Secure Web Development Teaching Modules[1]

Vulnerability Management

Contents

1 Concepts

1.1 Vulnerability Management

Vulnerabilities are flaws (or bugs) in operating systems, software applications, or networking protocols that can be exploited by adversaries to obtain access or elevate access privilege of the computer systems. Vulnerability management is a security practice by which organizations set up procedures to discover vulnerabilities in its systems/applications and to fix the vulnerabilities proactively before being exploited[2].

Patches are additional program added to the original software or systems to address the vulnerabilities. Like bandages, patches fix vulnerabilities or reduce the impact of the vulnerabilities once exploited. Applying patches timely is critical since attackers these days usually exploit vulnerabilities soon after they are discovered. Hence, automatic vulnerability scanning and a standard patch management process enable the risk of vulnerable systems/applications.

1.1 Vulnerability Discovery

Vulnerability scanning is a process of searching for known vulnerabilities in software or computer systems. The scanning can be focused on different aspects of a computer system. For example, port scanning searches for opening ports on a computer system and web application scanning searches for vulnerabilities in web applications/services. Vulnerability scanner such as Nessus looks for vulnerabilities in a computer system against a list of known vulnerabilities, which are mostly well documented in public accessible vulnerability databases.

Web vulnerability scanning focuses on discovering vulnerabilities in a web application/service. Instead of checking for a broad range of software or system vulnerabilities, web vulnerability scanners, different from general vulnerability scanners, check for vulnerabilities embedded in a web application such as testing invalid form inputs or exploiting cookies. Both OWASP[3] and WASC[4] web sites maintain a list of web vulnerability scanners. The Web Application Vulnerability Scanner Evaluation Project (WAVSEP)[5] develops a set of test cases for benchmarking web vulnerability scanners.

1.2 Regulatory Compliance

Regulatory compliance often is a driver of vulnerability management program in both private corporations and government agencies. To be compliant to regulations, corporations will need tools to automate the process of vulnerability discovery and management so that they can address the regulated security controls. Under the Federal Information Security Management Act (FISMA), the U.S. federal agencies have to be compliant to various security controls for operation and management. In addition, the Federal Desktop Core Configuration (FDCC) has extensive requirements for federal agencies in terms of desktop and laptop PCs that run on Microsoft Windows. The National Institute of Standard and Technology (NIST) has developed SCAP (the Security Content Automation Protocol)[6] to standardize the

[2] Peter Mell. Tiffany Bergeron and David Henning. "Creating a Patch and Vulnerability Management Program," NIST Special Publication 800-40, Version 2.0, November 2005.

[3] http://www.owasp.org/index.php/Phoenix/Tools

[4] http://projects.webappsec.org/Web-Application-Security-Scanner-List

[5] http://sectooladdict.blogspot.com/2010/12/web-application-scanner-benchmark.html

[6] David Waltermire, Stephen Quinn and Karen Scarfone. "The Technical Specification for the Security Content Automation Protocol (SCAP): SCAP Version 1.1," NIST Special Publication 800-126 Revision 1, February 2011.

format that security software communicates vulnerabilities and security configuration. Although it cannot address issues in FISMA entirely, SCAP provides capabilities to conduct security checks in controls required by both FISMA and FDCC. Various security vendors have provided SCAP-compatible products to meet this demand[7].

SCAP is a suite of specifications consisted of six components including[8]

- Common Vulnerabilities and Exposures (CVE): a standard for numbering and listing known software vulnerabilities.
- Common Configuration Enumeration (CCE): a standard for identifying security parameters in the configuration of operating systems and software applications.
- Common Platform Enumeration (CPE): a structured naming scheme for information technology systems, platforms, and packages.
- Common Vulnerability Scoring System (CVSS): an open framework for communicating the characteristics and impacts of IT vulnerabilities. CVSS quantifies the risk imposed by vulnerabilities.
- Extensible Configuration Checklist Description Format (XCCDF): is a specification language based on XML for writing security checklists, benchmarks, and related kinds of documents.
- Open Vulnerability and Assessment Language (OVAL): an international, information security, community standard to promote open and publicly available security content, and to standardize the transfer of this information across the entire spectrum of security tools and services. OVAL, based on XML, includes a language to encode system details, and an assortment of content repositories held throughout the information security community.

