

COMP 4632

Practicing Cybersecurity: Attacks and Counter-measures

Week 8 Lab Exercise

Topic: Web Application Vulnerabilities

Lab Objective

In this lab, you will try to perform web application attacks. This will include simulated teaching lab and custom made application and aim at achieving the following objectives:

- Understand HTTP and HTTPS protocol
- Understand basics of web related programming language, such as Javascript, SQL
- Identify web vulnerabilities
- Exploit web vulnerabilities

Task 1 – Install, Configure and Use Fiddler

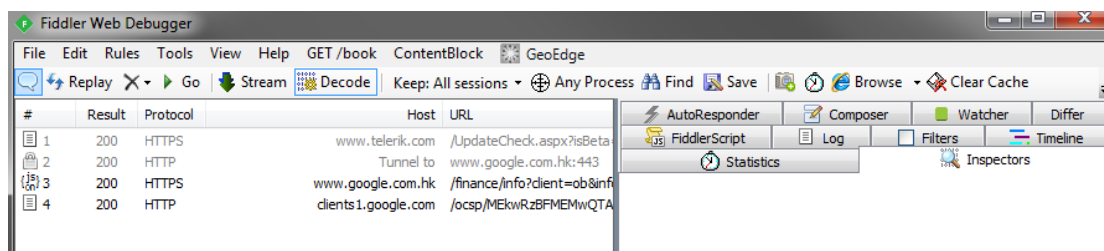
Fiddler is a free web debugging proxy which logs all HTTP(s) traffic between your computer and the Internet. Use it to debug traffic from virtually any application that supports a proxy like IE, Chrome, Safari, Firefox, Opera and more.

Task 1.1 Copy and install Fiddler

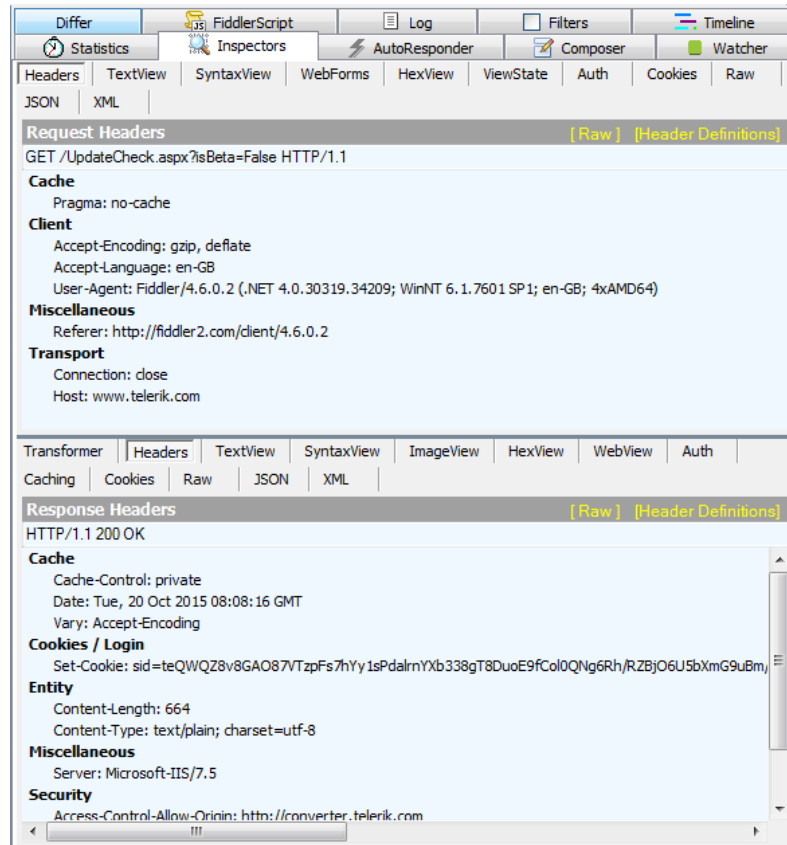
- Switch on the Windows 7 VM
- Drag and drop the fiddler2setup.exe file into Windows 7 VM
- Install Fiddler by double clicking the installation file and following the onscreen instruction.

Task 1.2 Configure Fiddler

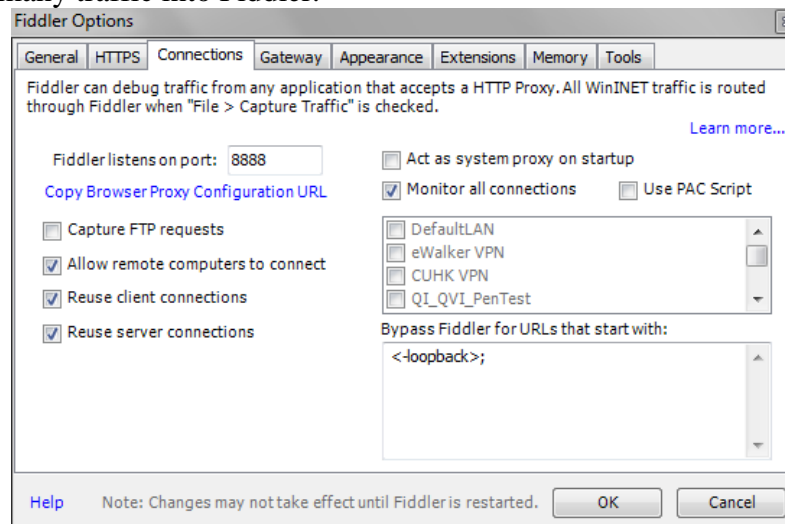
- Start Fiddler from the start menu or desktop icon



