Cyber-Security at Desert Academy

New Mexico

Supercomputing Challenge

Final Report

April 1st, 2015

Team #20

Desert Academy

Team Members

Jonas Kaare-Rasmussen

Lileigh Thomas

Teachers

Jocelyne Comstock

Jeff Mathis

Mentor

Jed Crandall

Table of Contents

Executive Summary 3

Introduction 4

- Background Information 5
- Problem 6

Method 6

- Permission 6
- Low-Security Information Gathering 7
- Local-Privilege Escalation Attack 9
- High-Security Information Gathering 10
- Hacks 13

Results 16

- Patches 19
- Conclusion 19

Appendixes 21

- Appendix 1 21
- Appendix 2 22
- Appendix 3 22
- Appendix 4 26

Sources 26

Executive Summary

Cyber-security is indisputably a new and big problem in our society. Major hacks occur often. This proves that cyber-security is a major problem in our present day society. This begs the question, “how secure are we?” This project revolves around attempting to see how secure our student accounts at Desert Academy are from a vicious external hacker, and from students inside the school.

This project attempted to violate the security of the server of which hosts the student accounts from inside the school. This was done in five steps. The first step was gaining legal permission from the school to attempt to hack into the server, within certain boundaries so that only student-generated information was accessed in the project. The second step was finding external information about the server using the program Nmap. This program finds the open, closed, and blocked ports. The third step was a very specific Local-Privilege Escalation attack. This involved giving our student accounts heightened privileges. The fourth step was gathering server-based information using a plethora of bash commands. The final step was the actual hacks. The hacks included many shellshock attacks, a vulnerable script attack, and finally a DOS attack.

All of the attacks were unsuccessful. This gives strong evidence to the fact that the server is safe from both an internal hacker, and an external hacker. However, during this project it became evident that a student could easily access other student-generated information. Simply changing the working directory in a shell to another student’s home directory does this. As our goal stated, we successfully found a vulnerability in the server setup, and plan on patching it with a couple changes in the server permissions layout.

Introduction

Background Information

Cyber-security is a very large topic, and a very new struggle for our society. Information integrity and security has always been important, but it has become exponentially more important. It is estimated that every two days we generate about 5 Exabyte's of new data online [1]. This is approximately the same amount of data generated from the dawn of civilization to about 2003[57]. With this much information, and sharing of information, it is hard to simply quarantine the data from people who use it viciously. Though much of this data is insignificant, like the most recent picture of you in the park, a large portion of this data is incredibly confidential and sensitive. Some of this data includes bank account information to even personal identities.

To protect the important information, companies have designed different protection methods. These methods are in essence what cyber-security is about. Initially security on the Internet was very weak. A person could hack a system, simply by naming a password file something as trivial as "message-of-the-day.[64]" This is no longer the case. However, cyber-security is still a big problem. For instance, approximately 73% of all Americans have been victims of a computer leak[66].

Present day security protocols are powerful to say the least. However, if a computer is connected to a network, it can be hacked simply because

[1] This data come from Techcrunch.com, and is an old figure from 2010. (citation # 57)

another computer can access parts of it. In the same way, if a person knows information, that person could leak that information, simply by knowing it.

Different types of computer have different types of security protocols. For instance, a supercomputer in a National Lab probably runs on what is called a “Local Area Network” (LAN). This means that the computer resides on a network with other computers, but not connected to the Internet at large. In other words, that supercomputer cannot access data on the Internet, like emails or YouTube. This is a very good method of security, because then only an insider can hack the system.

A more often used security method for big computers is limiting the connection power. This is what most big companies do. When a company must access the Internet, they use massive security methods, like firewalls, and other very well planned security protocols. This is what most big companies, and governments use to protect their computers. Though, small and in reality not powerful computers can hack through such firewalls, these computers are great for two reasons. The first is the information of the computers, which is generally sensitive. The second is the fact that servers as such, have a lot of power, which makes them ideal to launch hacking campaigns, and spam.

Small businesses and schools often have servers for websites and student information. These are generally powerful computers, with little information on them, and therefore little security. This makes them big targets for hackers. The idea is simple; if a hacker can gain access to a powerful server, like a school server, they can launch a much bigger and more in-depth attack. This makes school servers’ security key.

Desert Academy is a small school that could potentially be a target for some big attacks, so security is very important. This project is based off of analyzing Desert Academy's security, and attempting to violate it.

At the same time, this project looks at how secure student accounts are from other students. This is to protect student-generated information.

Desert Academy has two servers. They have a server named Doc. This server is the host for the teacher, staff and test accounts. It is also the DNS (Domain Name System) server. The secondary server is name Marty. This server hosts the student accounts and is also the DHCP server (Dynamic Host Configuration Protocol).

Problem

Our project basically attempts to understand the security of our school, to help protect the student's data. Our question was "can we hack into the Desert Academy Server system?"

Method

Our method consisted of five distinct steps. The first step was obtaining permission from the school. The second was finding information about the system externally. The third step was a very specific hack on a client computer to give our already existing student accounts heightened privileges. The fourth step consisted of finding more information accessible only by the newfound privileges. The final step was the major hack attempts, which were only possible because of the information gathered in steps one and three.