1.3 Vulnerability Database

Vulnerability databases provide a list of known vulnerabilities and their description in details. These databases include vendor owned databases, such as X-Force database and VUPEN security advisories, community databases, such as the Exploit Database, Open Source Vulnerability Database (OSVDB) and Bugtraq, and the government maintained databases, such as the National Vulnerability Database (NVD). NVD[9] is the U.S. government repository of standards based vulnerability management data, which enables automation of vulnerability management, security measurement, and compliance. All vulnerabilities represented using SCAP.

Web application vulnerabilities as other software vulnerabilities are also included in general purpose vulnerability database such as NVD or OSVDB. OWASP maintains a list of top ten web application vulnerabilities[10] to highlight the prevalence of vulnerabilities in web applications from practitioners' perspectives.

[7] http://nvd.nist.gov/scapproducts.cfm.
[8] Steve H. Weingart. "Using SCAP to Detect Vulnerabilities," atsec information security corporation, 2008.
[9] http://nvdn.nist.gov
[10] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

1.4 Countermeasures to SQL injection

Once vulnerabilities are discovered in software applications, patches have to be applied to remove or avoid the vulnerabilities. This module will discuss examples of countermeasures against two common web application vulnerabilities: SQL injection and Cross-Site Scripting (XSS).

SQL injection is caused by insufficient input validations. Web vulnerability scanners should be able to spot some of the potential risk raised by SQL injections. A proactive countermeasure is to have safe coding practice[11], which include

- Input type checking: developers can filter for invalid input types;
- Encoding of inputs: developers should use functions provided by the programming language or the database APIs to encode inputs so that the invalid inputs will not be considered as database meta-characters;
- Positive pattern matching: developers check for valid input types if checking for negatives are hard to achieve; and
- Identification of all input sources: all input sources are subject to checking.

In our lab exercises, we will examine the SQL injection vulnerabilities in Badstore.net, in which the database queries are handled in Perl. To sanitize inputs, Perl uses two methods to handle dynamic SQL queries: *quote()* and *bind_param()*[12]. The purpose of *quote()* method is to put a pair of quotation marks on the input variables so that any special character in the inputs will be treated as inputs, not part of the SQL meta-characters. For example, in the SQL command below, a quote() method should be applied on the variable to prevent invalid inputs.

```
SELECT * FROM mysql.customer WHERE name = '$username';
```

To prepare for database inputs, the Perl scripts need to sanitize the variable $username as below.

```
my $Username = "Lisa O'Conners";


### $dbh is the database handle

my $quotedUsername= $dbh->quote( $Username );

my $sth = $dbh->prepare( "SELECT * FROM mysql.customer WHERE name =
$quotedUsername" );

$sth->execute();

exit;
```

In Perl, parameters, also called binding or placeholders, are the variables that hold inputs. To sanitize parameters, you will need to use a different method called *bind_param()*.The index number, 1, on the method refers to the first input parameter to be bound.

[11] William G.J. Halfond, Jeremy Viegas, and Alessandro Orso. "A Classification of SQL-Injection Attacks and Countermeasures." the Proceedings of the IEEE International Symposium on Secure Software Engineering Arlington, VA, USA: , March (2006).