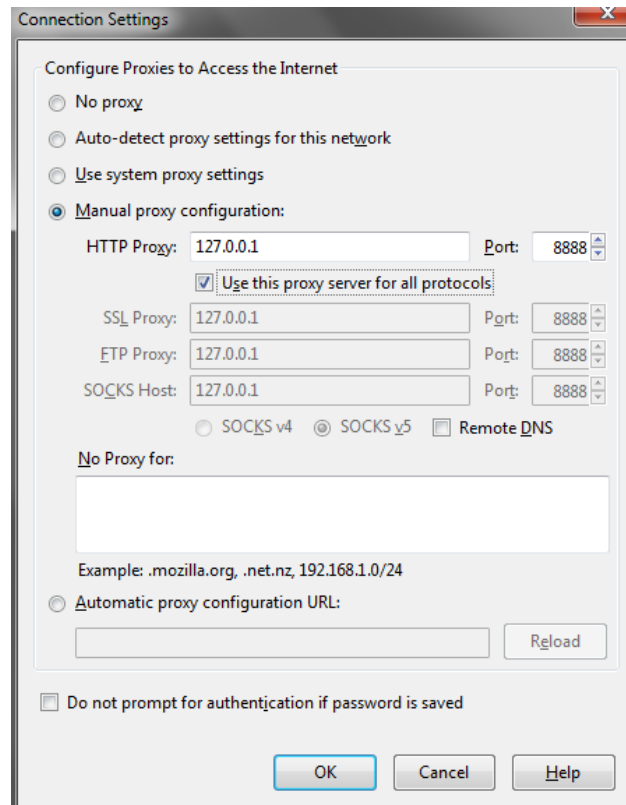
- The left pane is a record of all HTTP(s) traffic that is passing through Fiddler, each of those are called session
- Session details can be viewed in the Inspectors tab on the right pane once a session is being selected.



- Configure browser to use Fiddler as the proxy. First we need to check what port Fiddler is listening on. On the top menu bar, Tools → Fiddler Options → Connections. You should see that Fiddler is listening on port 8888. You may also uncheck the option “Act as system proxy on startup” to avoid collecting too many traffic into Fiddler.







- Configure browser to use Fiddler as the proxy. Depending on the browser you using, choose Options/Settings → Network/Connections → Proxy Setting. Set it to use 127.0.0.1 as the IP and use port 8888 for all connections.

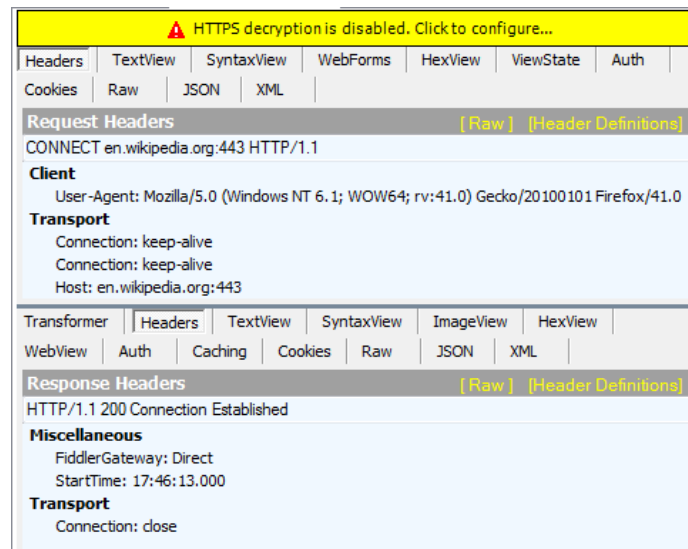


- Verify that your browser traffic is passing through Fiddler.

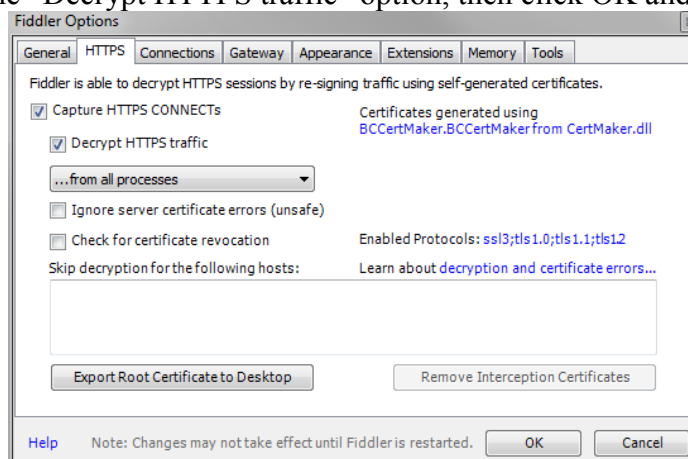
Task 1.3 Decrypting HTTPS traffic

HTTPS (also called HTTP over TLS, HTTP over SSL, and HTTP Secure) is a protocol for secure communication over a computer network which is widely used on the Internet. HTTPS consists of communication over Hypertext Transfer Protocol (HTTP) within a connection encrypted by Transport Layer Security or its predecessor, Secure Sockets Layer. The main motivation for HTTPS is authentication of the visited website and to protect the privacy and integrity of the exchanged data.