Permission

Hacking is illegal. There are many international, national and local laws protecting digitized information and computer system's integrity. One such law is called FERPA (Family Education Rights and Protection Act) [67]. This law protects student education records, and makes it illegal for such records to be published without the consent of the student as well as the parent, if the student is less the 18. FERPA makes it illegal for others to view educational records, so if a hacker were to hack into a server they would be in direct violation of FERPA, as well as other laws. To protect ourselves from violation of FERPA, we needed to find out what was on the server. After talking to the school's technology director, we came to the conclusion that the server contained only student-generated data. In other words, the server held only student accounts. Any personal or confidential data was held on a local computer, so our project was legal, as long as we obtained written permission from the Technology Director, the Head of School, and the legal authority at the school (see appendix 1). After this was obtained, our project could begin.

Low-Security Information Gathering

The first step on infiltrating Desert Academy's network was finding its public IP address. Like telephones, every device that connects to the network has a specific number, called an IP number (IP stands for Internet Protocol). Some organizations have many devices that are routed through an internal server rather than connecting to the Internet directly. These organizations generally have an umbrella public IP address, and then every device under the umbrella address has a local address, which generally takes the form of 192.168. x. x, where x is an integer between 0 and 255 (both included). Finding the public IP of the Desert sever was key, and it was also

very easy. We simply turned on a school computer and went to the website http://www.whatismyip.com/. This site spit out a number (not printed here due to security concerns). To verify that we had the correct IP address, we asked a fellow student who was using his personal computer but had connected to the Internet in the school to go to the same site and look up the IP address. He got the same sequence of numbers verifying that we had the correct IP address.

There are two types of IP addresses. There are dynamic public IP addresses, and static public IP addresses. As the name suggest, a dynamic public IP address changes, whereas the static public IP address does not. After finding Desert's IP address, we were ready to gather more information, but this was only possible if the IP address remained constant. We proved this by accessing the same website (http://www.whatismyip.com/) for the next couple days. The idea behind a dynamic address is that it changes often, therefore making it harder to hack. Because of the fact that the address did not change within a couple days (much less minutes or hours) we made the educated assumption that the school worked under a static IP address.

The Internet is composed of computers talking to each other. A computer talks to another computer through doors, called ports. There is a large number of ports that have their own ways of communication. Most of the ports are closed, which means that they do no accept information, but still register if there is an attempt to communicate. Some ports are open, which means that information travels freely through them. Lastly, a small amount of ports ignore all information sent to them. These are called filtered ports. A good way to learn about the security of a server is to understand the server's port settings. This is done with a program called Nmap. Nmap is a program that does port scans. There are three types of port scans. The first

port scan, called a vertical scan, sends packets of information to all the port on a single IP and records the responses. There is also a horizontal scan, which sends packets to a single port on numerous different IP's looking for a computer with a specific port open. The final type is called a box scan, which is a mix between both a horizontal scan and a vertical scan.

We downloaded Nmap on a Linux computer and did an external port scan on the umbrella IP address for Desert Academy. This showed us that the Desert server had two services running, and the ability to receive "pings". These services were share screen services, so that in all, was no help. This, in the end, isn't completely true, as it was masked by the firewall, which blocked a couple ports.

After the port scan was complete, we were finished with our external research, and we started working on internal research, which required a local-privileges escalation.

Local-Privilege Escalation Attack

Desert Academy uses primarily Apple computers. Mac computers run the Unix operating system. This means that there are different things that different user types can do. There are two major user types, the standard user, and the superuser. The superuser, often called the root user can do almost anything. At Desert Academy, all student accounts are mounted on a server named Marty. The student accounts have user privileges, but not root privileges for the obvious security reasons.

Based on the fact that we have student accounts, we can easily give ourselves the root privileges. Mac computers have a mode called single-user mode. This is accessible by booting the computer up while hold the command and s key. This mode is a read only version of the computer

without network capabilities. It is easy to mount the system so that it is writable, using the commands “fsck –y” (this command looks for problems in the system and fixes them)[47] and the command “mount –uw /” (this command actually reattaches the mount “/” with read and write privileges)[28]. As default, single-user mode runs in root. With the power of a local root account, it is possible to change some very specific files.

Unix-based systems have a command line called BASH (Bourne-Again SHell). In Bash there is a command called sudo, which stands for ‘superuser do’. It is a command that lets the user run a command as root. To be able to use the sudo command, however, the user account must be noted by the system. The system allows only accounts noted in the sudoers file use the sudo command.

While in single user mode, we changed the permission of the sudoers file, and simply added our names to the sudoers file. This gave our student accounts root privileges on the local machine. This was the third local privilege escalation attack we tried, and the only successful attack.

We tried two other attacks that occurred in the single-user mode. We attempted an attack that revolves around unlocking the keychain on the local admin account. This was not a hack, but rather a feature. It was a method for password retrieval, if the user forgot the password. This feature however, is very outdated. It was removed in Mac OS X 10.3, whereas we were using Mac OS X 10.10. Another Local Escalation attack we attempted was running a script we found online. This script could be installed using a thumb drive. It worked by giving a root shell to an external computer, but this script was also outdated.

High-Security Information Gathering

The next step was gathering information about the server so that we could find vulnerabilities. This step was done with relatively small tasks that cumulated to a large image of the server. Most of this step was done with small bash commands that had extensive outputs. A description of each of these commands follows.

The first command we used to map the server was 'traceroute'. This command is called such because it follows the route of a packet, and displays where the packet is, as well as how long it takes to get there[27]. This command showed us that there were no steps between the local computer and the server. This meant that there was no firewall or router between the server and the computer. This was a good sign for our project.