[12] Suehring, Steve. Beginning Perl Web Development. Berkeley: APress, 2006.

```
my $sth = $dbh->prepare("SELECT * FROM mysql.customer WHERE name = ?");
$sth->bind_param(1, $username);
```

The same method can also bind multiple parameters as the example below.

```
my $sth = $dbh->prepare("SELECT * FROM mysql.customer WHERE name = ? AND
pass = ?");
$sth->bind_param(1, $username);
$sth->bind_param(2, $password);
```

Intead of using bind_param(), a developer can also use the *execute()* method to bind values with parameters. The example above can be coded as

```
my $sth = $dbh->prepare("SELECT * FROM mysql.customer WHERE name = ? AND
pass = ?");

$sth->execute($username,$password);
```

A dynamic SQL statement with user inputs cannot be injected with crafted SQL codes in Perl parameterized queries. A parameterized query specifies the structure of the query and the placeholders for all user input values. User inputs are interpreted as data instead of a part of the SQL statement code.

1.5 Countermeasures to Cross Site Scripting (XSS)

Sanitizing user inputs is also a key to prevent XSS. The complexity of sanitizing inputs lies on two fronts. First, a web application might permit various types of user inputs. Looking for negatives is always hard while the application developer is trying to avoid false positives. By exploiting a poorly developed application code, attackers are often able to embed malicious HTML-based content within client web requests and exploit these flaws by embedding scripting elements within the returned content without the knowledge of the sites visitors.

Allowing malicious user contents storing in the database often bring liability implication to the web site owners. While sanitizing all user inputs might be hard to achieve, a bandage solution would be for application administrator to scan all user contents being stored in the database and remove the malicious contents in the database periodically. In addition, to block malicious contents, users can protect themselves by blocking scripts from not only untrusted sites but also trusted sites, especially the ones that allow user inputs. However, blocking scripts is often unrealistic since it may greatly limit the functionality of the web site that the users would like to visit.

The best solution for the application developers is to adopt secure coding practice and sanitize all user inputs. One of the most common special characters used to define elements within the markup language is the “<” character, and is typically used to indicate the beginning of an HTML tag. These tags can either affect the formatting of the page or induce a program that the client browser executes, e.g. the <SCRIPT> tag introduces a JavaScript program. The labs in this module will guide students to explore how cross site scripting is executed as well as how the web site developer can fix the vulnerable codes.

2 Labs Objectives

From this lab, you will learn about

- How to discover SQL injection vulnerabilities, such as authentication weakness, on a web server, and investigate solutions to fix the vulnerable codes; and
- Investigate XSS vulnerabilities on a web server and revise the vulnerable codes to prevent XSS.

3 Lab Setup

1. You will use the *ubuntu 10* virtual machine for SWEET teaching modules.
2. Extract the virtual machine from *ubuntu10tm.exe*.
3. Under the folder *ubuntu10tm*, double click on *ubuntu10tm.vmx* to start the virtual machine.
4. The username is "user" and the password is "123456".

4 Lab Guide

4.1 Virtual Machine Startup

1. After logging in, you will see the Ubuntu desktop interface. The virtual machine runs Linux as an independent virtual computer separate from the Windows XP host operating system.

2. Please spend a little time to familiarize yourself with the Linux interface and try the following.
   - Swap back and forth between your Linux virtual machine and the host machine (i.g. Windows). The Linux is run within its own VMware window.
   - Explore the menu bar of the Linux GUI on top of the VM window. The menu bar includes Applications (similar to Windows Start Panel), Places (all devices and storage), and System (Linux system functions).
   - To copy a file from your host machine to the VM, you can drag and drop the file between the two platforms or vice versa.

3. From the menu bar on the top of the VM, select Applications > Accessories > Terminal. This will open up a Linux shell command terminal, execute the command **ifconfig**

4. You will receive several lines of output. You are going to look for the Ethernet interface **(i.g. eth0)**. Find the **inet addr**: field and write down the IP address in the space below.

   _____

5. Visit Badstore web site on the virtual machine. Open a browser. Type in the URL below:

   http://localhost/badstore/

4.2 SQL Injection Risk Area Discovery

1. There are three potential areas on the BadStore web site that is exploitable using SQL Injection. All these areas are related to HTML forms accepting user inputs.

