tutor and to the lab laptops. The Six-Oh Caster system allows course staff to easily manage every machine, perform maintenance tasks, and identify issues with individual machines. Finally, the Six-Oh Deploy system allows course staff to easily deploy updates and new operating system images to every machine in the lab. Together, these systems comprise the S4 Infrastructure Management System.

[1] 6.01 is the Introduction to Electrical Engineering and Computer Science class at MIT. Every year about 600 students (about 200 in the Fall and 400 in the Spring) take it.

The systems described in this thesis were designed to tackle specific issues encountered by the 6.01 class, but these issues were generic enough that these systems would be helpful in other classes relying on a computer lab equipped with laptops and using a networked file system. For the rest of this document, the issues and solutions will be described in the context of the 6.01 class.

1.1 Motivation

The Introduction to Electrical Engineering and Computer Science class at MIT faces unique challenges due to the resources it requires and the number of students that consistently take it. The course offers a broad overview of EECS by teaching overarching concepts in depth, specifically linear systems, circuits, probability, and search, using robotics as a unifying application of these concepts and programming as a common tool for applying them.

Before the development of the systems presented in this thesis, 6.01 relied on a complex infrastructure of hardware and software systems to provide a unified authentication and file management interface to the students. The students interacted with ThinkPad T410 laptops during lab hours, which utilized the Debathena [2] operating system. Debathena is developed at MIT for the students. The home directories of students were kept in the Andrew File System (AFS), which provided synchronization and backup for their files. Authentication in Debathena used Kerberos[17] which students also used to authenticate to other services provided by MIT.

1.1.1 Hardware Upgrades

The machines used in the lab were slower than more recent machines, and we considered upgrading them to newer computers. We decided, however, to investigate whether upgrades to certain hardware components on the existing machines could make them perform similarly to newer machines. Some investigation suggested that the laptops could be comparable to newer machines in performance by upgrading

their hard drives to solid state drives, and upgrading to wireless cards that support the 5GHz band. These upgrades constitute the first part of this thesis.

Due to software blocks on the original Lenovo BIOS, we were unable to simply replace the wireless cards in the laptops. When we replaced them, the BIOS stopped the boot process. Thus, we also replaced the original BIOS with the open source Coreboot BIOS which did not have these blocks, and was actively maintained. Installing the Coreboot BIOS presented two major challenges: it required fully disassembling the laptops, which had never been done on a ThinkPad T410. To tackle these challenges, a hardware maintenance guide was created, and we adapted the procedure for installing Coreboot on a different laptop with the same chipset[1]. Two systems were also developed to facilitate the hardware upgrade, specifically the Six-Oh Deploy system, described in section 2.1, and the Six-Oh Caster system, described in section 2.2. The Six-Oh Caster system automates the installation of an operating system image in the laptops with minimal user interaction. The Six-Oh Caster system allows a member of the staff to perform maintenance tasks, software updates, and failure detection on all machines simultaneously.

1.1.2 Debathena without stable Internet

There were also issues with Debathena. Specifically, the design goals and constraints of Debathena did not align with the constraints we faced in lab, in particular the fact that our Internet connection was not stable

The Debathena distribution for workstations is intended to be used on machines permanently connected to the Internet. If the connection fails, AFS blocks, and since the user's home directory exists on AFS, all applications using that directory[2] block as well. On desktops connected over Ethernet (such as the ones found in "Athena Clusters" on campus) this is not a concern, but for our computer lab this quickly became an issue. The laptops in the lab had difficulty maintaining a stable wireless connection, and the subsequent blocking of applications when the network

[2]In practice, many applications. The browser was of particular importance, since that was the tool students use to interact with the online tutor we use in the class.

got disconnected would interrupt the students' work. It was also important that the machines were mobile because we used them to control robots via a USB cable. If we were to use desktops, mimicking the Debathena workstations on campus, we would constrain the possible interactions students could have with the robots.

Our design constraints were different from AFS's. While AFS's design prioritized a consistent state across replicas, we put a higher priority on the availability of the file system. We could achieve this by accepting a temporary inconsistent state between the laptops and the servers holding the files. An interesting property of the lab's infrastructure was that, while the Internet was not stable in the short term, in the long term there was high confidence that a stable Internet connection would be achieved for however long it would be necessary to synchronize the full file system. This led to the design of a new file system: the Six-Oh File Transfer system, described in section 3.2. This system provides the students with a file system that does not block in case of network failure, and synchronizes their files, achieving an eventually consistent state with high probability.

1.2 Kerberos Authentication

The decision to develop a new file system with our requirements as design goals meant that either Debathena would have to be modified to use this new file system, or the course would stop using Debathena. Most of the software offered by Debathena is tied to AFS, and due to the issues with AFS described above, we decided to phase out Debathena. This decision was further supported by the possibility that "Athena Clusters" throughout campus might be closing [19]. One aspect of Debathena, however, that was advantageous was its integration with Kerberos authentication. Every student has a Kerberos account that they use to authenticate to various services at MIT. No longer using Debathena exposed the question of whether Kerberos support should be maintained, with the advantages that it brings (unified authentication with other MIT services), or if other methods would be more appropriate and bring other advantages.

We could implement support for Kerberos authentication via a Pluggable Authentication Module[21] and a custom script for dealing with creating the local users and downloading their files. Using Kerberos came with a trade-off, however. To maintain the students' information privacy and security, there was a strong emphasis on maintaining only encrypted copies of the students' files. The system could encrypt files with the student's Kerberos when they first login. This would, however, have serious issues if the students were required to change their password at any time during the semester. To re-encrypt a student's files, we would need to be part of the Kerberos password change process. We currently are not involved in that process, and that would present a security risk, as Kerberos is used in many other MIT services. It was thus apparent that an authentication system managed by the course would be advantageous.

A home grown authentication service also had other advantages over Kerberos. It would be easier to use by teachers outside MIT, whose schools do not have Kerberos authentication, and also easier to add users[3]. With the advantages over Kerberos clear, we decided to develop a new authentication service, the Six-Oh Authentication Program, described in section 3.3, the final part of the S4 Infrastructure Management System

1.3 Outline

The remainder of this thesis is structured as follows:

Chapter 2 describes in detail the issues that motivated the laptop hardware upgrades, and explains the investigation that was done to find their causes. Then, it goes into detail on how the hardware upgrade was performed, specifically the installation of Coreboot on every machine, and the development of the Six-Oh Deploy and Six-Oh Caster systems to simplify the upgrade. It concludes with an empirical eval-

[3]There were often small classes with students from outside MIT in the 6.01 lab. These required manual creation of local users for those students to use, which a new system for authentication could make a lot easier

uation of the upgrades, showing how they tackled the issues described at the start of the chapter.

Chapter 3 describes in detail the issues associated with using Debathena, and which design goals and constraints an appropriate system for the lab would require. The Six-Oh File Transfer and Six-Oh Authentication Program are then presented as systems that meet those goals, and their design and implementation are explained.

Finally, chapter 4 provides concluding remarks, as well as suggestions for future work.

In addition, Appendices A and B contain historical accounts of alternative solutions to some of the problems solved in this thesis, and why they did not work; Appendix C contains the source code for the several systems described in this thesis; and Appendix D contains figures

Chapter 2

Issues and Initial Solutions

6.01 uses ThinkPad T410 laptops in the lab. These machines have specifications comparable to more modern machines[1], but we faced issues using them. Specifically, there was a delay when logging in[2] and loading software[3], and it was difficult to maintain a stable wireless connection. The difficulty with maintaining a wireless connection meant that the laptops were always connected to Ethernet[4], otherwise AFS would not work. This limited the mobility of the laptops, which limited the scope of projects available to the students. We taught students concepts in Electrical Engineering and Computer Science using robots as a motivating platform, and these connect to the laptops via a USB cable. The limited mobility of the laptops meant that the robots also had limited mobility, constraining the motions of the robots to a small area.

Due to these issues, we considered obtaining new laptops. The budget for new machines, however, allowed about $300 per machine, which would afford machines with similar specifications[3] as the ThinkPad T410 laptops in use (dual core CPUs, around the 2.0GHz clock speed mark, and 4GB of ram), and features that were not

[1]The specific variation we had used Intel Core I3-370M processors at a clock speed of 2.4GHz, with 2GB of RAM.

[2]It took about 16 seconds for a student to get to a working desktop after entering their password

[3]In particular, we had just upgraded to *Python 3.5*, and loading the library *Matplotlib* took a long time. This library was used in a lot of the course's software.

[4]This was not reliable either, as the Ethernet cables and ports got worn out with use, making it very easy to accidentally disconnect the Ethernet cable with a small amount of movement.

relevant to the course (such as touch screens and thinner bodies). One big motivation to buy new machines, however, was the fact that our laptops were unable to maintain a stable wireless connection in the lab room. Some experiments with potential new laptops showed that it should be possible to maintain a stable connection with new hardware. We decided, however, to investigate whether upgrades to certain hardware components on the existing machines could make them perform similarly to newer machines.

The first issue of delays when loading software seemed to have a simple solution: upgrade the hard drives to solid state drives. It was not entirely clear that this would improve the speed of login, as the use of Kerberos[17] and AFS[14] involved the use of the network for authentication. Evidence supporting this, however, was that most of the delay happened while the hard drive activity light was turned on.[5] The difficulty maintaining a wireless connection, however, was not well understood[6], and required some more investigation. It was concluded, however, that the source of the problem was the lack of 5GHz support in the cards we were using, and the solution was to upgrade them to newer ones.

To facilitate the upgrade to solid state drives, two systems were developed: the Six-Oh Deploy System, which allowed for a quick installation of a new system image on every machine, and the Six-Oh Caster system, which allowed staff to easily apply updates and fixes to all the machines simultaneously. These systems are described in section 2.1 and section 2.2, respectively.

2.1 Upgrading to Solid State Drives

We upgraded the mechanical hard drives to solid state drives not only to solve the delays when logging in and loading software, but also to extend the laptops' service life. The ThinkPad T410 laptops in use were purchased in 2011, and were still using

[5] AFS caches directories locally when they are requested, so it seemed likely that the hard drive activity was due to a large download of files, and the high amount of activity suggested that the bottleneck was the writing speed of the disk, rather than the network.

[6] Several hypotheses (bad drivers and bad routers) had been presented in the past and solutions tried, but all were proven wrong, as they didn't solve the issue

the same hard drives 4 years after being purchased. Although it is not clear what the real service life of consumer hard drives is, one of the most comprehensive studies done[6] suggests that the median service life is 6 years of use. These hard drives are not used continuously, so they likely have a longer life, but this suggests that they may be likely to start failing soon; therefore, new drives were acquired. Given the decreasing cost of solid state drives and their higher performance, we upgraded the laptops to solid state drives.