The next command was df -h. df is a command that shows the free disk space on the computer. -h is a modifier to the df command to make it human readable, rather than a string of numbers[26]. This command showed us that the local computer did not have very much stored on it, but that the information was stored on the server. This was already known.

The third command was netstat -a. Netstat is a command that showed how the network was working. -a is a modifier to show network connecting with the server[25]. This showed us the handshakes between Kerberos; the password daemon Mac servers have, as well as the fact that the server was using afpovertcp (Apple Filesharing Protocol over Transfer control Protocol) to communicate with the local machine. This just goes to prove that the server is in fact a Mac server, which we already knew.

The fourth scouting command was one of the most useful commands for this section of the project. System_profiler is a command used to show all the programs on a computer, their version number, as well as a ton of information about the installed applications[24]. Though this command reports

the account programs and setup, the accounts are stored on a server, and they run mirroring the server, so it is basically a window into the servers programs. One of the most interesting things the system profile showed us was the fact that the Adobe Reader is outdated. There is a serious vulnerability in this version of Adobe. However, it yet to published in great detail. This means that, though the vulnerability has been uncovered, the actual exploit has not been published. It is called CVE-2014-9159[12]. The idea is that there is some malicious code buried in a PDF, and upon opening the PDF the code is executed, however, there are no details on how this works. It is a type of buffer overflow attack. In simplest terms, a buffer overflow attack works by executing in the wrong place. When a code runs to display a PDF for instance, it writes to a buffer, but a special PDF might have some code that does not fit into the buffer's limits. This code then gets written to overflow memory, and the malicious code runs. This showed us that outdated stuff gets updated for a reason, and it is a good idea to update your systems.

The fifth command used for this section was the dig command. The dig command looks domain information such as DNS (Domain Name System) and DHCP (Dynamic Host Configuration Protocol). Therefore it is aptly named the domain information groper[30]. This command helped our project by showing us the DNS is passed out by the Doc server, whereas the Marty server passes out the IP's.

The mount and host commands are very similar, and both did not help with the project too much. The mount command looks at the mounts on the computer[28], and host looks at the host[29]. Mount showed us the mounts, but we already knew them, as they were shown mostly in the df output. The host

similarly showed the hostname and the IP address. This was obtained earlier with the dig command.

While inside, we used the Nmap program again, to generate a port scan of the server internally. This proved to do very little, as it gave the same results as outside the server.

Hacks

We attempted three different major hacks, each requiring different information, infiltration methods and steps. They were many shellshock attacks, a script attack, and finally a DOS ping flood attack.

Shellshock attacks

Shellshock is a vulnerability that was published on the 24th of September in 2014[5]. It is actually a wide spread of similar exploits. Shellshock, in general, works by passing the computer an empty variable, and then attaching a line of code, or a script to the empty variable. This only works when the variable runs in a different, and new environment. There are three major remote types of the shellshock bugs. The first is used against websites hosted on a Unix-based server. Some websites have Common Gateway Interface (CGI) running[9]. CGI is a very common way of creating dynamic features on a websites[68]. Another shellshock exploit is through openSSH (Open Secure SHell)[9]. The final type of remote shellshock exploit is through a DHCP server[9].

The CGI attack works by handing the CGI scripts a new environment. The key component for making this work is of course the CGI capability, which is found only in websites. The marty.da.org server does not host a website other then the default Mac Server website. This website does not use

anything close to CGI, but we decided to try to pass it something anyways. We used the command

```
curl -H “User-Agent: () { :;}; cat homedirs_students” marty.da.org
```

Curl simply transfers a URL, and -H modifies it to add extra headers. User Agent is an already existing header, which has a value but this command attempts to change the value of User-Agent to nothing. Afterwards, it attaches a malicious command. Though this example simply prints a directory all student accounts already have access to, more malicious code could be inserted into the ‘cat homedirs_students’ section. Of course, the third section specifies where the curl command is pointed. This did not work because there was nowhere to pass the curl command to. This is an example of code injection, which generally takes the form of Expected Command; Unexpected Command; This is shown with yellow, and green respectfully:

```
curl -H “User-Agent: () { :;}; cat homedirs_students” marty.da.org
```

The OpenSSH attack works by simply piling too many commands on each other[5]. OpenSSH has a thing called “ForceCommand.” This command must be run upon shell startup, even if the user asks the shell to run a different command. The forced command is placed in a new environment generally called “SSH_ORIGINAL_COMMAND.” If the shell is set to bash, the bash shell will parse the “”SSH_ORIGINAL_COMMAND” upon start up. A manufactured “SSH_OPEN_COMMAND” can exploit the shellshock vulnerability. For this to be successful, a server must have OpenSSH and have the shell set to bash. Desert Academy's server does not have OpenSSH[5].

The DHCP attack is by far the easiest to understand. As stated earlier, every device connected to the Internet has an IP number. In the case that computer connects to a server with Ethernet, generally, they need an IP address to be given to them. This is what the DHCP server is for. When a new computer pops up on a DHCP controlled network, it does not know what it is, so the new computer asks, “Who am I?” The server then responds with “I have never seen you before, so here a new address” and then hands the new IP address to the recently opened computer, or the server states “I have seen you before, here is your old IP” and passes the IP that computer had used before[5].