2. Browse the list and input fields on the left panel of the web site. Try to identify at least two risk areas where a malicious SQL query can be injected.

   1) _____

   2) _____

4.3 SQL Injection Attack and Countermeasure: Login Authentication

1. This exercise will examine the SQL injection flaw at login authentication. On the left panel of the BadStore home page, click on the link to Login/Register page. There are several text fields that may be susceptible to SQL injection on this page. Users are possible to be authenticated without providing any credentials. Passwords are usually encoded or hashed so that it is less likely to exploit password fields for SQL injection. The injection must take place within the Email Address text field.

2. Look at the text directly above "Welcome to BadStore,net!" This text box shows what user account that you login as. Your current account should be "Unregistered User".
3. Directly type the following SQL injection string into the Email Address text field under the Login to Your Account section and leave the password field empty.

   'OR 1=1 OR'

4. Look at the text directly above "Welcome to BadStore,net!" again. After the above SQL injection, your user account should have changed. Which user has BadStore.net authenticated you as?

   _____

5. Now, you learned that it is possible to login this website without any password and the SQL injection is possible during registered user login.

6. The next step is to try and login as an administrative user. Since we do not know the backend SQL query for the login/password user entries, we will need to guess the correct syntax to achieve the purpose.

7. Let us type in the following SQL injection string in the email field and see what will happen.
8. admin 'OR 1=1 OR'

9. Again, look at the text directly above "Welcome to BadStore,net!" After the above SQL injection, which user has BadStore.net authenticated you as?

   _____

Take a screen shot of the Badstore home page showing the above user login privilege.

_____

10. What is the security implication that someone could use a SQL injection above to login Badstore?

_____

11. Assume that you are now the web master of the Badstore website and you would like to fix the SQL injection flaw discovered above. You will need to vestigate the CGI program in Badstore. The BadStore CGI program is written in Perl. The program is placed under

    /var/www/badstore/cgi-bin/badstore.cgi

12. To edit the CGI program, in the terminal window execute command:

    sudo gedit /var/www/badstore/cgi-bin/badstore.cgi

13. As discussed in the previous sessions, you will need to format all SQL statement in parameterized queries (also known as prepared statements) so that the input values from the browsers will only be treated as values, not SQL meta-characters. The construction of a SQL statement contains user input is performed in two steps:

14. The Perl code specified the structure of the query, leaving placeholders for each place where user input is expected.

    - The application specified the contents of each placeholder.

    - When the parameterized queries are formatted, the crafted SQL code cannot interfere with the structure of the query specified in the attack above. The query structure has already been defined so that user inputs are always interpreted as data rather than a part of the SQL statement.

15. The next steps will guide you to fix the vulnerable Perl script. Please search for the following original authentication query in a sub procedure called "authuser" in **badstore.cgi**:

    Original Query:

    ### Prepare and Execute SQL Query to Verify Credentials ###

    my $sth = $dbh->prepare("SELECT * FROM userdb WHERE email='$email' AND passwd='$passwd'") ;
    $sth->execute() or die "Couldn't execute SQL statement: " . $sth->errstr;

16. Please modify the authentication query into the following to sanitize the parameters. The modified scripts have been placed on the CGI program as comments (starting with a "#" sign) below the original query. You can just uncomment the modified query and comment out the original query.

    Modified Query:

```
### Prepare and Execute SQL Query to Verify Credentials ###

my $sth = $dbh->prepare("SELECT * FROM userdb WHERE email= ? AND passwd= ?");
$sth->bind_param(1, $email);
$sth->bind_param(2, $passwd);
$sth->execute() or die "Couldn't execute SQL statement: " . $sth->errstr;
```

17. Save the CGI file.

18. Close the browser that has Badstore web site to clear up the login information. Open another new browser to visit Badstore. Go to Login/Register page and type the SQL injection string into the email address field as you did in the previous steps. The SQL injection attack should have been prevented after the change in the Perl script.