- Access any HTTPS website, such as <https://en.wikipedia.org>
 - View the session in Fiddler, an example is shown below
- | | | | | | |
|---|---|-----|------|-----------|--------------------------|
|  | 2 | 200 | HTTP | Tunnel to | en.wikipedia.org:443 |
|  | 3 | 200 | HTTP | Tunnel to | upload.wikimedia.org:443 |
|  | 4 | 200 | HTTP | Tunnel to | login.wikimedia.org:443 |
|  | 5 | 200 | HTTP | Tunnel to | meta.wikimedia.org:443 |
- When you click to see details of a session, you should see a HTTP CONNECT request and not content can be seen.



- When you click to see details of a session, you should see a HTTP CONNECT request and no content can be seen.
- Enable HTTPS decryption in Fiddler, Tools → Fiddler Options → HTTPS, check the “Decrypt HTTPS traffic” option, then click OK and restart Fiddler.



- Observe the difference when accessing the HTTPS site after using this decrypt option.

Task 2 –Using Google Chrome DevTools

The Chrome Developer Tools (DevTools for short), are a set of web authoring and debugging tools built into Google Chrome. The DevTools provide web developers deep access into the internals of the browser and their web application. Use the DevTools to efficiently track down layout issues, set JavaScript breakpoints, and get insights for code optimization. Similar tools are also available in other common browsers nowadays.

Task 2.1 Accessing Google Chrome DevTools

For basic information you may reference the following web pages

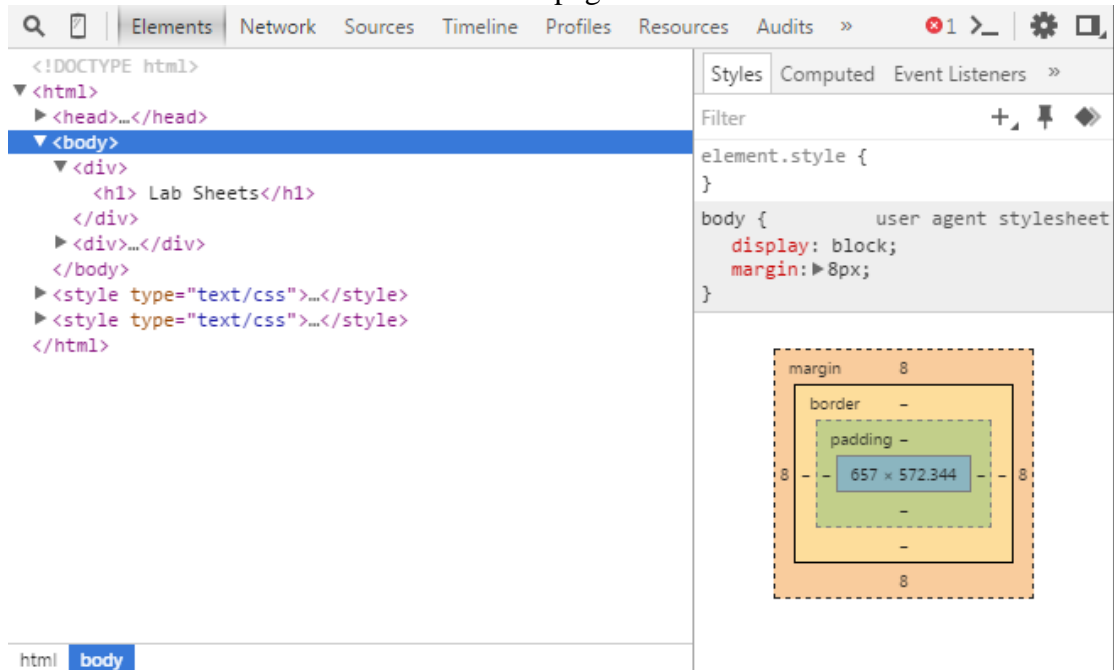
<https://developer.chrome.com/devtools>

To access the DevTools, open a web page or web app in Google Chrome. Either:

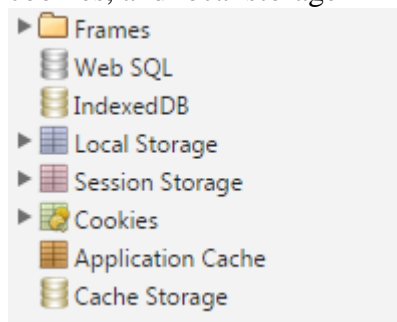
- Select the Chrome menu Chrome Menu at the top-right of your browser window, then select Tools > Developer Tools.
- Right-click on any page element and select Inspect Element.
- The DevTools window will open at the bottom of your Chrome browser.