A DHCP attack works by implementing a rogue DHCP server on the network. This server is designed so that it passes an IP address when a computer asks for it, but also passes the string “{ :;}; ./script” as a special option. The ./script then executes in the DHCP environment of the server, effectively “shellshocking” the computer. For this to be successful, the target computer must be rebooted, or at least drop from the network, while the rogue server is online, and the target computer must ask for a DHCP address. This will not work if the target computer has a static IP address. We used the tftpd32 server program to run a rogue server[5].

Script attack

Macintosh computers can have a file called .profile in their home directory. This file is generally empty, but its purpose is to run commands upon opening the terminal application. This file is generally used to tell users that the company head is monitoring them, or explain something special with the shell to a user, or lastly, give a greeting to a returning shell user. It can also be used to say something to the user, such as “Hi User!”

These commands generally are simply print commands, but they are not limited to print commands. The idea with this script was to add lines to the server .profile file so that it copied vital information from the server to a place we could easily access it. This script would be run in a root shell with no hold ups, using the command “nohup ./script.sh”. (See script in Appendix 2).

The third and final attack was a DOS attack. This attack was very simple, and not very thought out. A DOS (Denial of Service) attack, in the most general sense, is an attack that is used to bring down a server so that other users cannot access it. A very simple method of doing such attacks is called a ping flood. The ping command is simply a crafted packet that asks a computer to reply to the sender. It is a common method to test if a computer is working, and connected to the network. Generally pings are sent in succession with a certain amount of time between them so that it does not overwhelm the server. A ping flood without the time increment does overwhelm the server. It literally floods the server with ping request, until the server shuts down. This only works if the server does not have an ICMP (Internet Control Message Protocol) cap.

These were the three hacks attempted on the school server.

Results

The three hacks were altogether unsuccessful, and the server was never exploited. Each hack had its own error, mostly caused by us overlooking a vital detail. The following will explain why each failed.

Our Shellshock attack was in a way almost successful. We had three different attempts. The first was a local shellshock attempt, just to prove that

the bash running on the client computer was vulnerable. We used the command

env val=’() { :;}; echo BUSTED’ bash -c “echo End of command”

The results for this local attack showed us that the bash on the client computer was vulnerable. Because it was vulnerable on the client computer, it was a safe assumption that it was vulnerable on the server as well. Because of these, we tried to shellshock one of the CGI-scripts. We used the command

curl -H “User-Agent: () { :;}; cat homedirs_students” marty.da.org

Because of the fact the server has no CGI scripts this gave us an error message about the target of the curl command, and stated it was not working, but it also spit out a massive HTML (HyperText Markup Language) file. Upon compiling this file we realized that it was the default Mac website for servers that do not host websites. (The raw HTML and compiled HTML is found in Appendix 3). So, in a sense this exploit worked, because it did pass us information, just not the expected information.

The final attempt at a Shellshock attack was the DHCP attack; this attack did not work because of the fact that the server does not use DHCP to find its IP address. In other words, the server always knows who it is, and does not use DHCP hookups, which makes this attack 100% obsolete.

The Script attack did not work. While trying to make it work on a local machine, it was able to change the user accounts .profile file, while on the local account, but when switching between two different local accounts (the guest, and admin account) it gave the error that the .profile was only root editable. This should not have been a problem, as I was using the sudo command to change into the root environment. However, after some

extensive research, I realized that sudo always asks for the user authentication, which is in fact a hold up. Because we used the command nohup, it ignores the hold up, and the script was running at user privilege level. To fix this problem, we tried to open a root shell using the command “sudo su -” and then running the script. This again did not work, and we do not truly understand why. It again threw permissions errors, but we had a root shell. Thought the root of these errors remains unknown, it is something we will be looking into further.

The server, being a Macintosh computer, comes with a default ICMP cap. This means that if too many ping requests are received, it will ignore some of them. Therefore, a ping flood is completely ineffective, as the computer simply ignores the attack.

These three attacks were not successful, however, when it comes to a standpoint of student security, especially from the perspective of students accessing other student’s information. Though this is more of an administrator failure than a hack, it should be noted. Every file on a Unix based system has 3 levels of permissions. It has Owner, Group, and World permissions. Generally, the owner is the user, but at Desert Academy, all the students are in the same group. If a student were to limit the groups’ power on a certain file, they would limit their own power over the file, due to the fact they are in the “student group”. They are simply being imaged on the local machine. In other words, students can easily access, and change other student accounts. This is shown by the fact that a simple change directory command can bring a student into a fellow student's Desktop folder. This is a security problem.

This was also attempted from a student account (hosted on Marty) to a teacher account (hosted on Doc). This was an attack that was meant merely to prove a point. The two servers are not connected other than by the network. To access anything on the Doc server, a student must use an operable file transfer method, such as afpovertcp (Apple Filesharing Protocol over Transfer control Protocol). The cd (change directory) method described above does not do this. An error was expected, but the error that occurred was far from expected. The error was an “Input/output error” which traditionally indicates a disk failure. The Technology Director was promptly informed, and he started doing disk analysis. Some of the Disks on the Doc server were replaced, due to disk failure. After the disks were repaired, the cd attack was attempted again to see if the teacher accounts were now accessible from student accounts. Again, the input/output error was given, so the original error was not due to disk failure. The error comes from unknown origins, and will be reviewed in the future.