The initial strategy for performing this upgrade was to install the solid state drives on the machines, install Lubuntu on each, and update it with the software required for the class. This would have been a lengthy process, and upgrades of this scale were usually performed before the start of a new semester. This allowed time to perform the upgrade and test the new system to guarantee that it would work well without impacting the students. This upgrade, however, happened during the semester, so this initial strategy was not ideal. We instead used a different strategy. We first installed the Lubuntu operating system on one machine, and then fully cloned its SSD's contents to the other drives. Although this strategy seemed to work at first, it had three important issues. First, the Lubuntu operating system keeps track of hardware UUIDs to name its network devices, so while certain scripts were setup to initialize *wlan0* to connect to the wireless network, *wlan0* did not exist. Instead, the wireless interface was named *wlan1*. Second, the laptops used the *dhcpcd* DHCP daemon, which sends a unique identifier to the router so the router knows which machine it is talking to. Our initial assumption was that the unique identifier would depend on the MAC address of the network interface, but it actually depends on the network interface name (the same for all machines) and a separate unique identifier stored in a file (*/etc/dhcpcd.duid*) which was copied between computers. This resulted in DHCP conflicts. Finally, the cloning procedure was fairly slow, taking several minutes per drive, since it cloned every sector of the drive, even if that sector contained no information.

A new strategy was necessary, and this motivated the design and implementation of the Six-Oh Deploy System. The Six-Oh Deploy system, rather than cloning the

Metric	Mechanical	Solid State
Time to Login (Ethernet)	16	11
Time to Login (Wireless)	18	11
Time to StartUp (s)	29	11
Average Read Rate (10MB samples, MB/s)	65.0	274.4
Average Write Rate (10MB samples, MB/s)	57.8	219.3
Average Read Rate (1MB samples, MB/s)	65.7	248.1
Average Write Rate (1MB samples, MB/s)	35.8	145.8
Average Access Time (ms)	17.85	0.08

Table 2.1: Metrics comparing new SSDs to old mechanical hard drives; read/write rates were done with 100 samples, access time was done with 1000 samples

full hard drive, finds the smallest amount of information necessary to restore the system, and regenerates unique ids that get cloned. Specifically, this involved cloning the partition layout and partition types of the drive to clone; and copying the files, maintaining their original permissions, and resetting those files that set or relied on unique ids. This strategy was programmed in two separate scripts. The first copies the files and partition layout of a source hard drive, and removes files depending on UUIDs (this script, *setup.sh* can be found in section C.3). The second recreates the partition layout in a new drive and copies the original files to that drive. After completing the copy, the script also reconstructs the Linux */etc/fstab* file (which relies on partition UUIDs) and installs the GRUB boot loader (this script, *clone.sh* can be found in section C.4).

With this new strategy, the disk cloning took under a minute per disk. An unforeseen advantage was that multiple disks could be cloned at the same time (provided they could all be connected to the same machine simultaneously), which made the process faster. After all the disks had the software installed, they were installed in the laptops, which perceptibly improved the experience of using them. Table 2.1 shows a comparison between using a mechanical hard drive and a solid state drive on multiple metrics that are relevant to subjective experience. This also confirmed that a large part of the delay in logging in was due to speed of the hard drives.

The Six-Oh Deploy system could only be used if a hard drive was removed from

a laptop or if the system was run from a live-CD. This was sufficient for the solid state drive upgrade because the drives had not yet been installed in the laptops. For future upgrades, however, this was a limitation. Even if run from a live-CD, which would not require removing the hard drive, it was necessary to start the upgrade on every individual laptop. This system had the design goal of simplifying the upgrade process to the point where a member of staff would only have to run one command once to upgrade every machine.

To achieve this goal, we designed an extension to the Six-Oh Deploy system[7] that upgrades a laptop with a single shell command, and a second system, the Six-Oh Caster system (described in section 2.2) that runs a shell command entered by a user on every lab machine simultaneously. Combining these two systems, we upgrade every laptop by running the Six-Oh Deploy upgrade command simultaneously on all the machines, with the user only writing it once.

The extension to the Six-Oh Deploy system uses GRUB to launch a script from the boot menu. This script exists in a separate partition on the disk, and is a slightly modified version of the *clone.sh* script we described above. This version, instead of copying files and a partition description from a local copy, copies them from a known server.[8], and guarantees that the partition in which it exists is not modified. Finally, the script creates the GRUB entry that launches Six-Oh Deploy, and also creates a script in the installed operating system that reboots into that GRUB entry. With Six-Oh Caster, that script can be run on every laptop simultaneously, with the user only entering it once.

2.2 Six-Oh Caster

The Six-Oh Caster system broadcasts a shell command entered by a user in a trusted machine, to every laptop in the lab. This shell command is then run on every laptop

[7] This extension is still under development due to time constraints, although its design is complete. It is our plan to complete it over the Summer following the completion of this thesis.

[8] An SSL connection, protected with certificates is used to prevent man in the middle attacks. This also allows the script to easily self-update.

with administrative privileges, allowing staff to perform upgrades and other maintenance tasks without having to repeat the task on every machine. This system consists of two scripts: *client.py* (included in appendix C.5) and *server.py* (included in appendix C.6).

The *client.py* script runs on the laptops and is launched when the laptop boots. The script first connects to a trusted server through an SSL connection.[9] Then it checks for a file in a specific location (in this case */etc/6.01/loc*) which identifies the laptop. If that identity exists, the script sends it to the server, otherwise it goes unnamed from the server's perspective. After connecting, the *client.py* script waits for data from the server, and forwards it to the input of a concurrently running *bash* process. The output of that *bash* process is redirected back to the server for logging/processing purposes. In case of network failure, the client continuously tries to reconnect to the server.[10]

The *server.py* script is launched from the trusted server when an administrative action needs to be executed. The script listens for connections on an SSL socket, and when a client connects, it adds them to a list of message receivers. When a client sends their name, the server associates their socket with that name. If the server is started with the logging option, it creates a file in the directory assigned to logging with the client's name, and logs all the responses from that client in that file. The logs directory can also be used to see which named laptops are online.[11] Once all the laptops are connected, the user can type a command in the standard input of the server process, and that command is sent to all connected machines and executed on each. The responses from the named machines are written to their respective log file.

The first uses of this system were to reboot all the machines concurrently, and to run update scripts. This system will eventually be used to run the Six-Oh Deploy

[9]The certificate verification process prevents an attacker from masking as the trusted server. It also makes sure that the connection is encrypted.

[10]It is setup with an aggressive *TCP_KEEPALIVE* option so that it can detect early if the network connection goes down, and try to reconnect from there. Otherwise, if the network connection went down, and then came back up, it would take a long time to reconnect, since the socket connection would no longer be open.

[11]These names can be used by a user with access to the server to determine which laptops are turned on and connected to the network. This facilitates diagnosis of failures on specific laptops.

script on every machine, allowing for an easy upgrade. The objective of these two systems together is for a user to be able to test an operation or system image on a single laptop, and then easily apply it to every laptop in the lab. After the completion of the Six-Oh Deploy system, it will be possible to run a single command that upgrades every machine. With the use of the Six-Oh Caster system, that command can be run concurrently on every machine, with minimal user work.

2.3 Upgrading Wireless Cards

One of the major issues with the 6.01 laptops was their unreliable network connection. This would cause AFS and consequentially other software to block, and was disruptive when students were working. Using the laptops' stock wireless cards, it was difficult to maintain a stable connection for more than a few minutes in the lab. Their Ethernet ports had become worn out by use, unable to keep an Ethernet cable connected, and the USB adapter solution only provided temporary relief, and required more maintenance. It was also highly desirable to increase the mobility of the laptops to increase the scope of the projects students are able to do with the robots. The most desirable solution to this problem was to understand what caused the wireless connection to fail, and fix that specific issue.

There was evidence that it was be possible to keep a stable wireless connection in the lab. Specifically, at any time during lab hours, most students' phones and many personal laptops were connected to the network without any obvious issue. This was only circumstantial evidence, however, since for most use cases it is not noticeable if the Internet intermittently disconnects, unlike with AFS, where an Internet connection that disconnects intermittently causes many applications to block for a long period of time. This, however, supported the case that a solution might exist. A key insight into that solution was the realization that, while the laptops stock wireless cards were only able to connect to the 2.4GHz band, most modern personal laptops and smartphones are able to connect to the 5GHz band. A scan of the available wireless networks also showed that there were a large number of unsanctioned wireless

networks on the 2.4GHz band, potentially causing a large amount of interference.

Following that hypothesis, we conducted an experiment, upgrading the stock wireless cards to new wireless cards supporting 5GHz. This experiment, however, turned out to not be as straightforward as expected.[12] It is standard procedure for Lenovo to program a whitelist of supported wireless cards into their BIOSes. This meant that it would be necessary to either somehow bypass the BIOS whitelist, or find an appropriate wireless card that was supported by the laptops in use (ThinkPad T410). We found supported wireless cards, but there was not enough quantity to install them on all the laptops if the experiment was a success. We attempted to contact Lenovo/IBM to procure more stock, but we did not receive a response, so we decided to find a method to bypass the BIOS protection.[13]

2.3.1 Installing the Coreboot BIOS

We found methods to remove the whitelist from the original Lenovo BIOS, although they lacked documentation, and did not work consistently[4][10]. There were also binaries for several Lenovo/IBM BIOSes, reportedly only changed so that the whitelist was removed, and Internet forums existed for requesting BIOSes with the whitelist removed[11]. One of the big disadvantages with this approach was that it required either modifying code directly on the BIOS binaries without a clear knowledge of what the consequences of such modifications would be, or trusting that the provided binaries had not been modified in other ways. This was not acceptable, as students would be using the laptops and handling their sensitive data on them, which an unsafe BIOS could potentially access. We decided on a different option: installing the open source Coreboot BIOS[12]. This approach had the advantage that course staff could audit the code if necessary, and that the software was under active development. It had the caveat, however, that it had never been successfully installed on a ThinkPad T410.

[12]The original expectation was that it would consist of acquiring a new wireless card with support for the 5GHz band, and installing it, and the required drivers.

[13]Lenovo/IBM eventually responded, but at that time, the experiment had been done, deemed a success, and all the laptops had their wireless cards upgraded.

The Coreboot BIOS had, however, been successfully installed in a laptop with the same chipset as the ThinkPad T410[1]. One of the main challenges with following the documented procedure was that there was no known way to install the Coreboot BIOS from within an operating system. The mechanism used by Lenovo for BIOS upgrades is not clear, but it seems that they that they lock writing to the BootBlock region of the ROM[14], and on a BIOS upgrade, the new BIOS code is written to other regions of the ROM; on startup, the old BIOS overwrites itself with the newly written BIOS. It is not clear if the BIOS also performs signature checking to prevent non-Lenovo authorized BIOS from getting installed.[15]

Another option was to disassemble the laptop and directly write to the ROM with a hardware connection. This was the method suggested in the original article describing how to install CoreBoot on the ThinkPad X201 laptop. Specifically, the ROM on these machines was usually a MX25L6445E[9] or compatible chip, on which it was possible to write using the SPI protocol. To interface with the chips, we could use an Integrated Circuit Clip, or directly solder wires to the chip. Initial research indicated that the clip could not be used due to a specific type of ROM chip being used[8]. An inspection of the laptops, however, revealed that they had a different type of chip (SOIC-8 chips, seen in figure D-2) which were compatible with clips that could be easily obtained.[16]

We could, then, write to the ROM using a clip and an SPI programmer. For this procedure, we followed the guide for installing Libreboot[5] (a free and open source BIOS based on Coreboot) on a ThinkPad X200 laptop[16]. We decided to follow that guide because the ROM chips on the X200 and T410 laptops are compatible in terms of connections and programming procedure. The first challenge was to create

[14]This is the region that has the code that the CPU runs on startup.