19. What is the response from the web site after SQL injection?

_____

4.4 SQL Injection Attack and Countermeasure: Quick Item Search

1. Let us explore another risk area: quick item search. Click on the What's New link in the left pane. The link will bring you to an item page that lists all the items available for sale. If you look at the ItemNum column you can see that the item numbers are not consistent. The missing item numbers imply that there might be more items stored in the database.

2. The text field above the Home link in the left pane is the search window called "Quick Item Search". If you enter an item number, such as 1001, the site will search the database and return with any items that match the value.

3. In order for the database to list all the items that it stores, a search command must be entered with the correct syntax so that it will append the backend SQL query and resolve SQL to be true for every item. Now, leave the search box empty, click on the search bottom. The result will show that it cannot find the item but it provides you with the SQL query it used. Please list the SQL query below.

   _____

4. This SQL query provides abundant information as how to inject SQL command. Based on this SQL query, what value can be injected to list the entire item database?

_____

5. What is the range of item numbers after you performed the SQL injection?

_____

6. What is the risk for a web application when it shows its SQL query to the users?

_____

7. What should an application developer do to mitigate the risk mentioned above?

_____

8. The next steps will guide you to fix the code related to item search. To edit the CGI program, in the terminal window execute command:
   sudo gedit /var/www/badstore/cgi-bin/badstore.cgi

9. The next steps will guide you to fix the vulnerable Perl script. Please search for the following original authentication query in a sub procedure called "search" in **badstore.cgi**:

   Original Search Query:

   ```
   ### Prepare and Execute SQL Query ###
   $sql="SELECT itemnum, sdesc, ldesc, price FROM itemdb WHERE '$squery' IN (itemnum,sdesc,ldesc)";
   my $sth = $dbh->prepare($sql) or die "Couldn't prepare SQL statement: " . $dbh->errstr;
   $temp=$sth;
   $sth->execute() or die "Couldn't execute SQL statement: " . $sth->errstr;
   ```

10. Please modify the query into the following to sanitize the parameters. The modified scripts have been placed on the CGI program as comments (starting with a "#" sign) below the original query. You can just uncomment the modified query and comment out the original query.

    Modified Search Query:

    ```
    ### Prepare and Execute SQL Query ###
    $sql="SELECT itemnum, sdesc, ldesc, price FROM itemdb WHERE ? IN (itemnum,sdesc,ldesc)";
    my $sth = $dbh->prepare($sql) or die "Couldn't prepare SQL statement: " . $dbh->errstr;
    $temp=$sth;
    $sth->execute($squery) or die "Couldn't execute SQL statement: " . $sth->errstr;
    ```

11. In addition, to avoid revealing the SQL command to the users, you should modify the error control script below in the same sub procedure. The modified scripts have been placed on the CGI program as comments (starting with a "#" sign) below the original script. You can just uncomment the modified query and comment out the original query.

Original Script:

```
if ($sth->rows == 0) {
    print h2("No items matched your search criteria: "), $sql, $sth->errstr;
```

Modified Script:

```
if ($sth->rows == 0) {
    print h2("No items matched your search criteria: Please change your search criteria");
```

12. Save the CGI file.

13. Close the browser window that has Badstore web site. Open a new browser to visit Badstore again.
14. Click on "What's New" to see the items. In the Quick Item Search window, type the previous SQL injection string into the search field as you did in the previous steps. The SQL injection attack should have been prevented after the change in the Perl script.

15. What is the response from the web site after SQL injection?

_____

16. Explain the differences between the original search query and the modified search query. Also explain how the modification can fix the SQL injection flaw.

_____

4.5 SQL Injection Attack and Countermeasure: Supplier Login

1. The next SQL injection vulnerability can be exploited through the Supplier Login page. In the left pane, click the Supplier Login link. Unfortunately the suppliers are considered as registered users so that they are stored in the same SQL table as the registered users. The SQL injection queries that were injected into the Login Email Address fields will also work on the Supplier Login page.