Patches

The key to this project was gaining root privileges. Booting the computer into single-user mode did this. Macintosh computer have a feature where administrators can block single-user mode on computer that are meant for more than one user. This is the firmware password. The firmware password disables the single user mode as well as the verbose mode. It also limits the power of recovery mode.

The problem with the students being able to access other student accounts is a much more prominent problem. There is not an easy fix for this problem. A temporary solution is for students to place all of their files in a truecrypt folder, so that they can access their stuff, but others cannot. Other

students can still place numerous copies of files on that student's desktop for the fun of it, but they cannot ruin, or alter any of the students files themselves. These, along with constant updating of software, are the only patches this system needs.

Conclusion

The goal of this project was simply to analyze the security methods of the Desert Academy server. Though, technically, we never broke into the server, we did develop an idea of what the security at Desert Academy is like. Most servers use firewalls and other precautions for protecting open services. Desert however, simply does not have those services. Their philosophy seems to be a hacker cannot break down the door, if there are no doors.

Though the actual server is quite secure, the student-generated information is far from protected from other students. As stated earlier, students sit in the same group. Anyone in that group can access all the groups' information. In other words any student can access any other student information.

Acknowledgements

This project was only possible because of the support of the Desert Academy Technology Department. Throughout the project, they always found time to talk to us when we did not understand how the computers worked. They gave us permission to potentially harm the servers, local accounts, student computers and their own administrator accounts. They were incredibly helpful and inspiring. In return, our project identified some major disk failure, which the Technology Department promptly fixed.

Though the Technology Department was immensely important, Jed Crandall, our advisor, was truly the biggest inspiration and help in this project. Our team came into this project not knowing anything about cyber-security, but Jed quickly changed that. Every time something did not work, or the outcome looked bleak, Jed would think of a new idea to try. He spent many hours writing emails to us, and helping us attack the server. He was the sole reason this project was finished. Without him, we would have given up a long time ago.

Rudy Martinez, Geoff Alexander and Jeff Knockel also deserver credit for this project, as they helped Jed help us. Their insight was one of the main reasons are project has come this far.

Appendixes

Appendix 1

CyberSecurity Initiative at Desert Academy
Hacking into the Main Server

2014-2015 School Year

White-hat hacking is the process of which a hacker attempts to break into a secure database, server, individual computer, or website with the explicit permission from the owner and administrator of the device. White-hat hackers break into devices for the purpose of finding a vulnerability. Upon finding a vulnerability. they immediately inform the system administrator, and owner of the system. The hacker then doesn't tell anyone else about the vulnerability, even if it has been fixed by the system administrators. A white-hat hacker only accesses personal, or confidential data if it helps them on their process of breaking into a machine. For example, they would not look for a social security number, but would look for a password. If they would by chance come upon personal, incriminating, of confidential data, they immediately back out without recording any of the data, and erasing any recorded data, and report to the owner and system administrators.

Our Supercomputing project is based off hacking into Desert Academy's main server. The information on this server is mostly not confidential, incriminating, or personal, other than an Archive folder which will not be accessed in the process of hacking the server. The only personal, incriminating, or confidential data on the server is data placed there by the users. The users are only students and teachers. We will not access any official student, or teacher accounts. We have asked the system administrator to set up fake accounts on the server, which are in all ways the same to a student and teacher account, but not used by either party. This will protect any personal, incriminating, or confidential information on the student and teachers accounts. If we do run into any personal, incriminating or confidential data, we will proceed to reverse our path, delete any recorded data, and inform the owner of the system and the system administrator. We will leave all data untouched, and not change anything. If we mistakenly change something, we will inform the system administrators and the owner of the system.

We, as a Supercomputing team, hereby promise to follow all procedures and protocols outlined above.

Jonas Kaare-Rasmussen | Lileigh Thomas | Lexi Glinsky

As the system owner, and system administrator give this supercomputing full rights for hacking into our server following the procedures and protocols outlined above.

Jud Osborn
Finance Director

Chris Zappe
Technology Director

Terry Passalacqua
Head of School

Appendix 2

```
#!/bin/bash
```

```
str="hello"
filename=/User/macadmin/.profile
while : ;
do
  sleep 1
  if [ -e $filename ]
  then
     value=$( grep -c "hello" $filename )
     if [ $value -eq 0 ]
     then
        echo echo "it worked" >> $filename
echo cd Desktop >> $filename
       exit
fi
 else
    touch $filename
 fi
done
```

Appendix 3

Raw HTML script:

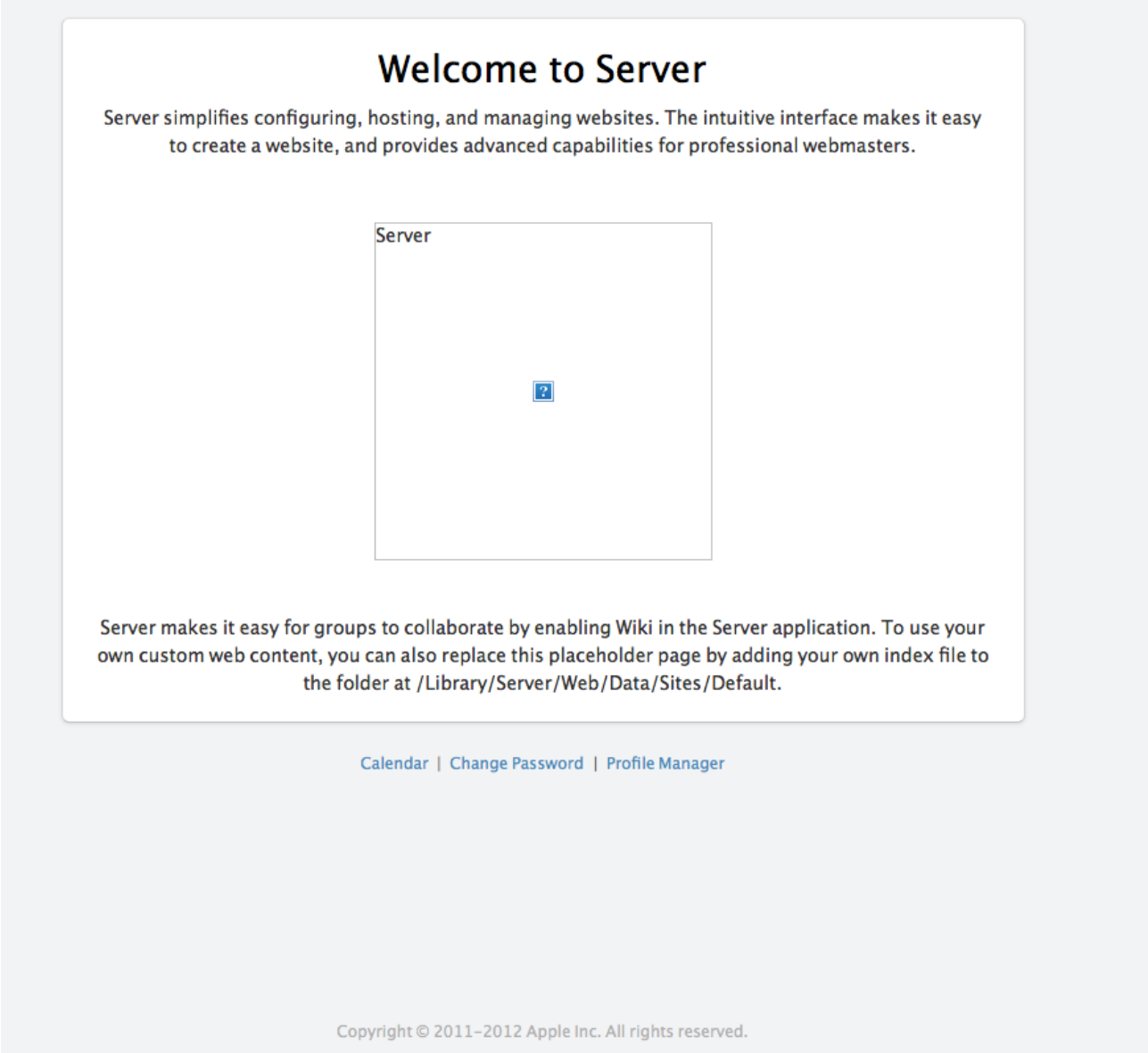
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
        <head>
                <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
                <title>Server</title>
                <style type="text/css">
                * {
                        -webkit-box-sizing: border-box;
                        -moz-box-sizing: border-box;
                        -box-sizing: border-box;
                }
                body, html {
                        background-color: #F2F3F4;
                        color: #333333;
                        font-family: 'Lucida Grande', 'Lucida Sans
```

```
Unicode', Helvetica, Arial, Verdana, sans-serif;
                font-size: 14px;
                height: 100%;
                line-height: 21px;
                margin: 0px;
                text-align: center;
                word-spacing: -1px;
        }
        #wrapper {
                height: 100%;
                min-height: 660px;
                position: relative;
        }
        #content {
                padding: 26px;
        }
        #main {
                background: #FFFFFF;
                border: 1px solid #D5D5D6;
                border-top-color: #E0E1E2;
                border-bottom-color: #C0C1C2;
                margin: 0px auto 0px auto;
                padding: 20px 26px 19px 26px;
                width: 730px;
                -webkit-box-shadow: 0px 1px 3px rgba(0,0,0,0.1);
                -moz-box-shadow: 0px 1px 3px rgba(0,0,0,0.1);
                box-shadow: 0px 1px 3px rgba(0,0,0,0.1);
                -webkit-border-radius: 6px;
                -moz-border-radius: 6px;
                border-radius: 6px;
        }
        h1 {
                color: #000000;
                font-size: 28px;
                font-weight: normal;
                line-height: 36px;
                margin: 0px 0px 8px 0px;
        }
        p {
                line-height: 21px;
```

```
                margin: 0px;
            }
            #main img {
                margin-bottom: 40px;
                margin-top: 48px;
            }
            #navigation {
                color: #646464;
                font-size: 12px;
                margin-top: 20px;
            }
            a, a:link, a:visited, a:active {
                color: #4B8ABA;
                text-decoration: none;
                margin: 3px;
            }
            a:hover {
                text-decoration: underline;
            }
            #footer {
                bottom: 6px;
                left: 0;
                position: absolute;
                right: 0;
            }
            #footer a {
                font-size: 12px;
                color: #B1B1B1;
            }
            </style>
        </head>
        <body>
            <div id="wrapper">
                <div id="content">
                    <div id="main">
                        <h1>Welcome to Server</h1>
                        <p>Server simplifies configuring,
```
hosting, and managing websites. The intuitive interface makes it easy to create a website, and provides advanced capabilities for professional webmasters.</p>