[15]The fact that there are successful modifications to the Lenovo BIOS to remove its whitelist suggests that such checking either does not happen, or happens at a limited scope.

[16]Soldering was too risky: even if successful on one machine, it would be too time-intensive, and it was likely to damage the other machines. Initially, we believed that it was not possible to use a clip to write to the T410 ROM due to a different type of ROM chip connection (TSOP). Specifically, while it is easy and cheap to obtain a clip for interfacing with a SOIC-8 chip, such as the one in figure D-1, the same is not true for a TSOP chip. Soldering directly on the pins seemed the only option available to us. We decided to confirm this, and disassembled a laptop, and found that it had a SOIC-8 chip instead.

a working image of Coreboot for the laptop. The most recent version of Coreboot did not work with the ThinkPad T410. It would display characters but block before booting into a hard drive. This confirmed that the procedure for writing to the ROM with the clip worked correctly, especially as it was possible to backup and restore the factory image through that same procedure.

Instead of patching the latest Coreboot image, the version of Coreboot used in the guide for the ThinkPad X201 was chosen. The challenge in using that version was that it required a VGA BIOS to initialize the graphics.[17] The process of extracting the VGA BIOS was a lot more involved than expected, as following the existing guide[7] did not work. To facilitate this process, a script *vga_extract.sh* (presented in section C.2) was created. This script takes the location of a BIOS upgrade ISO (which can be obtained from Lenovo) and places all the possible VGA BIOS files in a directory specified by the user[18]. These files can be inspected to find the VGA BIOS corresponding to the graphics adapter in the laptop, which can then be used to build a Coreboot image (following the instructions in the guide for the ThinkPad X201) that works with the ThinkPad T410.

After installing Correboot on one laptop, we tested it with a new wireless card supporting the 5GHz band. The Atheros AR5B22 chipset was chosen specifically due to its support for the 5GHz band, and for the fact that it was supported by free software drivers. This initial test was highly successful: the machine was usable, and the students and staff who chose to use it did not seem to notice that it was not connected to Ethernet.

With this initial successful test we decided to increase the sample size of the experiment. The 6.01 lab has about 80 machines. It was possible that if all 80 machines were connected, we would lose the stable wireless connection observed on the first test. To mitigate this risk, we decided to do a gradual deployment of Coreboot and

[17]The newer version includes an open source VGA BIOS that works with this chipset, which is why we got a picture on the screen, despite the laptop not booting up. In the image without a VGA BIOS it would boot—we confirmed this with an external display—but it would not display a picture on the laptop display.

[18]The ThinkPad T410 laptops could be configured with different graphics adapters, which is why the BIOS image provided different VGA BIOSes.

the wireless cards. We tried to create a method by which Coreboot could be installed without disassembling the laptops (this attempt is described in Appendix A), but this was unsuccessful. We decided instead to install Coreboot via the hardware interface method described above. This process was lengthy because it required disassembling the entire laptop, and removing the motherboard to be able to reach the ROM chip. To make this process as efficient as possible, a fast ThinkPad T410 maintenance guide was created, and staff members were recruited to aid in the process (the SPI flashing part of the procedure was fast; the bottleneck was on how quickly the machines could be disassembled and reassembled).

2.3.2 Results of the Wireless Upgrade

The first phase of this experiment was to have 10 laptops running Coreboot with a 5GHz card. This was followed by a second phase with 30 more laptops upgraded. This second phase was also successful, so we upgraded the remaining machines in a final third phase. Each of these phases showed promising results: students did not have issues logging in or using the machines, except for two isolated issues[19] during the interim upgrade period.[20]

Overall, the upgrade of the wireless cards was a success. For the first time since the course began, all the laptops were connected to the wireless network without stability issues. We empirically evaluated these observations with a simple test: 5 laptops concurrently sent one ICMP packet per second for one hour to the same server[21], using the ping tool. One of the machines was connected to a stable Ethernet connection, three of the other machines were using the 5GHz wireless card[22], and the

[19] These happened during a lab section and were quickly resolved, although the underlying cause was not found. It could have been a network failure, or a password entered incorrectly.

[20] A period of 3 weeks, as more machines with new wireless cards got deployed.

[21] This was the most important server that these machines had to reach, as it hosts the online tutor the students use. Sending packets to this server also meant that outside network issues would not affect the test, as the server is in the same room as the laptops, and the wireless switches are connected to the same switches as the server.

[22] Most laptops in the lab were turned on at this time, which meant there were about 70 other laptops on the same network as these 3 machines, thus this was a good approximation of what the students faced during lab hours.

Laptop	Packet Loss Fraction	Min RTT	Max RTT	Avg RTT	RTT σ
5GHz #1	0	0.9	48.3	1.6	1.3
5GHz #2	0	0.8	68.1	1.6	2.3
5GHz #3	0	0.8	57.3	1.5	2.0
Ethernet	0	0.1	0.7	0.3	0.1
2.4GHz	0.007	1.5	214.4	8.4	15.4

Table 2.2: Results of the Wireless Card Experiment, Packet Loss and Round Trip Time Metrics (in milliseconds)

remaining machine was using the stock 2.4GHz wireless card.[23] The results of this test can be found in Table 2.2.

One of the most relevant results in the table is the packet loss amount. It is worth stressing that in this case, lost packets most likely correspond to network connection drops. Every connection drop can cause a student to lose several minutes of work due to AFS waiting for some time before trying to reconnect.[24] Course staff can intervene and make this process faster, but it still means a few minutes of lost work for a student. The 2.4GHz wireless card had a few packets lost, which made it unusable for AFS. The round trip time is also very high for the 2.4GHz cards compared to the 5GHz cards, which, from subjective experience, made using AFS, which synchronizes on every close operation, a very slow experience. In practice, when all the machines were on the 2.4GHz band, the interference was a perceptible problem: network drops were much more common, which made wireless effectively unusable for AFS.

Comparing the 5GHz cards to the Ethernet shows that Ethernet is preferable, however. This was expected, but it is also because of the stability of the connection in this case. In practice, Ethernet is very reliable as long as the laptop is kept mostly stationary. While there is a very slightly perceptible difference between using the

[23]Since this was the only machine using the 2.4GHz band, it was not a good approximation of what the students had to deal with before the upgrade, but unfortunately this test was not designed or run before the upgrade to the new wireless cards.

[24]When the wireless network was disconnected, it would quickly reconnect. AFS, however, would permanently lose the connection it had to the server. AFS is, however, designed to try and use its current connection for a user-specified amount of time before trying to create a new connection. This is due to the fact that the workstations it was designed to work in were more likely to fail in ways that did not eliminate the existing connection.

5GHz wireless cards and Ethernet with AFS[25], it does not affect the students' experience in practice, and it improves the laptops' mobility,, allowing for more educational opportunities.

The hardware upgrades proved to be highly valuable to the lab by increasing the service life and mobility of the laptops. The development of Six-Oh Caster and Six-Oh Deploy also means that we will be able to deploy and apply upgrades to the machines in a much more efficient manner. The hardware upgrades also slightly improved the experience with Debathena[2] due to the reduced login time and stabler Internet connection. Still, Debathena is not the best fit for the lab. We still expect network connection failures, even if they are a much rarer occurrence. These will cause AFS to block, and consequently any software using the student's home directory. It should be possible for the students to keep using the laptops even if the network is disconnected, as for many assignments the Internet is not essential. Thus, we designed the Six-Oh Transfer and Six-Oh Authentication Program systems to replace Debathena.

[25]This is mainly visible if you press the *ENTER* on a terminal. The time it takes for a prompt to display is higher than on Ethernet, particularly visible if the key is kept pressed.

Chapter 3

Replacing Debathena

3.1 Issues with Debathena

One of the major issues with Debathena[2] in the lab was the use of AFS[14]: this file system worked well in the settings it was designed for, but it did not fit the use case presented in the 6.01 lab. Specifically, AFS synchronizes a file on every close operation. Assuming a stable network connection, this does not present an issue, particularly due to the high bandwidth, low latency networks that campus workstations are connected to. The workstation that AFS is designed to be installed on have low risk of an unstable network condition, since they are fixed machines connected to the network via Ethernet. The 6.01 lab's computers, however, have a much higher risk of network instability.

In a computer lab equipped with laptops, the network is not stable. Despite multiple attempts at improving the wireless signal quality within the lab room, the laptops could not hold a persistent, stable wireless connection. An Ethernet connection was often stable enough, but moving the laptops around caused both their Ethernet ports and cables to get worn over time. The wear of the ports and cables made it very easy to accidentally disconnect a cable, even with just a small amount of movement. A temporary solution already in place was to use USB Ethernet adapters to replace the worn down ports of the laptops, which allowed for a temporary relief of the issue. There were cases, however, where a cable or Ethernet adapter was worn down,

causing the Ethernet cable to disconnect with movement. These adapters were also more prone to failure when compared to the Ethernet ports of the laptops, requiring occasional replacement. This constituted a continuous investment in maintenance and a higher frequency of interruptions to the students' work than if we could use the wireless network. Even with the wireless card upgrades, however, we expected and observed rare connection failures. The frequency of failures was also expected to increase, as devices in surrounding labs and buildings would start using the 5GHz band, and causing interference.

The difficulty of maintaining a stable network connection was highly disruptive due to AFS blocking when trying to synchronize files. In particular, this would cause the code editor *IDLE* and the web browser *Firefox* to block, which are necessary for the students to work and learn in lab, and thus should block as infrequently as possible.[1] Despite these issues, Debathena was preferred over other solutions due to historical reasons. These reasons, however, were less relevant when we decided to replace it with a new system.[2].

The decision to replace Debathena with a new system for the course meant evaluating what functionality Debathena provided that was important, what was desirable, and if there were desirable features that Debathena did not provide. The most important features that Debathena provided were file synchronization via AFS and common authentication across machines via Kerberos[17]. It was also desirable to maintain the feature that the authentication to the online tutor is shared with the authentication to the machines (via certificates or some other means). Furthermore, Debathena did not provide an easy way to deploy code files that the students used in their work. With Debathena, the students ran a script to download the code to their home directory, or downloaded it from the online tutor website. With a new file synchronization mechanism, it was desirable that it would allow for an easier deployment of code files. Another desirable feature that Debathena did not provide was easy user management. The Kerberos authentication system is out of the control of the 6.01 staff members,

[1]IDLE is the main editor the students use for writing code, and the browser is essential for interacting with the online tutor.

[2]A more detailed description can be found in Appendix B

so we could not add users easily. This was of particular importance because other educational programs, with non-MIT students, used the lab on occasion. Moving away from Kerberos could provide the staff with a means to easily add users to the laptops.