The followings are some useful panels available in the DevTools:

- **Elements:** Shows the *DOM* of the current page



- **Network:** Display the details of HTTP requests and responses
- **Source:** List the scripts used by the page and support debugging features such as breakpoints, watches, and etc
- **Resources:** List the resources used by the current page, such as frames, cookies, and local storage



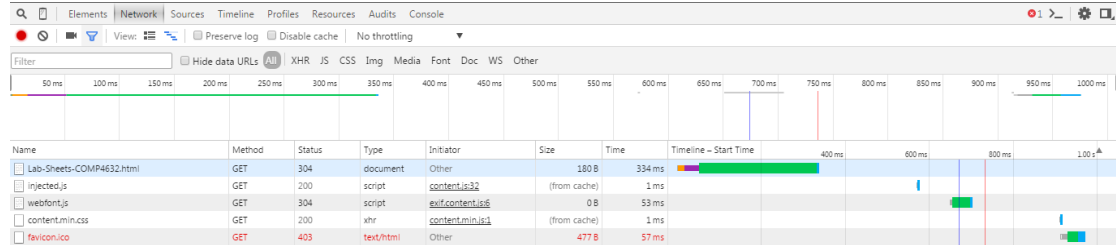
- **Console:** Allow user to run -
 - Javascripts
 - Console API - <https://developer.chrome.com/devtools/docs/console-api>
 - Command Line API: <https://developer.chrome.com/devtools/docs/commandline-api>

There are also some shortcuts for opening the DevTools:

- Use Ctrl+Shift+I or F12 (or Cmd+Opt+I on Mac) to open the DevTools.
- Use Ctrl+Shift+J (or Cmd+Opt+J on Mac) to open the DevTools and bring focus to the Console.

Task 2.2 Analyze web requests with DevTools

- Open Google Chrome and DevTools
- Browser to the main page of course web site by typing URL directly
<https://course.cse.ust.hk/comp4632/>
- Go to Network panel, observe the requests sent when loading the course web site.



- Check the order of the requests (why are they loaded in such order?)
 - There should be some requests in different colors, figure out why?
- Select some resources from the bottom part of the Network panel and observe the requests/responses
 - Find one HTTP request header that does not exist when loading other web page (e.g. CSE home page)
 - Find the server type and version used by the server hosted the course web site
- Keep the DevTools opened, and reload the course web site by pressing F5. Go to the network panel of the DevTools again and identify the difference(s) in headers of HTTP requests and responses comparing to the 1st load.
- Keep the DevTools opened, and browse to the “Lab Sheets” page by clicking on the hyperlink on the course web site. Go to the Network panel again and identify the difference(s) in headers of HTTP requests and responses comparing to the 1st load of the main page of the course web site.

Task 2.3 Analyze and manipulate DOM with DevTools

- Browse to the course web site with DevTools opened. Go to the Element tab of DevTools.
- Pressed the magnifier button (🔍) in the DevTools, and then put the mouse cursor on different elements in the page. Finally click on the left menu (in grey, see below picture) of the course main page. Observe the changes in the Element tab throughout the process.

Course outline

Reference Books

Lecture Slides

Lab Sheets

Assignments

Events

- What type of element is it? What attribute(s) does it have?

- Right click on the HTML element representing the grey box in the Element panel. Add an extra attribute to change the color from grey to other color (Tips: CSS)
- Use the Element panel to add 1 more item in the menu named “LAB 8 Task 2.3 test”
- Reload the page and see what will happen.

Task 2.4 Try out other functions in DevTools

- Browse to the course web site with DevTools opened.
- Try the following tasks with DevTools:
 - Find the list of Javascript file used by the course web pages
 - Add a cookie to the web site in the browser
 - Use the Console panel to change the color of the grey menu box to green.
 - Emulate the behavior of Apple iPhone 6 Plus with DevTools (Find out how ☺) and reload the course web page. Observe the difference.

Task 3 – Configure and Use WebGoat

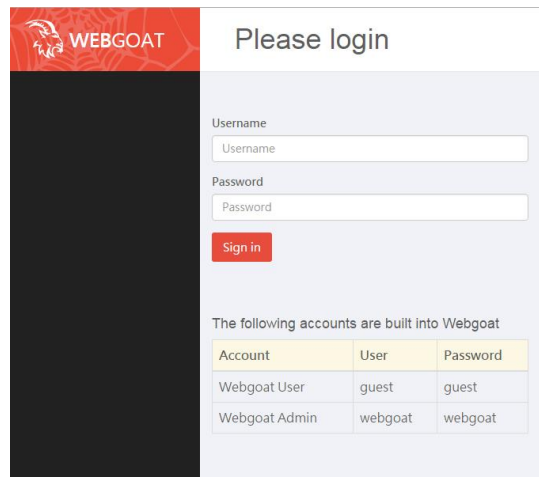
WebGoat is a deliberately insecure web application maintained by OWASP designed to teach web application security lessons. You can install and practice with WebGoat in either J2EE or WebGoat for .Net in ASP.NET. For this lab, we will use the J2EE version. In each WebGoat lesson, users must demonstrate their understanding of a security issue by exploiting a real vulnerability in the WebGoat applications. For example, in one of the WebGoat lessons the user must use SQL injection to steal fake credit card numbers. The application is a realistic teaching environment, providing users with hints and code to further explain the lesson.

Why the name "WebGoat"? Developers should not feel bad about not knowing security. Even the best programmers make security errors. What they need is a scapegoat, right? Just blame it on the 'Goat!