2. What injection string can authenticate you as a supplier?

_____

3. Paste the screen shot that the Badstore site authenticates you as a supplier.

_____

4. To fix the SQL injection flaw related to supplier authentication. Again, you need to edit the CGI program, in the terminal window execute command:

   sudo gedit /var/www/badstore/cgi-bin/badstore.cgi

5. You will need to modify a sub procedure called "supplierportal". Please identify the original scripts that need to be fixed in **badstore.cgi** and provide modified scripts.

Original Query:

_____

Modified Query:

_____

6. Close the browser that has Badstore web site to clear up the login information. Open another new browser to visit Badstore. Go to Supplier Login page and type the SQL injection string into the email address field as you did in the previous steps. The SQL injection attack should have been prevented after the change in the Perl script.

7. What is the response from the web site after the SQL injection?

_____

4.6 Explore the Guest Book on BadStore

1. Once on the Badstore.net page, feel free to explore the Badstore.net functionality although you will be focusing on the guestbook in this lab. The guestbook will be the stage where you exploit XSS.

2. To get a feel for how the guestbook works, click the *Sign Our Guestbook* link which is located in the left pane of BadStore.net. This is an average guestbook, where an anonymous visitor can submit his name, email and a general comment to be posted to the guestbook page.

3. Enter a username, email and comment and click *Add Entry* just to test the functionality of the guestbook for yourself.

4. Once your guestbook entry is submitted, you will be able to view your comment in the guestbook plus any previous comment that were submitted. There is no limit to the amount of comments that visitors can add to the guestbook.

4.7 XSS Vulnerability Testing

1. You should be familiar with the guestbook and its functionality by now. You will now test the guestbook for XSS vulnerabilities using JavaScript. If the web application does not sanitize the form data being submitted, Javascript can be injected in a user input and execute on other visitor's browser who views the guestbook using within original web site's security sandbox. To prevent XSS from occurring on others' browser, the only solution for other visitors is to turn off Javascript execution even from trusted sites.

2. Navigate to the *Sign Our Guestbook* page and enter this script below into the comments section of the form.

```
<script>alert(document.cookie)</script>
```

3. The functionality of this script is to produce an alert window with your shopping cart cookie session ID if it is successfully be executed.

4. Click on "Add Entry" to submit the JavScript script to the guestbook. Once the web application processes the script, an alert window will appear showing that the guestbook is vulnerable to XSS.

5. Paste a screen shot of the web site with the alert window.

_____

6. Once the script is submitted to the guestbook, whoever views the guestbook will execute the script within their browser. The script will always execute when the guestbook is viewed by you or others. With malicious payloads, the website moderator will need to remove the malicious comment to restore guestbook functionality.

7. What is the security impact of the XSS attack above?

_____

8. Now, assume that you are the web master for Badstore. To remove the malicious Javascript, you will need to clean up the guestbook contents. Since there is no guestbook moderator web

interface, you will need to go into the BadStore web file directory, locate the guestbook file and delete all the store contents.

9. In the Linux *Places* menu, select *Home Folder*. In the left pane of the File Browser, select *File System* and navigate to the following directory, /var/www/badstore/data/.

10. In the data directory there will be a file called *guestbookdb*. This is the file where all the comments are stored. Right click the file and select *Open with gedit.*

11. Once the file is opened in gedit, you will be able to see all the contents of the guestbook comment section.

12. Delete the malicious content and save the file.

13. In additional to deleting malicious input in the database, Please describe what else can a web master do to prevent such an XSS attack.

4.8 Patching BadStore for XSS vulnerabilities

1. Now that you have experienced possible XSS attacks. As the web master, you will need to fix in the source code of BadStore. The code is implemented as a CGI script below in Perl.