The most important component to replace was AFS, which caused the majority of the issues. Existing options for file synchronization were investigated, but none were deemed adequate. Many were not freely available (such as Dropbox, or BitTorrent Sync), which made their use not desirable, while others introduced features in their design that added complexity, but were not useful to the course. The implementation of these systems was also not ideal in many cases. One example was the ORI filesystem[15], which was almost ideal for the course. The available implementation, however required a complex system of trust to exist (every machine that was sharing files needed SSH keys to access every other machine), and did not work reliably. Thus, we decided to develop a new system with design goals and constraints tailored to our lab, the Six-Oh File Transfer system.

3.2 The Six-Oh File Transfer System

Due to time constraints, this system was never fully implemented. Its design, however, has been completed, and is presented in this section. It is our plan to complete the implementation in the Summer following the completion of this thesis.

The move away from AFS meant that the course (rather than the University) would be hosting the student's files. To protect the students' privacy, it was essential that these files were not visible to the staff. To achieve this constraint, we encrypted the files that are stored in the server, using the authentication passphrase as a key to decrypt them. When a student uses a laptop, they see the decrypted files, but only the encrypted copies are stored on the server.

This constraint, however, meant that whatever service held the encrypted files had to be involved when there was a passphrase change. Otherwise, if a student logged in after changing their passphrase, the files would still be encrypted with the

previous passphrase, causing conflicts. To achieve this, a new authentication system would have to be developed to replace Kerberos. An alternative solution would have been to prompt the user on login that, while they authenticated successfully, the decryption step failed, and ask them to enter both their previous and new passphrase to re-encrypt the files. While both solutions were similarly complex to implement, the implementation of a new authentication service was preferable for a few reasons. First, it would enable easy creation of new users. Second, its implementation could be greatly simplified if we used the existing password based authentication from CAT-SOOP[13], the online tutor used by the class. Finally, if we used the CAT-SOOP password authentication system, we would no longer need to use certificates for authentication, which had historically caused many issues for the students (see appendix B for more details). This system, the Six-Oh Authentication Program, is described in section 3.3 in detail.

One issue with encryption, however, was how to implement the desirable feature of easily deploying code to the machines.[3] To achieve this goal we used public key encryption. Every student would have a private and public key pair. This pair would be created upon student registration, and the private key would be encrypted by their passphrase. Using public key encryption, staff members would be able to deploy code files to the students' directories, using their public key, but would not be able to read their contents. In practice public key encryption is not suitable for files. Encrypting an entire file with a public key algorithm such as RSA[18] takes a long time, and would cause the file system to slow down considerably. There are, however, encryption mechanisms suitable for file encryption. These are, however, symmetric encryption mechanisms that rely on a single key. We can combine a public key encryption mechanism with a symmetric one, however, by following the following strategy:

- Generate a random key for the symmetric algorithm.

[3]With Debathena, the students ran a shell command that downloaded the code files to their desktop. While we could continue using this solution, we designed Six-Oh File Transfer with the design goal of enabling an automatic solution for code deployment.

- Encrypt the file with the symmetric algorithm and the generated key.
- Encrypt the generated key with the public key algorithm.
- Place the encrypted key in the header of the file.

Decrypting the file now consists of decrypting the generated key at the header of the file, and using it to decrypt the remainder of the file.

Another challenge with encryption is handling the case of a lost password. Under the scheme described above, a student would lose access to their files entirely, since it is impossible to recover the information without the passphrase. A possible solution to this would be to have a master key accessible by course staff. This key could be made secure by encrypting it with multiple keys, each belonging to a different member of the staff, thus requiring multiple members of the staff to come together to recover a student's files. This solution, however, adds a large amount of complexity to the system, and makes the students' files less secure, as now staff would be able to access them. Due to these concerns, and the fact that losing one's files does not come at a high cost to a student (their grade is most likely not affected, as CAT-SOOP keeps track of all their work), we did not tackle this challenge.

It was also important that the Six-Oh File Transfer system not block when the network failed, with the acceptable tradeoff that eventual consistency was achieved across replicas of the user's files. To satisfy all these requirements, we designed the the system as follows:

When a user logs in, a request is made to a central server for their files. The user's home directory is then downloaded from the server to the machine.[4] Using the authentication token from the user, these files are decrypted and placed in a separate location from the encrypted copies, where they are only accessible by root

[4]Any file synchronization strategy should work here. For this particular implementation, we plan to use rsync, particularly because it simplifies caching—a machine can keep a local copy of the encrypted files, which are updated with minimal traffic on login; a better mechanism might be to build a lazy file synchronizing system that immediately returns, but keeps synchronizing in the background, lazily blocking when requests come for files that have not been synchronized yet.

and the user.[5] Ideally, this location is destroyed when the machine turns off, or made inaccessible if the operating system is not on.[6]

An encrypted write ahead log (WAL) is kept on disk to improve the performance of the system[7] and allow for recovery in case of failures, such as system crashes and network issues. In case of a crash, the system would try synchronize the files on startup. Any actions in the WAL that had not been applied to the files, however, would not be available to the system, as it would no longer have the encryption key accessible. Therefore, the WAL is also synchronized in this case. On user login, if there are any synchronized WAL actions, these get applied, the modified files are sent to the server, and the WAL is purged.

A custom FUSE filesystem is then mounted in the home directory of the user. Every request that does not modify the files is passed through to the location of the non-encrypted files. Every request that modifies a file is first written as an encrypted WAL, a message is sent through a Unix socket advertising which file was changed, and finally the change passes through to the decrypted files. When the filesystem is mounted, a separate process is started in the background. This process listens on the same Unix socket that the FUSE filesystem advertises on. If files are changed, the process periodically encrypts the modified files and sends them to the server. While encrypting the file, operations that may modify it are blocked by a file lock to keep consistency. The WAL entries describing the changes between the previous encrypted version and the new one are then purged. To handle the failure of a system crash before purging the WAL, the headers of the encrypted files are augmented: the files are versioned, and each WAL entry increments their version number. Thus, from the WAL, it is possible to know which version a file would be in when applying a specific entry. Even if a WAL entry remains after the file has been updated, it will not get applied repeated times because it has a smaller version number. This design does

[5] The root user is trusted in this system, as it would have many other ways of obtaining the user's files otherwise, for example by modifying the login scripts, and logging their authentication tokens.

[6] Ways to do this would be to put the files in a ram disk, provided there is enough system memory, or encrypting the disk, and only allowing staff to turn on/reboot the machines.

[7] Without the WAL, every file modification would require a re-encryption of the modified file. With the WAL, only the modification needs to get encrypted.

not guarantee that the system will never enter an invalid state, however. Specifically, when encrypting the new version of a file, if the operation used to replace the previous version can fail in an invalid intermediate state, the whole system enters an invalid state. It is thus important that this operation has a low probability of having invalid intermediate states, so that either the original file, or the new one are the only possible results in case of failure.

On logout, all the pending operations are applied and synchronized, and the unencrypted copy is deleted (the encrypted copy is kept as a cache). A separate process, launched on system startup, manages the cached copies, deleting them if disk space is required. Further, if a failure happens that results in a reboot, a process is started on startup that synchronizes every cached copy with the server, to guarantee that no changes are lost.

The Six-Oh File Transfer system achieves its design goals by separating operations on the file system from the synchronization operation. The FUSE file system does not block if the network is disconnected, it just keeps writing new WAL entries. The process in charge of synchronizing changes with the server only blocks in the synchronization step, not on the encryption step. The encryption step would be the only place that could cause the FUSE filesystem to also block. When the laptop reconnects to the network, the process in charge of synchronization will guarantee that the files are correctly synchronized. In case of failures, there are multiple safeguards in place to guarantee that the state of the filesystem on the local machine is eventually consistent with the state on the server.

3.3 The Six-Oh Authentication Program

With Debathena, the students were using Kebreros to authenticate and obtain certificates that authenticated them with CAT-SOOP. The main goal of the authentication system was that it would transparently allow for the students' files to be re-encrypted. Secondary goals were transparent authentication with CAT-SOOP on login and easy creation of new users by staff members. The CAT-SOOP system can provide au-

thentication via password and can easily be extended to perform extra actions on a password change.[8] It was also very easy to add new users to CAT-SOOP. Due to these advantages, we decided to implement the Six-Oh Authentication Program by using the existing CAT-SOOP password authentication mechanism, and modifying CAT-SOOP to re-encrypt the students' files on a password change.

This system was developed as a PAM[21] script that would talk to CAT-SOOP. A desirable feature would be to have this script not only authenticate the user to the laptop, but also login the user to CAT-SOOP on their browser. When a student uses password authentication on CAT-SOOP, their browser sends a POST request through HTTPS with the student's username and password. The server replies with either an error page, or a success page, and a session ID cookie. That session ID allows the user to access CAT-SOOP resources without having to authenticate.

The *catsoop_login.py* script (presented in section C.7) communicates with the CAT-SOOP server via POST and tries to login. The exit code of this script indicates whether login was successful or not. If it was, its output is the session ID. The *firefox_sid_cookies.py* script (presented in section C.8) takes a session ID and home directory location. If that user has a firefox profile, the script adds the session ID as a cookie. Finally, the *catsoop_login_pam.sh* script (presented in section C.9) ties those two scripts together by using the first to determine whether the authentication was successful, and the second to add the session ID to that user's firefox session. It also creates a home directory for the user if one does not already exist[9] and initializes a default *Firefox* profile before running the *firefox_sid_cookies.py* script. In case of failure, the script tries to diagnose what could have failed, and displays helpful error messages.

The Six-Oh Authentication Program thus allows users to use their CAT-SOOP credentials to login to a laptop, and it also logs them in to the CAT-SOOP online tutor. This system not only allows for the implementation of the Six-Oh File Transfer

[8]This system is free software, and was originally developed by the Adam Hartz, who ran the 6.01 course; the familiarity of the course staff with CAT-SOOP motivated the decision to use its authentication mechanism and extend it.

[9]Note that this portion of the code will be replaced with the Six-Oh File Transfer system once it is implemented.

system, but allows the course staff to easily add students to the system.

Chapter 4

Conclusions and Future Work

We presented two sets of work that all contributed to making a better experience for students in our lab, with fewer interruptions: the evaluation and deployment of Hardware upgrades, aided by two new software systems (part of Six-Oh Deploy, and the Six-Oh Caster system), which will facilitate future Software and Hardware upgrades; and the development of a replacement for Debathena[2], via the the Six-Oh File Transfer system design, and the Six-Oh Authentication Program. Together, these systems form the S4 Infrastructure System, which when complete provides a set of tools for using and managing a computer lab equipped with laptops.

The completed portions of this work are already adding value: the laptops have a higher useful service life, and mobility, allowing for new learning experiences, and the management of the software in the machines is easier for staff. We also plan to deploy the Six-Oh Authentication Program in the semester following the completion of this thesis, facilitating a seamless authentication with the tutor, and facilitating other classes happening in the lab.

There is still a lot of work to be done: it is necessary to complete the Six-Oh Deploy and Six-Oh File Transfer systems first. Once complete, there are other areas of research that can branch from the work in this thesis.