Task 3.1 Copy and start WebGoat 6.0.1

- Download the WebGoat-6.0.1-war-exec.jar from the course webpage
- Drag and drop the WebGoat-6.0.1-war-exec.jar file into Kali Linux
- Start WebGoat 6.0.1 using the following command

```
java -jar WebGoat-6.0.1-war-exec.jar
```
- Access the WebGoat web interface via `http://<kali_linux_ip>:8080/WebGoat/`



WEBGOAT Please login

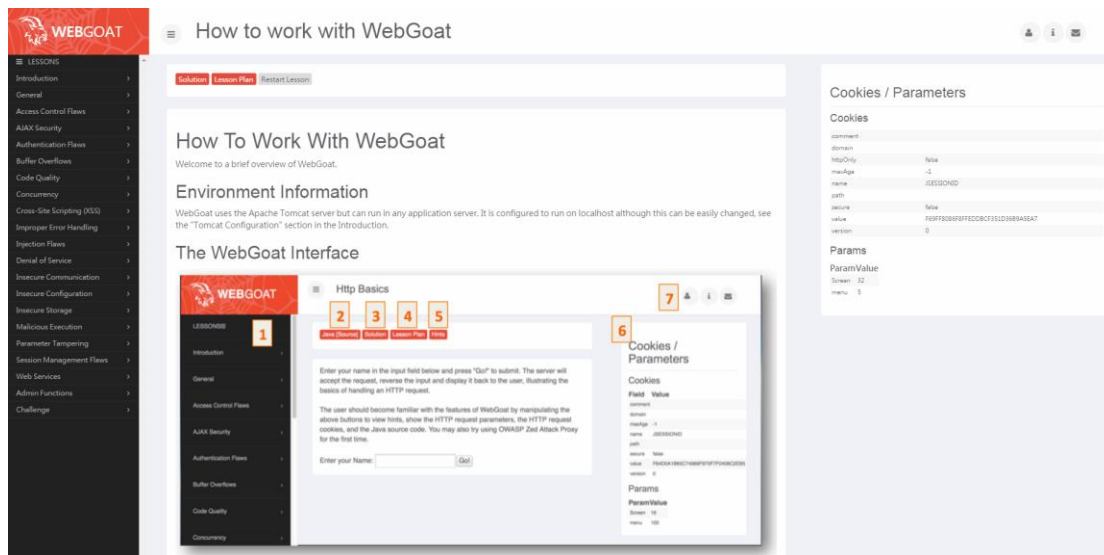
Username

Password

The following accounts are built into Webgoat

Account	User	Password
Webgoat User	guest	guest
Webgoat Admin	webgoat	webgoat

- Use the WebGoat User account to login:
 - Username: guest
 - Password: guest



WEBGOAT How to work with WebGoat

Solutions Lesson Plans Restart Lesson

How To Work With WebGoat

Welcome to a brief overview of WebGoat.

Environment Information

WebGoat uses the Apache Tomcat server but can run in any application server. It is configured to run on localhost although this can be easily changed, see the "Tomcat Configuration" section in the Introduction.

The WebGoat Interface

The user should become familiar with the features of WebGoat by manipulating the above buttons to view hints, show the HTTP request parameters, the HTTP request cookies, and the Java source code. You may also try using OWASP Zed Attack Proxy for the first time.

Enter your Name:

Cookies / Parameters

Cookies

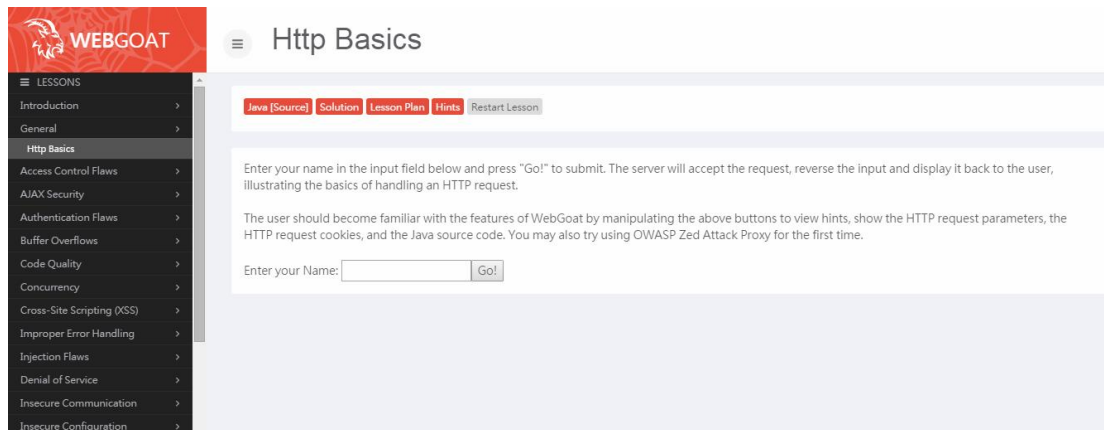
Field	Value
comment	
domain	None
httpOnly	None
maxAge	-1
name	JSESSIONID
path	
secure	None
value	8D7F5B38F8FFEDC8C711D3B8A5A7
version	0