   /var/www/badstore/cgi-bin/badstore.cgi

2. To edit the CGI script, open the terminal windows by navigating to *Applications -> Accessories -> Terminal*

3. Once in the terminal, run the following command to edit the program. If prompted for a password, the password is: *123456*

   sudo gedit /var/www/badstore/cgi-bin/badstore.cgi

4. BadStore.cgi will open in an editor window. Badstore.cgi is the main Perl CGI file that controls BadStore.net functionality. About half way into the file, you will find the Guestbook and Do Guestbook sections, a sub procedure called “doguestbook”.

5. The goal for the modification is to find the code that accepts user comments and sanitize the user comments before it is past to the data file. Since Javascript should not be stored in the guest book. To sanitize the input, the source code should look for any user inputs that contain “<script>” and “</script>” as well as converting them into HTML safe character.

6. In the Guestbook section of Badstore.cgi you will see some code with ### in front of it. This is the code to sanitize the input that is passed in the comments field of the guestbook entry form. There are two ways to sanitize code with the code that is shown.

7. The first section of code will delete any string that begins and ends with <script>:

   $comments=~ s/<script(.*?)<\/script>//isg;

8. This code searches through the text value that was taken from the comment textbox and assigned to the **comments** variable. The =~ expression is used for pattern match. The **s/** specifies substitution. The \ is an escape character for / since / is a reserved character. The **//isg** is a command option with each letter standing for a different option. The **i** option gets rid of any case sensitivity, the **s** option is a squeeze option which replaces every character for every group of matching characters being replaced and the **g** option is used to replace the entire string.

9. To uncomment the code, remove the # symbol from the code that sanitize inputs and save the file.

10. Go back to the BadStore.net Sign Our Guestbook page and enter a regular comment such as "This is a Testing" to make sure the new code does not delete legitimate comments.

11. Submit the comment and check to see if it is displayed correctly.

12. Enter another entry in the guest book but use the malicious script below instead.

    <script>alert('XSS')</script>

13. Does the sanitizing code delete the malicious script? What has happened after you enter the malicious script?

    _____

14. Let us try a different way of fixing the code. Go back to the badstore.cgi source code page and comment out the code that you just used.

15. You will now substitute the "<" or ">" characters with their HTML safe characters. The two sections of code which should be uncommented are:

    $comments=~ s/</\&lt\;/isg;
    $comments=~ s/>/\&gt\;/isg;

Just like before, the comment text value that is assigned from the comment textbox within guestbook to the **comments** variable is searched for characters "<" and ">" and replaces them with their HTML safe characters of **$lt** and **%gt** respectively.

16. Uncomment the code that was just mentioned and save the badstore.cgi file.

17. Go back to the BadStore.net Sign Our Guestbook page and enter a regular comment such as “This is Another Testing” to make sure the new code does not delete legitimate comments.

18. Submit the comment and check to see if it is displayed correctly.

19. Enter another entry in the guest book but use the malicious script below instead.

    <script>alert(‘XSS’)</script>

20. Does the sanitizing code delete the malicious script? What has happened after you enter the malicious script?

    _____

21. Compare the two different ways of sanitizing the user comments.

    _____

22. Comment out the code that was uncommented and save the badstore.cgi file.

23. Clean out any guestbook entries as instructed previously.

24. Close down all open windows and shut down the Badstore.net Linux virtual machine.

4.9 Turn Off Virtual Machines

1. After finishing this exercise, you should reset *Firefox* proxy setting so it stops using the proxy server. Otherwise you would not be able to visit web sites without running the proxy server at port 8088. To do so, Launch your Firefox web browser, and follow its menu item path “Edit|Preferences|Advanced|network Tab|Settings button” to reach the “Connection Settings” window. Check the “Use System Proxy Settings” checkbox.

2. Close Paros (**File|Exit**). Close **Terminal** Windows (type “**exit**” under command line.)

3. Click on the power button on the VM and turn it off.