Installing Coreboot on machines is still an involved process. It takes about one hour per person per laptop to install Coreboot. It would be very valuable to have a way to install Coreboot in the ROM from an operating system. Some research work

shows that this should be possible by modifying modules in the BIOS [20] to unlock the BootBlock, which would enable a direct write of the Coreboot BIOS. It would also be of value to have this research happen for more modern devices.

One disadvantage of the current Six-Oh File Transfer system design is that it counts on a centralized system. That is also true of the other tools in use in the 6.01 course (for example CAT-SOOP, and the other tools developed in this thesis). That means that the 6.01 server is a single point of failure. A valuable research direction would be to find ways to make all these tools distributed. An interesting application would be to leverage the fact that there are a lot of laptops in the lab, which most of the time are underused, since lab is closed outside of certain scheduled times.

The Six-Oh File Transfer system can also be the base for other research projects. Specifically, which synchronization mechanism to use, and supporting cooperation between different users. A bidirectional synchronization mechanism could allow more than one machine modifying the same files, particularly if there is some way to handle conflicts. Leveraging such a mechanism, and the fact that public key encryption could be used to allow multiple users to decrypt the same files, one could extend the Six-Oh File Transfer system to support directories shared by multiple users, but only visible to those users. Another direction for further research in the file synchronization mechanism is supporting lazy serving of files. As it stands, the Six-Oh File Transfer system might require long login times, if the user's synchronized files are large and not cached on the machine. A better synchronization system would have the files immediately available as they're synchronizing, and lazily serve them as needed. Thus, the login time would only be as long as needed to synchronize the files read on login.

It is our hope that the S4 Infrastructure System is completed soon after this thesis is submitted, and keeps providing value to the 6.01 course and possibly other courses with similar labs. These systems, and the challenges and tradeoffs faced also suggest interesting directions for future research extending them.

Appendix A

Attempted Software Installation of Coreboot

When trying to install Coreboot, we researched ways to install it from within an operating system. The method we attempted consisted of modifying the ROM flashing scripts in the BIOS upgrade images from Lenovo. Specifically, by calling the *phlash16.exe* script, and providing it modified BIOS images (to include Coreboot instead of a Lenovo BIOS). This did not work, and made the computer unusable. A post-mortem analysis of the resulting ROM image showed that, while part of Coreboot was written to the ROM, part of the ROM was unchanged. This portion of the ROM corresponded to the BootBlock region, which the X201 guide[1] theorized was blocked by software. Several experiments were done with this, specifically changing the location of the Coreboot binaries within the ROM, but none managed to install Coreboot. It is unknown if this method could be used at all, because the Lenovo BIOS might implement a limited signature verification so only authorized BIOS images get loaded.

Further research suggested that the only reason it was possible to make the system non-functional was that, while the main Lenovo BIOS boot code was protected, and potentially signature checked, it loads other modules that were not protected, and could be modified[20]. We posit that it should be possible to inject code into one the BIOS modules that unlocks the BootBlock region of the ROM, thus allowing a

tool such as *Flashrom* to be used to write Coreboot[12] to the ROM from within an Operating System. Due to time constraints and inexperience with reverse engineering of software, this approach was not explored further.

Appendix B

Historical Reasons for Choosing Debathena

There was one historical reason why AFS[14] was preferred over other solutions: it would synchronize the students' files with workstations spread out across campus. That allowed students to easily retrieve their files and work on the course's projects without having to be in lab, instead going to an "Athena Cluster" anywhere on campus. This advantage is no longer as relevant as it used to be. It was observed that students don't rely as much on "Athena Clusters" as at the time of the course inauguration, instead preferring to use their personal laptops. This actually made it more of a hindrance for a student to retrieve their files, as they would have to install extra software to access OpenAFS, or get their files over SSH[1]. It was also uncertain whether the "Athena Clusters" were going to exist for a lot longer[19], making this advantage even less relevant. This advantage also came with one caveat: it wasn't always possible to have the same versions of software in common with the campus workstations, and the files that each different version stored were sometimes incompatible, causing software to behave incorrectly for the students using those workstations (one example of this was Firefox's locking mechanism preventing students from opening new

[1]This might not require extra software, depending on the operating system—both Linux and Mac OS come with software to access files over SSH usually pre-installed, although Windows does not—, but requires a familiarity with the necessary software that many students taking 6.01 did not have initially and which is out of the scope of what the course was trying to teach

sessions).

Despite the issues with AFS, Debathena provided the course with one major advantage: using Kerberos[17] credentials for authentication. MIT uses Kerberos as a central authentication mechanism for many of its services. In particular, students can get client certificates for their browsers that authenticate them with many services at the school. The course used Kerberos authentication so that the users did not have to manage user accounts for the online tutor[13]: authentication was done via certificates, which would be available across different labs via AFS. Certificate authentication, however, did not work as well as expected.

For reasons that were never completely understood, the Firefox's certificate store became corrupted, not only preventing students from being able to authenticate with the online tutor, but also preventing them from installing new certificates. The initial solution to this problem was to delete their entire Firefox profile, erasing their browsing history and other settings, but ultimately solving the problem. Later, it was found that there was a single file *cert8.db* in the profile directory that contains the certificates. Deleting that file seemed to fix the issue, although it required students to re-install certificates. Eventually, a script *certfix* (presented in section C.1) was written and deployed to the lab's machines as a way for staff to quickly fix the issue.[2]

An initial theory as to why this file corruption happened was that it could be due to the course using custom Firefox profile locations on the lab laptops. This was done initially to minimize interference with the profiles used by the campus workstations, due to the different Firefox versions in use. An experimental fix was done that consisted of simply using the default Firefox profile. This did not solve the issue, however, and it exposed other issues: Firefox would now sometimes be unable to generate a key for certificate creation, even after running the aforementioned *certfix*. The best theory for why that happened was due to version mismatches with campus workstations' Firefox installations.

An interesting property of these certificate store corruption cases was that they

[2]This script found the profile directory and deleted the cert8.db file from there; it was written by the author of this thesis in cooperation with Geronimo Mirano from the 6.01 staff

tended to happen to certain students with a disproportionate probability. This led to the theory that there could be laptop usage patterns that increased the odds of having the certificate store corrupted. A hypothetical such pattern would be to consistently leave the browser window open when logging out, or logging out via a hard shutdown without closing the browser. It is possible that, due to the rarity of client certificate usage in browsers, this functionality isn't well tested and that Firefox might leave the certificate store in invalid states when not properly shutdown. It is possible that the AFS mechanism for synchronization (only synchronize when a file descriptor is closed) couple with an unreliable network connection made that more likely. These hypotheses, however, were not confirmed, as, despite repeated attempts, a test case that could consistently (or at least with high probability) reproduce the issue was not found.

Appendix C

Code

C.1 certfix

```
#!/usr/bin/python
import sys
import os


def main(argv):
    filepath = os.environ['HOME'] + '/.mozilla/firefox/profiles.ini'
    f = open(filepath)
    lines = f.readlines()
    f.close()
    try:
        indx = lines.index("Default=1\n")
        pathline = lines[indx-1]
        path = pathline[5:-1]
    except ValueError:
        for l in lines:
            try:
                n, v = l.split('=')
                if n == 'Path':
                    path = v[:-1]
                    break
```

```
22              except:
23                  pass
24
25      cert_file_path = os.path.dirname(filepath) + "/" + path + "/cert8.db
        "
26      try:
27          os.remove(cert_file_path)
28          print("Removing %s to fix firefox." % cert_file_path)
29      except:
30          pass
31
32  if __name__ == "__main__":
33      main(sys.argv)
```

code/certfix

C.2 vga_extract.sh

```
#!/bin/bash
# Requires: 7z + geteltorito, and tools to compile bios_extract and
    phnxsplit
#
INITIAL_DIR=`pwd`
ISO_FILE=$1
OUT_DIR=$2
if [ ! -e "$ISO_FILE" ] || [ ! -d "$OUT_DIR" ]
then
    echo "Usage: $0 lenovo_bios_disk.iso output_directory" && exit 1
fi

TMP_DIR=`mktemp -d`
function clean {
    echo DONE
    rm -rf $TMP_DIR
}
trap clean EXIT
CURRENT=`pwd`
cp $ISO_FILE $TMP_DIR/disk.iso
cd $TMP_DIR

# Extract files from ISO file, so we can get the rom
echo "Extracting files"
geteltorito -o boot.img disk.iso || exit 1
rm disk.iso
dd if=boot.img of=boot2.img skip=16384 iflag=skip_bytes || exit 1
rm boot.img
mkdir disk
cd disk
7z x ../boot2.img || exit 1
rm ../boot2.img
BIOS_FILE=`find . -name *.FL1`

```

```
34 # Decompress BIOS
35 echo "Decompressing BOIS"
36 git clone https://github.com/coreboot/bios_extract.git || exit 1
37 cd bios_extract
38 make || exit 1
39 ./bcpvpd ../$BIOS_FILE ../../bios.rom || exit 1
40 cd ../../
41 rm -rf disk || exit 1
42
43 # Split the BIOS into its modules
44 echo "Splitting BIOS into modules"
45 wget http://www.endeer.cz/bios.tools/phnxsplit.zip || exit 1
46 unzip phnxsplit.zip || exit 1
47 rm phnxsplit.zip
48 cd phnxsplit
49 # Fix phnxsplit
50 patch <<EOF
51 --- Makefile  2008-11-01 12:45:40.000000000 -0400
52 +++ ../Makefile.old 2016-05-15 20:51:44.270978708 -0400
53 @@ -4,15 +4,16 @@
54  #
55
56  MAKE = make
57 +GCC = gcc -m32 -march=i686
58
59  phnxsplit:
60 - gcc phnxsplit.c lzint_decode.o phnxfunc.o -s -fpack-struct -o
       phnxsplit
61 + \$(GCC) phnxsplit.c lzint_decode.o phnxfunc.o -s -fpack-struct -o
       phnxsplit
62
63  lzint_decode:
64 - gcc lzint_decode.c -c -o lzint_decode.o -fpack-struct
65 + \$(GCC) lzint_decode.c -c -o lzint_decode.o -fpack-struct
66
67  phnxfunc:
```

```
68 - gcc phnxfunc.c -c -o phnxfunc.o -fpack-struct
69 + \$(GCC) phnxfunc.c -c -o phnxfunc.o -fpack-struct
70
71  clean:
72   rm -f *.o
73 EOF
74 make rebuild
75 cd ..
76 mkdir DUMP
77 cd DUMP
78 ../phnxsplit/phnxsplit ../bios.rom
79 cd ..
80 rm -rf phnxsplit
81 rm bios.rom
82
83 # For every file that has a mention of vga rom, put them in the output
84 # directory
85 cd $INITIAL_DIR
86 for f in $TMP_DIR/DUMP/*
87 do
88     if grep -i vga\ bios $f
89     then
90   cp $f $OUT_DIR
91     fi
92 done
```

code/vga_extract.sh

C.3 setup.sh

```
#!/bin/bash
ORIGIN=""
DISK=$1
if (!([ -e $DISK ] && [ "$DISK" != "" ]))
then
    echo "Usage: $0 ORIGIN_DISK"
    exit 1
fi

function clean {
    if [ "$CLONE" != "" ]
    then
        umount $ORIGIN
        rm -rf $ORIGIN
    fi
}
trap clean EXIT

# Copy the files now
ORIGIN=`mktemp -d`
mkdir original
cp -a $ORIGIN/. original/

# Fix files for cloning
rm original/etc/dhcpcd.duid # DHCP Unique ID
rm original/var/lib/dhcpcd5/* # Lease memory
echo > original/etc/udev/rules.d/70-persistent-net.rules # Device names

echo "DONE"
```

code/setup.sh

C.4 clone.sh

```
#!/bin/bash
ORIGIN=""
CLONE=""
DISK=$1
NAME=$2
if (!([ -e $DISK ] && [ "$DISK" != "" ])) || [ -z "$NAME" ]
then
    echo "Usage: $0 DISK NAME"
    exit 1
fi

function clean {
    if [ "$CLONE" != "" ]
    then
        umount $CLONE/dev
        umount $CLONE/proc
        umount $CLONE/sys
        umount $CLONE
        rm -rf $CLONE
    fi
}
trap clean EXIT

# Create the layout
sfdisk ${DISK} < layout.dump
mkfs.ext4 ${DISK}1 -F
mkswap ${DISK}5

# Mount for copying
ORIGIN=original
CLONE=`mktemp -d`
mount ${DISK}1 $CLONE
cp -a $ORIGIN/. $CLONE/