Params

ParamValue
Screen: 32
menu: 5

Task 3.2 WebGoat Lab – Http Basics

- Power on the Kali Linux VM and the Windows 7 VM
- Start Fiddler on the Windows 7 VM
- Access the WebGoat web interface via http://<kali_linux_ip>:8080/WebGoat/ from the Windows 7 VM
- Login with the Webgoat User account
 - Username: guest
 - Password : guest
- Verify that your WebGoat traffic is passing through Fiddler
- WebGoat Lesson – Http Basics
- On the WebGoat menu, select General → Http Basics




- Follow the lab instruction and observe the web traffic in Fiddler

Task 3.3 Modify HTTP Request & Response with Fiddler

- Configure Fiddler to intercept request mode on the bottom tool bar



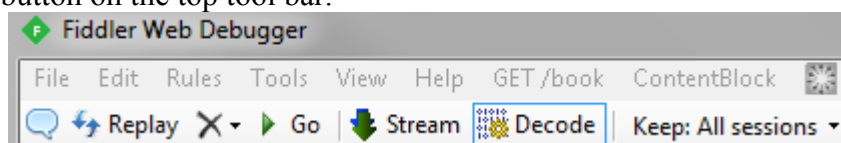
- Try the Http Basics Lab again, you should see an HTTP session which request is intercepted by Fiddler

 97 - HTTP 192.168.222.128 /WebGoat/attack?Screen=16&menu=100

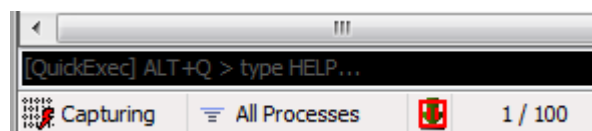
- Go to the Inspectors tab on the right pane and locate your input
- Modify your input and click “Run to Completion”

Breakpoint hit. Tamper, then: Break on Response Run to Completion

- View the result in the web browser
- To resume all other intercepted request without modifying any content, click “Go” button on the top tool bar.

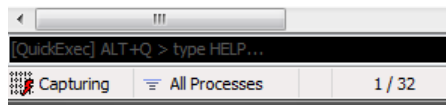


- You can also try out the intercept response mode by click on the bottom tool bar

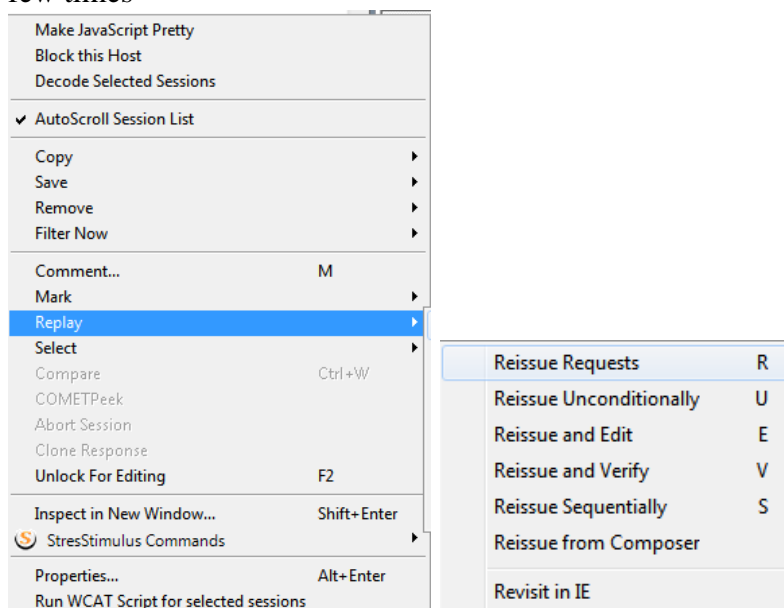


Task 3.4 Analyze HTTP Request & Response with Fiddler

- Configure Fiddler to non-intercept mode on the bottom tool bar (make sure there is no special icons  and )



- Close the browser completely. Reopen it and browse to the WebGoat application and log in.
- Browse to some arbitrary items on the left menu.
- Find out the followings in the logged requests and responses in Fiddler:
 - Server and version hosting WebGoat
 - Cookies involved in the WebGoat
 - Name some possible frameworks / libraries / technologies WebGoat might have used.
 - What are the HTTP methods used by WebGoat? In what situations are these methods used?
 - What are the parameter(s) used by WebGoat to specify what page to load? Where are these parameter(s) submitted to?
- Find the last session for the page </WebGoat/service/lessonplan.mvc> in Fiddler, right click on it and then choose **Replay > Reissue Requests** (or press R) for a few times



The server should return requests with the same content.

- Try to browse to a few other pages in the left menu. Observe the requests and responses for the page </WebGoat/service/lessonplan.mvc>. The requests are usually the same but responses are sometimes different. Try to figure out how this could happen.

Task 4 – Web Application Vulnerabilities – SQL Injection

In following tasks, we will walk through some of the common web application vulnerabilities. You are expected to understand the how to identify and exploit vulnerabilities in the WebGoat lessons with Fiddler.