```
                                    <img src="Server.png"
width="256px" height="256px" alt="Server" />
                                    <p>Server makes it easy for groups to
collaborate by enabling Wiki in the Server application. To use your
own custom web content, you can also replace this placeholder page
by adding your own index file to the folder at
/Library/Server/Web/Data/Sites/Default.</p>
                                </div>
                                <div id="navigation">
                                    <a href="/webcal"
title="Calendar">Calendar</a> |
                                    <a href="/changepassword"
title="Change Password">Change Password</a> |
                                    <a href="/profilemanager"
title="Profile Manager">Profile Manager</a>
                                </div>
                                <div id="footer">
                                    <a
href="http://www.apple.com/server/">Copyright &copy; 2011-2012
Apple Inc. All rights reserved.</a>
                                </div>
                            </div>
                        </div>
                    </body>
</html>
```

Appendix 4



Work Cited

Works Cited

1) Dr. Jed Crandall E-mail interview.

2) NVD. "Vulnerability Summary for CVE-2006-2369." Http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2006-2369. NVD, 15 May 2006. Web. 20 Nov. 2014.

3) Wikipedia. "IP Address." Wikipedia. Wikimedia Foundation, 19 Nov. 2014. Web. 20 Nov. 2014.

4)Wikipedia. "Port Scanner." Wikipedia. Wikimedia Foundation, 14 Nov. 2014. Web. 20 Nov. 2014.

5)Wikipedia. "Shellshock (software Bug)." Wikipedia. Wikimedia Foundation, 16 Nov. 2014. Web. 20 Nov. 2014.

6) Adobe. "Vulnerability Details : CVE-2014-9159." *CVE-2014-9159 : Heap-based Buffer Overflow in Adobe Reader and Acrobat 10.x before 10.1.13 and 11.x before 11.0.10 on Windows and OS X a*. CVeDetails, n.d. Web. 29 Jan. 2015.

7) "Adobe_Acrobat_vulnerabilities.html." *Adobe_Acrobat_vulnerabilities.html*. N.p., n.d. Web. 29 Jan. 2015.

8) Google Security Research. "OS X 10.9.x - Sysmond XPC Privilege Escalation." *OS X 10.9.x - Sysmond XPC Privilege Escalation*. Google Security Research, n.d. Web. 29 Jan. 2015.

9) Graham-Cumming, John. "Inside Shellshock: How Hackers Are Using It to Exploit Systems." *Inside Shellshock*. CloudFlare, 30 Sept. 2014. Web. 29 Jan. 2015.

10) Hunt, Troy. "Troy Hunt: Everything You Need to Know about the Shellshock Bash Bug." *Troy Hunt*. Troy Hunt, 25 Dec. 2014. Web. 29 Jan. 2015.

11) Huttunen, Henrik. "Interactive Vim Tutorial." *Interactive Vim Tutorial*. N.p., n.d. Web. 29 Jan. 2015.

12) Jurczyk, Mateusz, and Gynvael Coldwind. "Adobe Reader and Acrobat CVE-2014-9159 Heap Buffer Overflow Vulnerability." *Adobe Reader and Acrobat CVE-2014-9159 Heap Buffer Overflow Vulnerability*. SecurityFocus, 9 Dec. 2014. Web. 29 Jan. 2015.

13) Kaare-Rasmussen, Jonas. Traceroute to servers. 6 Jan. 2015. Raw data. Desert Academy, Santa Fe.

14)Kaare-Rasmussen, Jonas. Mounts on Marty Server. 6 Jan. 2015. Raw data. Desert Academy, Santa Fe.

15) Kaare-Rasmussen, Jonas. What the Marty server is hosting. 6 Jan. 2015. Raw data. Desert Academy, Santa Fe.

16) Kaare-Rasmussen, Jonas. Dig command on the server. 6 Jan. 2015. Raw data. Desert Academy, Santa Fe.

17) Kaare-Rasmussen, Jonas. The system profile of the system. 6 Jan. 2015. Raw data. Desert Academy, Santa Fe.

18) Kaare-Rasmussen, Jonas. Network statistic. 6 Jan. 2015. Raw data. Desert Academy, Santa Fe.

19) Kaare-Rasmussen, Jonas. df command output. 6 Jan. 2015. Raw data. Desert Academy, Santa Fe.

20) Kaare-Rasmussen, Jonas. Remote port scan on Desert output. 6 Jan. 2015. Raw data. Desert Academy, Santa Fe.

21) Kaare-Rasmussen, Jonas. Local Marty Port scan. 6 Jan. 2015. Raw data. Desert Academy, Santa Fe.

22) Kaare-Rasmussen, Jonas. Local Doc Port scan. 6 Jan. 2015. Raw data. Desert Academy, Santa Fe.

23) BSD. *Man Man*. BSD General Commands Manual: Unix, 19 Sept. 2005. GZ.

24)Darwin. *Man system_profiler*. BSD System Manager's Manual: Unix, 30 June 2003. GZ.