```

```
35 # Setup GRUB + fstab
36 mount --bind /dev $CLONE/dev
37 mount --bind /proc $CLONE/proc
38 mount --bind /sys $CLONE/sys
39 UUID_FIRST=`blkid -s UUID -o value ${DISK}1`
40 UUID_SECOND=`blkid -s UUID -o value ${DISK}5`
41 # Fix fstab with correct UUIDs
42 cat > $CLONE/etc/fstab <<EOF
43 # /etc/fstab: static file system information.
44 #
45 # Use 'blkid' to print the universally unique identifier for a
46 # device; this may be used with UUID= as a more robust way to name
      devices
47 # that works even if disks are added and removed. See fstab(5).
48 #
49 # <file system> <mount point>   <type>  <options>       <dump>  <pass>
50 # / was on /dev/sda1 during installation
51 UUID=$UUID_FIRST /                 ext4    errors=remount-ro 0       1
52 # swap was on /dev/sda5 during installation
53 UUID=$UUID_SECOND none             swap    sw                0       0
54 EOF
55 # Install grub
56 chroot $CLONE <<EOF
57 grub-install $DISK
58 GRUB_DISABLE_OS_PROBER=true update-grub
59 exit
60 EOF
61 # Fix name
62 echo $NAME > $CLONE/etc/6.01/loc
63
64 sync
65
66 echo "DONE"
```

code/clone.sh

C.5 client.py

```
1 #!/usr/bin/python3 -u
2 import time
3 import subprocess
4 import os
5 import ssl
6 import socket
7 import select
8 import sys
9 import getopt
10 import fcntl
11 import json
12
13 NAME = sys.argv[0]
14 argv = sys.argv[1:]
15 DEFAULT_CAFILE = 'rootCA.pem'
16 DEFAULT_PORT = 6601
17
18
19 def usage(out=sys.stderr):
20     global NAME
21     print('''Usage:
22     %s [-p --port port_number] -h,--host host [-a --ca ca_file] [-n --
    name name] [--help]
23     [-p\t--port\tPort number on the server. By default, it is port
    6601.]
24     -h\t--host\tServer's host. Could be name or IP. Required.
25     [-a\t--ca\tLocation of Certificate Authority file. Default: %s.]
26     [-n\t--name\tName to send to server.]
27     [--help\tShow this help text.]
28     ''' % (NAME, DEFAULT_CAFILE))
29
30 SHORT_OPTIONS = 'p:a:h:n:'
31 LONG_OPTIONS = ['port=', 'ca=', 'host=', 'name=', 'help']
32
```

```
33
34 def parse_arguments(argv):
35     try:
36         options, argv = getopt.gnu_getopt(argv, SHORT_OPTIONS,
    LONG_OPTIONS)
37     except getopt.GetoptError:
38         usage()
39         return (argv, None)
40
41     def parse_opt(x):
42         return x if x[-1] != '=' else x[:-1]
43     opt_to_names = {'--' + parse_opt(opt): parse_opt(opt)
44                     for opt in LONG_OPTIONS}
45     long_ind = 0
46     for i in SHORT_OPTIONS:
47         if i == ':':
48             continue
49         opt_to_names['-'+i] = parse_opt(LONG_OPTIONS[long_ind])
50         long_ind += 1
51     server_args = dict((opt_to_names[opt_name], opt)
52                        for opt_name, opt in options)
53
54     return argv, server_args
55
56 RETRY_WAIT = 1
57
58
59 def check_options(server_args):
60     ret = None
61     if 'help' in server_args:
62         return 0
63
64     if 'host' not in server_args:
65         return 2
66
67     if 'port' in server_args:
```

```
68          try:
69              server_args['port'] = int(server_args['port'])
70              if not (0 <= server_args['port'] <= 65535):
71                  print("Invalid port number", file=sys.stderr)
72                  ret = 2
73          except ValueError:
74              print("Port number must be integer", file=sys.stderr)
75              ret = 2
76
77      if ('ca' in server_args and not os.path.isfile(server_args['ca'])) \
78         or not os.path.isfile(DEFAULT_CAFILE):
79          print("Certificate Authority file must exist", file=sys.stderr)
80          ret = 2
81
82      return ret
83
84
85  def main(argv):
86      argv, server_args = parse_arguments(argv)
87      if server_args is None:
88          return 2
89      options_check = check_options(server_args)
90      if options_check is not None:
91          usage(sys.stdout if options_check == 0 else sys.stderr)
92          return options_check
93
94      while True:
95          bash_proc = subprocess.Popen(["/bin/bash"], stdin=subprocess.
    PIPE,
96                                       stdout=subprocess.PIPE, bufsize=0)
97          try:
98              run_client(bash_proc, **server_args)
99          except KeyboardInterrupt as e:
100             print("Terminating")
101             break
102         except Exception as e:
```

```
103                 print("Found exception:\n%s" % str(e))
104             finally:
105                 bash_proc.kill()
106             print("Retrying")
107             time.sleep(RETRY_WAIT)
108         return 0
109
110 # From:
111 # http://stackoverflow.com/questions/287871/print-in-terminal-with-
        colors-using-python
112 SERVER_COLOR = '\033[01;31m'
113 ENDC = '\033[0m'
114
115
116 def run_client(comm, host, port=DEFAULT_PORT, ca=DEFAULT_CAFILE, name=
        None):
117     if name == '':
118         name = None
119     context = ssl.create_default_context(cafile=ca)
120     after_idle_sec = 1
121     interval_sec = 1
122     max_fails = 1
123     sock = socket.socket(socket.AF_INET)
124     sock.setsockopt(socket.SOL_SOCKET, socket.SO_KEEPALIVE, 1)
125     sock.setsockopt(socket.IPPROTO_TCP, socket.TCP_KEEPIDLE,
        after_idle_sec)
126     sock.setsockopt(socket.IPPROTO_TCP, socket.TCP_KEEPINTVL,
        interval_sec)
127     sock.setsockopt(socket.IPPROTO_TCP, socket.TCP_KEEPCNT, max_fails)
128
129     conn = context.wrap_socket(sock, server_hostname=host)
130
131     # Make stdout nonblocking.
132     # Sometimes select says there's output, but read fails.
133     # This makes it possible to handle those cases graciously.
134     fd = comm.stdout.fileno()
```

```
135     fl = fcntl.fcntl(fd, fcntl.F_GETFL)
136     fcntl.fcntl(fd, fcntl.F_SETFL, fl | os.O_NONBLOCK)
137 
138     conn.connect((host, port))
139     identification = {'name': name}
140     conn.send(json.dumps(identification).encode('utf-8'))
141     try:
142         while True:
143             read_list, _, _ = select.select([comm.stdout, conn], [], [])
144             for r in read_list:
145                 if r == conn:
146                     o = r.read()
147                     if o == b'':
148                         return 0
149                     comm.stdin.write(o)
150                     sys.stdout.write('%sserver%s > ' %
151                                      (SERVER_COLOR, ENDC))
152                     sys.stdout.write(o.decode("utf-8"))
153                 else:
154                     try:
155                         o = comm.stdout.readline()
156                     except OSError:
157                         # clear or reset for example get here, it's
    weird.
158                         continue
159                     if (o == b''):
160                         print("Bash has found an error, quit this
    connection")
161                         return
162                     sys.stdout.write(o.decode("utf-8"))
163                     conn.send(o)
164     finally:
165         conn.close()
166 
167 if __name__ == '__main__':
```

```
168         sys.exit(main(argv))
```

code/client.py

C.6 server.py

```
#!/usr/bin/python3 -u
import sys
import socket
import ssl
import getopt
import os
import select
import json

DEFAULT_PORT = 6601
DEFAULT_CAFILE = "rootCA.pem"
DEFAULT_CERTFILE = "confidential/server.crt"
DEFAULT_KEY = "confidential/server.key"
DEFAULT_HOST = ''
DEFAULT_LOG = None

NAME = sys.argv[0]
argv = sys.argv[1:]


def usage(out=sys.stderr):
    global NAME
    print('''Usage:
    %s [-p --port port_number] [-c --cert certificate_file] [-k --key
    key_file] [-a --ca ca_file] [-h --help] [-l --log]
    [-p\t--port\tPort number to bind to. Default is %d.]
    [-c\t--cert\tLocation of certificate file.]
    [-k\t--key\tLocation of key file.]
    [-a\t--ca\tLocation of Certificate Authority file.]
    [--host\tHost the server should bind to.]
    [-h\t--help\tShow this help text.]
    [-l\t--log\tDirectory to put logs in.]
''' % (NAME, DEFAULT_PORT), file=out)