Task 4.1 SQL Injection

- Start Fiddler on the Windows 7 VM
- Access the WebGoat web interface via <http://<kali linux ip>:8080/WebGoat/> from the Windows 7 VM
- Login with the Webgoat User account
 - Username: guest
 - Password : guest
- Verify that your WebGoat traffic is passing through Fiddler

Task 4.1.1 Numeric SQL Injection

- WebGoat Lesson – Numeric SQL Injection
- On the WebGoat menu, select Injection Flaws → Numeric SQL Injection



- Observe the normal operation flow of the application, also observe what information is being sent in the HTTP request.
- Identify how many inputs/variables in the HTTP request
- Try modifying the inputs/variables with characters which have special or semantic meaning in SQL statements. These characters includes ‘ ‘ ; =, e.t.c.
- Observe the returned HTTP response and web pages.
- Identify the vulnerable parameter
- Try making hypothesis and test to see if it gets the expected outcome
- Complete the lesson objective

Task 4.1.2 String SQL Injection

- WebGoat Lesson – String SQL Injection
- On the WebGoat menu, select Injection Flaws → String SQL Injection

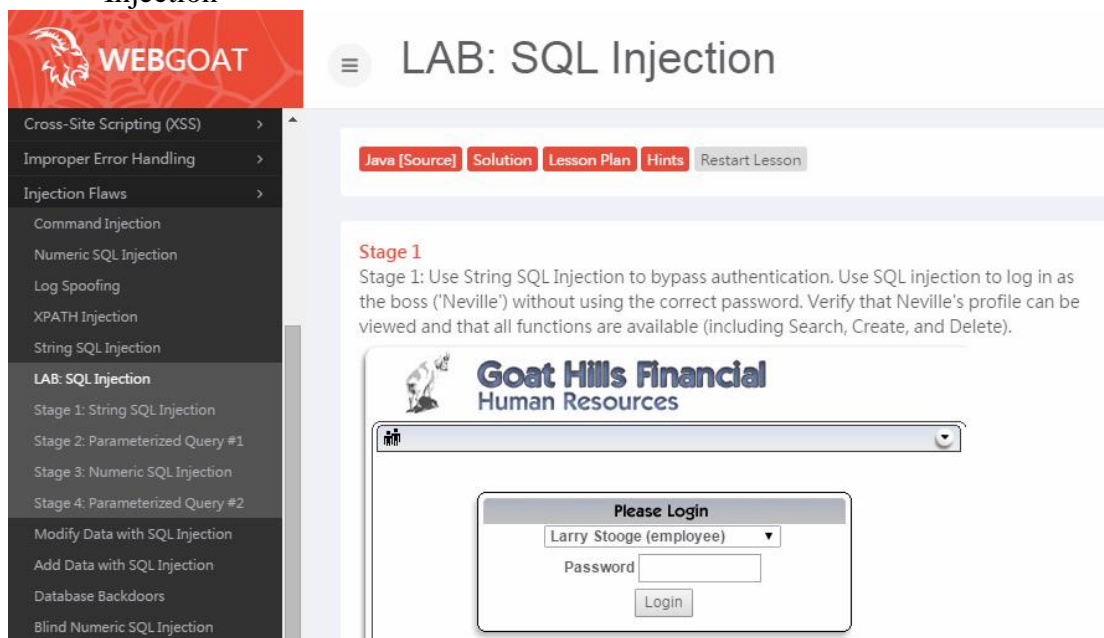


The screenshot shows the WebGoat interface for the 'String SQL Injection' lesson. The left sidebar lists various attack types, with 'String SQL Injection' selected. The main content area has a title 'String SQL Injection' and a sub-header 'LAB: SQL Injection'. Below this, there is a text block explaining that SQL injection attacks are a serious threat and that the methods are easy to learn. At the bottom, it states that the attack can be prevented with common-sense and forethought. Navigation links for 'Java [Source]', 'Solution', 'Lesson Plan', 'Hints', and 'Restart Lesson' are visible at the top of the content area.

- Observe the normal operation flow of the application, also observe what information is being sent in the HTTP request.
- Identify how many inputs/variables in the HTTP request
- Try modifying the inputs/variables with characters which have special or semantic meaning in SQL statements. These characters includes ‘ “ ; =, e.t.c.
- Observe the returned HTTP response and web pages.
- Identify the vulnerable parameter
- Try making hypothesis and test to see if it gets the expected outcome
- Complete the lesson objective
- How is the solution different from the one in Task 4.1.1? Why there is such a difference?

Task 4.1.3 LAB: SQL Injection – Stage 1

- WebGoat Lesson – LAB: SQL Injection
- On the WebGoat menu, select Injection Flaws → Stage1: String SQL Injection