25)Darwin. *Man netstat*. BSD General Commands Manual: Unix, 15 June 2001. GZ.

26) BSD. *Man df*. BSD General Commands Manual: Unix, 8 May 1995. GZ.

27) 4.3 Berkeley Distribution. *Man traceroute*. BSD General Commands Manual: Unix, 29 May 2008. GZ.

28) 4th Berkeley Distribution. *Man mount*. BSD General Commands Manual: Unix, 16 June 1994. GZ.

29) BIND9. *Man host*. BIND9: Unix, 30 June 2000. GZ.

30) BIND9. *Man dig*. BIND9: Unix, 30 June 2000. GZ.

31) BSD. *Man cat*. BSD General Commands Manual: Unix, 21 March 2004. GZ.

32) SUDO. *Man sudo*. MAINTENANCE COMMANDS: Unix, 19 July 2010. GZ.

33) Nmap. *Man nmap*. Nmap Reference Guide: Unix, 13 Aug. 2014. GZ.

34) BSD. *Man chmod*. BSD General Commands Manual: Unix, 8 July 2004. GZ.

35)Curl 7.27.0. *Man curl*. Curl Manual: Unix, 27 July 2012. GZ.

36) BSD. *Man env*. BSD General Commands Manual: Unix, 27 Aug Sept. 1993. GZ.

37) BSD. *Man ftp*. BSD General Commands Manual: Unix, 18 July 2007. GZ.

38)BSD. *Man ssh*. BSD General Commands Manual: Unix, 27 March 2015. GZ.

39)VIM. *Man vim*. Unix, 11 Apr. 2006. GZ.

40)1.7.4 *Man sudoers* MAINTENANCE COMMANDS: Unix, 21 July 2010. GZ.

41) BSD. *Man visudo*. MAINTENANCE COMMANDS: Unix, 14 July 2010. GZ.

42) BSD. *Man grep*. BSD General Commands Manual: Unix, 28 July 2010. GZ.

43) Darwin. *Man ipfw*. BSD System Manager's Manual: Unix, 27 Sept. 2012. GZ.

44) BSD. *Man du*. BSD General Commands Manual: Unix, 2 June 2004. GZ.

45) BSD. *Man id*. BSD General Commands Manual: Unix, 26 Sept. 2006. GZ.

46) BSD. *Man whoami*. BSD General Commands Manual: Unix, 6 June 1993. GZ.

47) 4th Berkeley Distribution. *Man fsck*. BSD System Manager's Manual: Unix, 18 May 2010. GZ.

48) Trf transformer commands. *Man bin*. Unix, 1 Feb. 2004. GZ.

49)Darwin. *Man dns-sd*. BSD General Commands Manual: Unix, 27 March 2015. GZ.

50) BSD. *Man arp*. BSD System Manager's Manual: Unix, 18 March 2008. GZ.

51) Mac OS X. *Man ifconfig*. BSD System Manager's Manual: Unix, 27 March 2013. GZ.

52) BSD. *Man rmdir*. BSD General Commands Manual: Unix, 31 May 1993. GZ.

53) Mac OS X. *Man open*. BSD General Commands Manual: Unix, 10 Feb. 2004. GZ.

54) BSD. *Man whois*. BSD General Commands Manual: Unix, 14 June 2004. GZ.

55) MacOSX. *Man dscl*. BSD General Commands Manual: Unix, 25 Aug. 2003. GZ.

56) BSD. *Man w*. BSD General Commands Manual: Unix, 6 June 1993. GZ.

57) Siegler. "Eric Schmidt: Every 2 Days We Create As Much Information As We Did Up To 2003." TechCrunch. Techcrunch, 04 Aug. 2010. Web. 27 Mar. 2015.

58) Stevens, D. "Malicious PDF Documents Explained." *IEEE Xplore*. IEEExplore, 31 Jan. 2011. Web. 29 Jan. 2015.

59) Symantec Security Response. "ShellShock: All You Need to Know about the Bash Bug Vulnerability." *Symantec Security Response*. Symantec, 25 Sept. 2014. Web. 29 Jan. 2015.

60) AllenD, and Sim. "How Can I Execute Local Script on Remote Machine and Include Arguments?" Bash. Ed. Gilles. UNIX&LINUX, 20 Aug. 2013. Web. 30 Jan. 2015.

61)Blaabjerg, Tor, M1k3yo02, and Dennis.hempler. "Enabling SSH Daemon from Terminal (OS X Lion)." Osx. Ed. Bytesum. Superuser, 24 Apr. 2012. Web. 30 Jan. 2015.

62) CVE Details. "Bypass." Security Vulnerabilities (Bypass). CVE Details, n.d. Web. 30 Jan. 2015.

63) The State of Security. Tripwire, n.d. Web. 30 Jan. 2015.

64) Wikipedia. "File Inclusion Vulnerability." Wikipedia. Wikimedia Foundation, 19 Nov. 2014. Web. 30 Jan. 2015.

65) "What Is My IP - The IP Address Experts Since 1999." What Is My IP. N.p., 1999. Web. 27 Mar. 2015.

66) "StopTheHacker.com | Ten Scariest Hacking Statistics." StopTheHacker Ten Scariest Hacking Statistics Comments. N.p., n.d. Web. 30 Mar. 2015. <https://www.stopthehacker.com/2012/04/20/ten-scariest-hacking-statistics/>.

67) "Family Educational Rights and Privacy Act." Wikipedia. Wikimedia Foundation, 30 Mar. 2015. Web. 30 Mar. 2015. <http://en.wikipedia.org/wiki/Family_Educational_Rights_and_Privacy_Act>.

68) "Common Gateway Interface." Wikipedia. Wikimedia Foundation, n.d. Web. 30 Mar. 2015. <http://en.wikipedia.org/wiki/Common_Gateway_Interface>.