```

```
34 SHORT_OPTIONS = 'p:c:a:k:hl:'
35 LONG_OPTIONS = ['port=', 'cert=', 'ca=', 'key=', 'help', 'log=', 'host='
       ]
36
37
38 def main(argv):
39     try:
40         options, argv = getopt.gnu_getopt(argv, SHORT_OPTIONS,
       LONG_OPTIONS)
41     except getopt.GetoptError:
42         usage()
43         return 2
44
45     def parse_opt(x):
46         return x if x[-1] != '=' else x[:-1]
47     opt_to_names = {'--' + parse_opt(opt): parse_opt(opt)
48                     for opt in LONG_OPTIONS}
49     long_ind = 0
50     for i in SHORT_OPTIONS:
51         if i == ':':
52             continue
53         opt_to_names['-'+i] = parse_opt(LONG_OPTIONS[long_ind])
54         long_ind += 1
55
56     server_args = dict((opt_to_names[opt_name], opt)
57                        for opt_name, opt in options)
58     if 'help' in server_args:
59         usage(out=sys.stdout)
60         return 0
61     else:
62         return start_server(**server_args)
63
64
65 def start_server(cert=DEFAULT_CERTFILE, key=DEFAULT_KEY,
66                  ca=DEFAULT_CAFILE, port=DEFAULT_PORT, host=DEFAULT_HOST
       ,
```

```
67                  log=DEFAULT_LOG):
68     try:
69         port = int(port)
70         error = False
71 
72         if not (0 <= port <= 65535):
73             print("Invalid port number", file=sys.stderr)
74             error = True
75 
76         if cert is None or not os.path.isfile(cert):
77             print("Invalid certificate file", file=sys.stderr)
78             if key is None:
79                 print("There is no appropriate default certificate file.
    ",
80                       file=sys.stderr)
81             error = True
82 
83         if key is None or not os.path.isfile(key):
84             print("Invalid key file", file=sys.stderr)
85             if key is None:
86                 print("There is no appropriate default key file.",
87                       file=sys.stderr)
88             error = True
89 
90         if ca is not None and not os.path.isfile(ca):
91             print("Invalid CA file", file=sys.stderr)
92             error = True
93 
94         if (error):
95             raise getopt.GetoptError("Bad Option")
96     except ValueError:
97         print("Port number must be integer")
98         usage()
99         return 2
100    except getopt.GetoptError:
101        usage()
```

```
102         return 2
103
104     print("Starting server at %s:%d\n"
105           "Certificate: %s\tKey: %s%s" %
106           (host, port, cert, key, '' if ca is None else '\tCA: %s' % ca)
    ,
107           file=sys.stderr)
108
109     context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH, cafile
    =ca)
110     context.load_cert_chain(certfile=cert, keyfile=key)
111
112     # For keepalive
113     after_idle_sec = 1
114     interval_sec = 1
115     max_fails = 1
116
117     bindsocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
118     bindsocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
119     bindsocket.setblocking(False)
120     bindsocket.bind((host, port))
121     bindsocket.listen(5)
122     connected = {}
123     try:
124         while True:
125             read_list = [sys.stdin, bindsocket] + list(connected)
126             readable, _, _ = select.select(read_list, [], [])
127             for r in readable:
128                 if r == sys.stdin:
129                     message = sys.stdin.readline()
130                     if message == '':
131                         return
132                     for c in connected:
133                         c.send(bytes(message, 'UTF-8'))
134                 elif r == bindsocket:
135                     try:
```

```
136                         conn, fromaddr = bindsocket.accept()
137                     except socket.error:
138                         print("Error on accept, ignoring.", file=sys.
    stderr)
139                         continue
140                     conn.setblocking(True)
141                     conn.setsockopt(socket.SOL_SOCKET,
142                                     socket.SO_KEEPALIVE,
143                                     1)
144                     conn.setsockopt(socket.IPPROTO_TCP,
145                                     socket.TCP_KEEPIDLE,
146                                     after_idle_sec)
147                     conn.setsockopt(socket.IPPROTO_TCP,
148                                     socket.TCP_KEEPINTVL,
149                                     interval_sec)
150                     conn.setsockopt(socket.IPPROTO_TCP,
151                                     socket.TCP_KEEPCNT,
152                                     max_fails)
153                     try:
154                         ssl_conn = context.wrap_socket(conn, server_side
    =True)
155                         connected[ssl_conn] = False
156                         print("Computer connected %s" % str(fromaddr))
157                     except (ssl.SSLEOFError, ssl.SSLError,
158                             ConnectionResetError) as e:
159                         print("Bad SSL connection. %s" % str(e), file=
    sys.stderr)
160                 else:
161                     print("Data received from socket.", file=sys.stderr)
162                     try:
163                         o = r.recv()
164                         if connected[r] is False and log is not None:
165                             try:
166                                 identification = json.loads(o.decode('
    utf-8'))
167                                 name = identification['name']
```

```
168                                 if name is not None:
169                                     f = open(log+'/'+name, 'wb', 0) \
170                                         if log is not None else False
171                                     connected[r] = (name, f)
172                                     print(connected[r])
173                                 continue
174                             except:
175                                 pass
176                         if o == b'':
177                             print("Socket closed connection.", file=sys.
    stderr)
178                             del connected[r]
179                             r.shutdown(socket.SHUT_RDWR)
180                             r.close()
181                             continue
182 
183                         if connected[r] is not False:
184                             name, f = connected[r]
185                             f.write(o)
186                     except ValueError:
187                         print("This can rarely happen if someone closes
    "
188                               "connection during accept or handshake
    start.\n"
189                               "Seems to be a bug with SSL or sockets.\n"
190                               "We are just removing the socket from our
    queue "
191                               "in this case", file=sys.stderr)
192                         del connected[r]
193                         continue
194                     except (ConnectionResetError, OSError):
195                         print("This seems to happen due to timeouts,
    although"
196                               " that is not 100% clear. We just cleanly
    remove"
197                               " the socket from the queue when it
```

```
        happens",
198                                 file=sys.stderr)
199                         del connected[r]
200     except KeyboardInterrupt:
201         pass
202     finally:
203         print("Cleaning up connections.")
204         for c in connected:
205             c.shutdown(socket.SHUT_RDWR)
206             c.close()
207         bindsocket.close()
208
209 if __name__ == '__main__':
210     sys.exit(main(argv))
```

code/server.py

C.7 catsoop_login.py

```
#!/usr/bin/python3
import pycurl
import sys
from urllib.parse import urlencode

username = input()
password = input()
web_page = "http://sicp-s4.mit.edu/6.01p/spring16?loginaction=login"
post_data = {
    'login_uname' : username,
    'login_passwd' : password,
}

logged_in = True
cookies = {}
def check_answer(line):
    global logged_in
    logged_in = False

def parse_headers(line):
    global logged_in, cookies
    hdr = line.decode('iso-8859-1')
    if ':' not in hdr:
        return
    name, value = hdr.split(':', 1)
    name = name.strip().lower()
    value = value.strip()
    if name == "set-cookie":
        cname,cvalue = value.split(';')[0].split('=', 1)
        cname = cname.strip().lower()
        cvalue = cvalue.strip()
        cookies[cname] = cvalue

c = pycurl.Curl()
```

```
35 c.setopt(c.URL, web_page)
36 c.setopt(c.WRITEFUNCTION, check_answer)
37 c.setopt(c.POSTFIELDS, urlencode(post_data))
38 c.setopt(c.HEADERFUNCTION, parse_headers)
39 c.perform()
40 c.close()
41
42 assert 'sid' in cookies
43 if not logged_in:
44     sys.exit(1)
45 print(cookies['sid'])
46 sys.exit(0)
```

code/catsoop_login.py

C.8 firefox_sid_cookies.py

```
#!/usr/bin/python3
import configparser
import os
import sqlite3
import time
import sys

DURATION=24*60*60 # 24 hours
NOW=int(time.time())
EXPIRES=NOW+DURATION

# Process the page link
page = "http://sicp-s4.mit.edu/6.01p/spring16"
if "://" in page:
    page = page.split("://", 1)[1]
host, path = page.split("/", 1)
path = "/" # cheating? maybe a bug?
_, domain0, domain1 = host.rsplit('.', 2)
domain = "%s.%s" % (domain0, domain1)

sid_value = input()

parser = configparser.ConfigParser()
home = input()
firefox = os.path.join(home, ".mozilla", "firefox")
profile = os.path.join(firefox, "profiles.ini")
if not os.path.exists(profile):
    sys.exit(0)
# assert os.path.exists(profile)

success = parser.read(profile)
assert profile in success

profile_paths = ((parser[k]['IsRelative'], parser[k]['Path']) \
```

```
35                  for k in parser if 'Name' in parser[k])
36 profile_paths = (os.path.join(firefox,p[1]) if p[0] else p[1] \
37                  for p in profile_paths)
38
39 for p in profile_paths:
40     conn = sqlite3.connect(os.path.join(p, "cookies.sqlite"))
41     c = conn.cursor()
42     try:
43         c.execute("delete from moz_cookies WHERE path=? AND host=? AND
       name='sid'",
44                   (path, host))
45         c.execute("insert into moz_cookies "
46                   "(baseDomain, name, value, host, path, expiry, "
47                   "lastAccessed, creationTime, isSecure, isHttpOnly)"
48                   "VALUES (?, 'sid', ?, ?, ?, ?, ?, 0, 0);",
49                   (domain, sid_value, host, path, EXPIRES, NOW, NOW))
50     finally:
51         conn.commit()
52         c.close()
53         conn.close()
```

code/firefox_sid_cookies.py

C.9 catsoop_login_pam.sh

```
#!/bin/bash
# Descriptive failure
function fail {
    echo $* > /etc/6.01/message
    exit 1
}

# Attempt to login to cat-soop
ping -c 1 sicp-s4.mit.edu &> /dev/null || fail Unable to connect to cat-soop server

# In case of error here,
SID=`(echo $PAM_USER; echo $PAM_AUTHTOK) | /etc/6.01/catsoop_login.py` || fail Incorrect Password
# We've succeeded on the login!
# Add user. If we are here, then this is a cat-soop user.
useradd -s /bin/bash $PAM_USER -d /home/$PAM_USER -K UID_MIN=10000 2> /dev/null
# Mark user as logged in via kerberos
touch /etc/6.01/logins/$PAM_USER.lock

# Create directory which will have unencrypted data
install -m700 -o $PAM_USER -g $PAM_USER -d /home/$PAM_USER || fail Failure creating home

###### START Six-Oh File Transfer STUFF HERE INSTEAD LATER ######
if [ ! -e /etc/6.01/users/$PAM_USER ]
then
    [ -n `mount | grep -w /home/$PAM_USER` ] || fail Home already mounted, no syncing FS
    # Let's create the directory which will have the encrypted data
    install -m700 -o $PAM_USER -g $PAM_USER -d /etc/6.01/users/$PAM_USER || fail Failure syncing
    # Let's encrypt it!
```

```
29      ( echo; echo $PAM_AUTHTOK; echo $PAM_AUTHTOK; ) | \
30    encfs -S /etc/6.01/users/$PAM_USER /home/$PAM_USER &> /dev/null ||
        fail Failure on encryption
31      # Fill it with skel
32      cp -r /etc/skel/. /home/$PAM_USER
33      chown -R $PAM_USER:$PAM_USER /home/$PAM_USER
34      fusermount -u /home/$PAM_USER || fail Failure unmounting home
35  fi
36  ###### END Six-Oh File Transfer STUFF HERE INSTEAD LATER ######
37
38  # Make sure current user owns encrypted directory
39  chown -R $PAM_USER:$PAM_USER /etc/6.01/users/$PAM_USER
40
41  # Mount home directory it not mounted already
42  if [ -z "`mount | grep -w /home/$PAM_USER`" ]
43  then
44      echo $PAM_AUTHTOK | sudo -u $PAM_USER encfs -S /etc/6.01/users/
       $PAM_USER /home/$PAM_USER -- -o allow_other || fail Failure mounting
        home
45  fi
46
47  # Setup firefox profile
48  (echo $SID; echo /home/$PAM_USER) | /etc/6.01/firefox_sid_cookies.py
```

code/catsoop_login_pam.sh

Appendix D

Figures

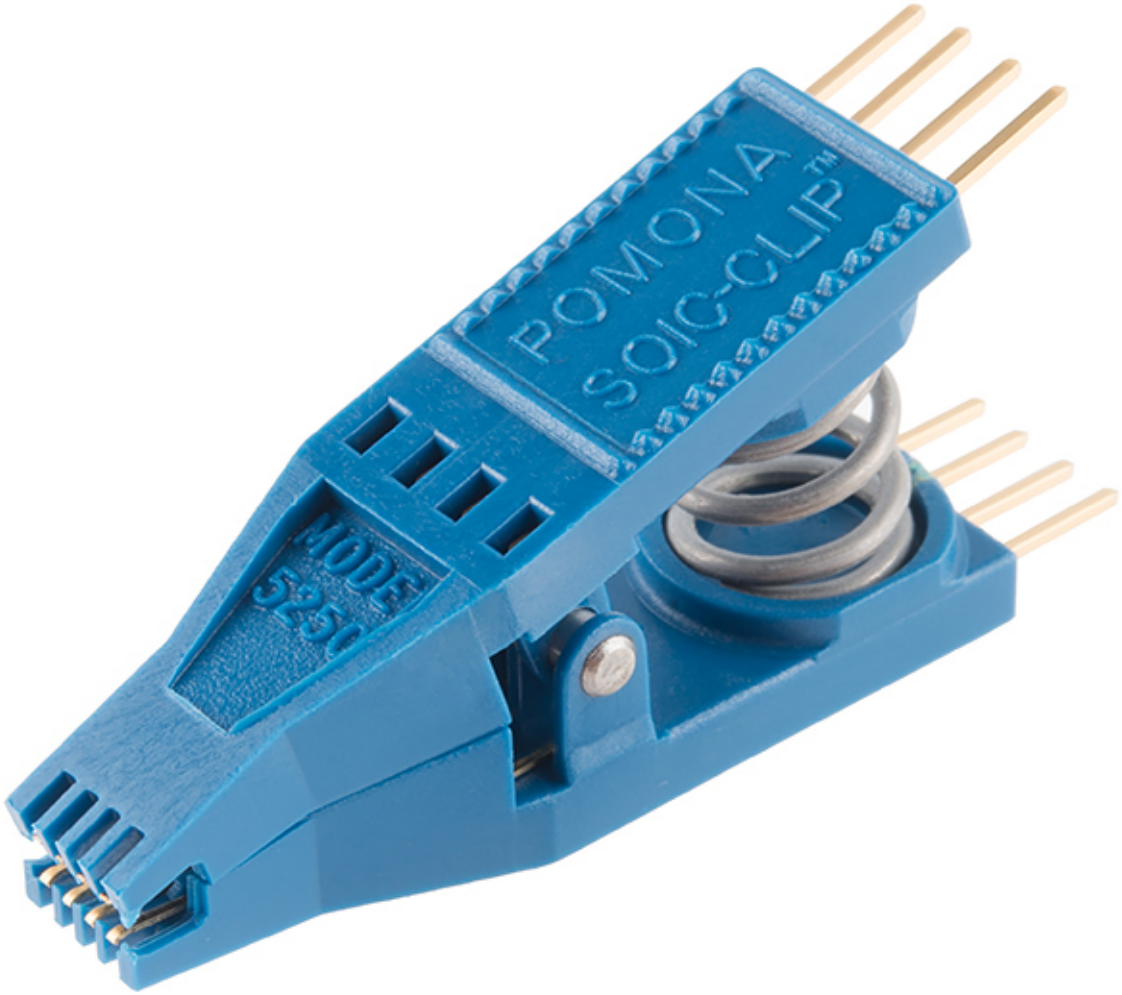

Figure D-1: Integrated Circuit Clip

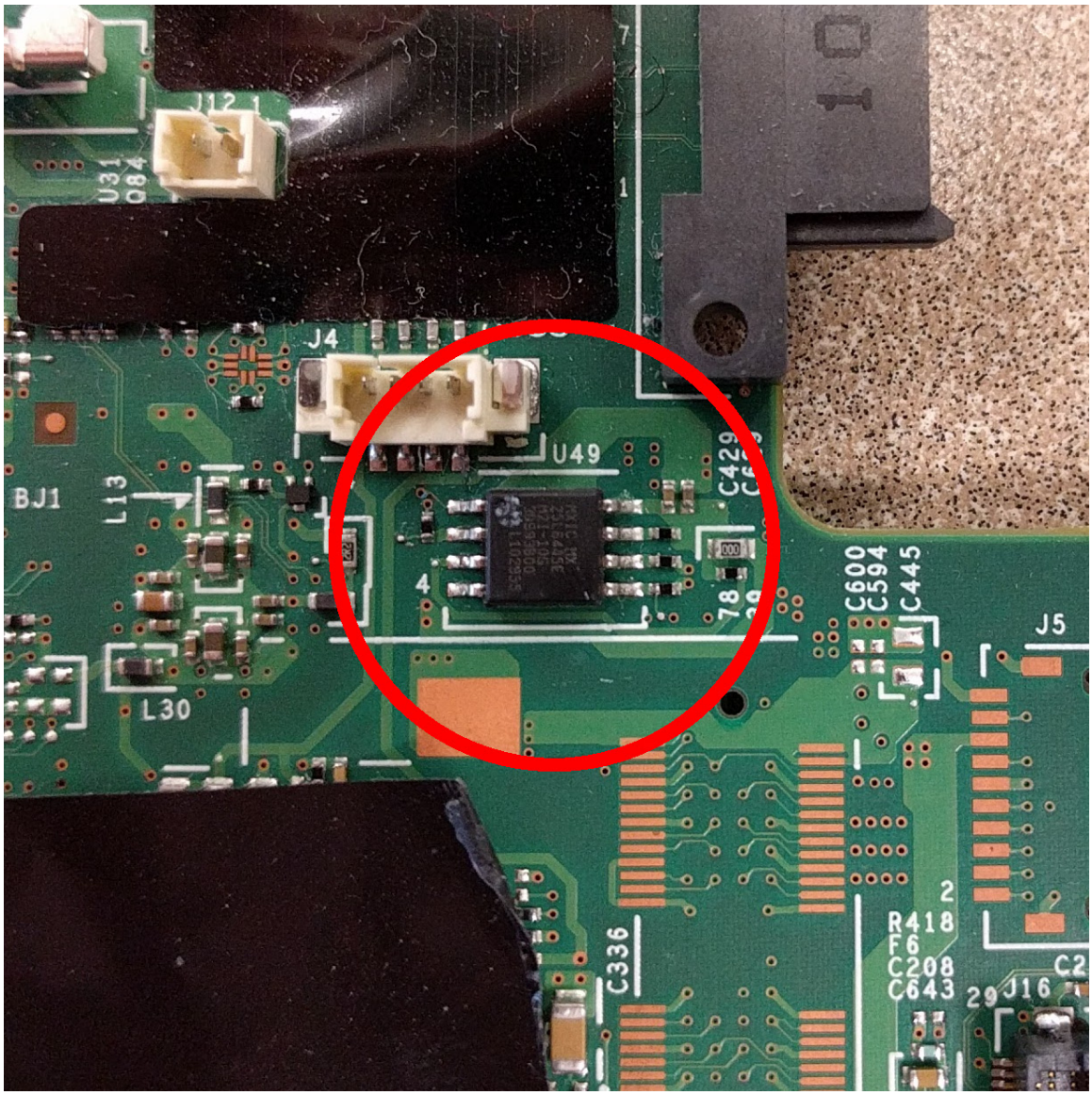

Figure D-2: SOIC-8 chip on T410 Motherboard

Bibliography

[1] Board:lenovo/x201. *Website. https://www.coreboot.org/Board:lenovo/x201.* [Online; accessed 03-May-2016].

[2] Debathena. *Website. http://debathena.mit.edu.*

[3] Detailed specifications - thinkpad t410. *Website. https://support.lenovo.com/us/en/documents/pd006109.* [Online; accessed 13-May-2016].

[4] Do-it-yourself to remove the "white-list" restrictions from lenovo's s10 netbook. *Website. http://www.sbbala.com/DellWWAN/Whitelist.htm.* [Online; accessed 07-May-2016].

[5] Gnu libreboot. *Website. https://libreboot.org/.* [Online; accessed 18-May-2016].

[6] How long do disk drives last? *Website. https://www.backblaze.com/blog/how-long-do-disk-drives-last/.* [Online; accessed 07-May-2016].

[7] How to retrive a good video bios. *Website. https://www.coreboot.org/VGA_support#How_to_retrieve_a_good_video_bios.* [Online; accessed 07-May-2016].

[8] Lenovo thinkpad t61. *Website. http://coreboot.coreboot.narkive.com/7ljvm3wv/lenovo-thinkpad-t61.* [Online; accessed 07-May-2016].

[9] Mx25l6445e high performance serial flash specification. *Website. http://pdf1.alldatasheet.com/datasheet-pdf/view/575511/MCNIX/MX25L6445E.html.*

[10] Thinkpad bios hacking guide. *Website. http://www.endeer.cz/bios.tools/bios.html.* [Online; accessed 07-May-2016].

[11] Thinkpad t410 whitelist removal. *Website. https://www.bios-mods.com/forum/Thread-Thinkpad-T410-whitelist-removal.* [Online; accessed 07-May-2016].

[12] Anton Borisov. Coreboot at your service! *Linux Journal*, 2009(186):1, 2009.

[13] Adam John Hartz. *CAT-SOOP: A tool for automatic collection and assessment of homework exercises.* PhD thesis, Massachusetts Institute of Technology, 2012.

[14] John H Howard et al. *An overview of the andrew file system.* Carnegie Mellon University, Information Technology Center, 1988.

[15] Ali José Mashtizadeh, Andrea Bittau, Yifeng Frank Huang, and David Mazieres. Replication, history, and grafting in the ori file system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 151–166. ACM, 2013.

[16] Patrick "P. J." McDermott. Thinkpad x200. *Website. https://libreboot.org/docs/hcl/x200.html.*

[17] B Clifford Neuman and Theodore Ts' O. Kerberos: An authentication service for computer networks. *Communications Magazine, IEEE*, 32(9):33–38, 1994.

[18] Ronald L Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[19] Rosa Ruiz. Athena clusters scheduled to be renovated or closed. *Website. http://tech.mit.edu/V133/N55/athenacluster.html*, 2016.

[20] Anibal L Sacco and Alfredo A Ortega. Persistent bios infection: The early bird catches the worm. In *Proceedings of the Annual CanSecWest Applied Security Conference, Vancouver, British Columbia, Canada. Core Security Technologies*, 2009.

[21] Vipin Samar. Unified login with pluggable authentication modules (pam). In *Proceedings of the 3rd ACM conference on Computer and communications security*, pages 1–10. ACM, 1996.