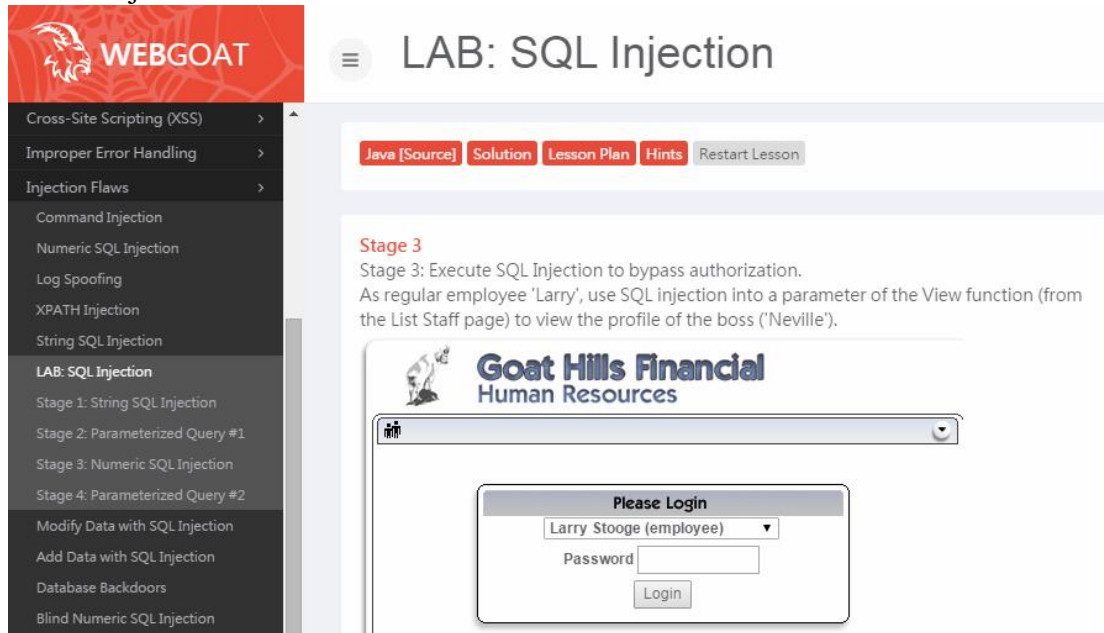
The screenshot shows the WebGoat interface for the 'LAB: SQL Injection' lesson, specifically Stage 1. The left sidebar lists various attack types, with 'LAB: SQL Injection' selected. The main content area has a title 'LAB: SQL Injection' and a sub-header 'Stage 1'. Below this, there is a text block explaining that the goal is to use String SQL Injection to bypass authentication and log in as the boss ('Neville') without using the correct password. A screenshot of the 'Goat Hills Financial Human Resources' login page is shown, featuring a 'Please Login' form with a dropdown menu for 'Larry Stooze (employee)', a password field, and a 'Login' button. Navigation links for 'Java [Source]', 'Solution', 'Lesson Plan', 'Hints', and 'Restart Lesson' are visible at the top of the content area.

- Observe the normal operation flow of the application, also observe what information is being sent in the HTTP request.
- Identify how many inputs/variables in the HTTP request
- Try modifying the inputs/variables with characters which have special or semantic meaning in SQL statements. These characters includes ‘ “ ; =, e.t.c.

- Observe the returned HTTP response and web pages.
- Identify the vulnerable parameter
- Try making hypothesis and test to see if it gets the expected outcome
- Complete the lesson objective

Task 4.1.4 LAB: SQL Injection – Stage 3

- On the WebGoat menu, select Injection Flaws → Stage3: Numeric SQL Injection



The screenshot shows the WebGoat application interface. On the left is a dark sidebar menu with the 'WEBGOAT' logo at the top. The menu items include 'Cross-Site Scripting (XSS)', 'Improper Error Handling', 'Injection Flaws', 'Command Injection', 'Numeric SQL Injection', 'Log Spoofing', 'XPath Injection', 'String SQL Injection', 'LAB: SQL Injection' (which is expanded to show 'Stage 1: String SQL Injection', 'Stage 2: Parameterized Query #1', 'Stage 3: Numeric SQL Injection', and 'Stage 4: Parameterized Query #2'), 'Modify Data with SQL Injection', 'Add Data with SQL Injection', 'Database Backdoors', and 'Blind Numeric SQL Injection'. The main content area is titled 'LAB: SQL Injection' and contains a lesson for 'Stage 3'. The lesson text states: 'Stage 3: Execute SQL Injection to bypass authorization. As regular employee 'Larry', use SQL injection into a parameter of the View function (from the List Staff page) to view the profile of the boss ('Neville').' Below the text is a screenshot of the 'Goat Hills Financial Human Resources' application. It features a 'Please Login' form with a dropdown menu showing 'Larry Stooze (employee)', a 'Password' input field, and a 'Login' button.

- Observe the normal operation flow of the application, also observe what information is being sent in the HTTP request.
- Identify how many inputs/variables in the HTTP request
- Try modifying the inputs/variables with characters which have special or semantic meaning in SQL statements. These characters includes ‘ “ ; =, e.t.c.
- Observe the returned HTTP response and web pages.
- Identify the vulnerable parameter
- Try making hypothesis and test to see if it gets the expected outcome
- Complete the lesson objective

Task 4.1.5 SQL injection in PHP

- Download 2 files from the course website, namely L8T415.php and L8T415.sql
- Upload the L8415.php to your web server VM via FTP or other means, and placed it in /var/www/html/, as in previous lab sessions.
- Make sure that mysql.php from previous lab is in the same directory as the L8T415.php
- Connect to your database server and use the content of L8T415.sql to setup an additional table in your database.
- Try to verify the access to L8T415.php from your client web browser.
- Identify and try to perform SQL injection to enumerate information about the database, such as database version.
- Enumerate the table and find hidden row or column.

Task 5 – Web Application Vulnerabilities – Cross-site Scripting (XSS)

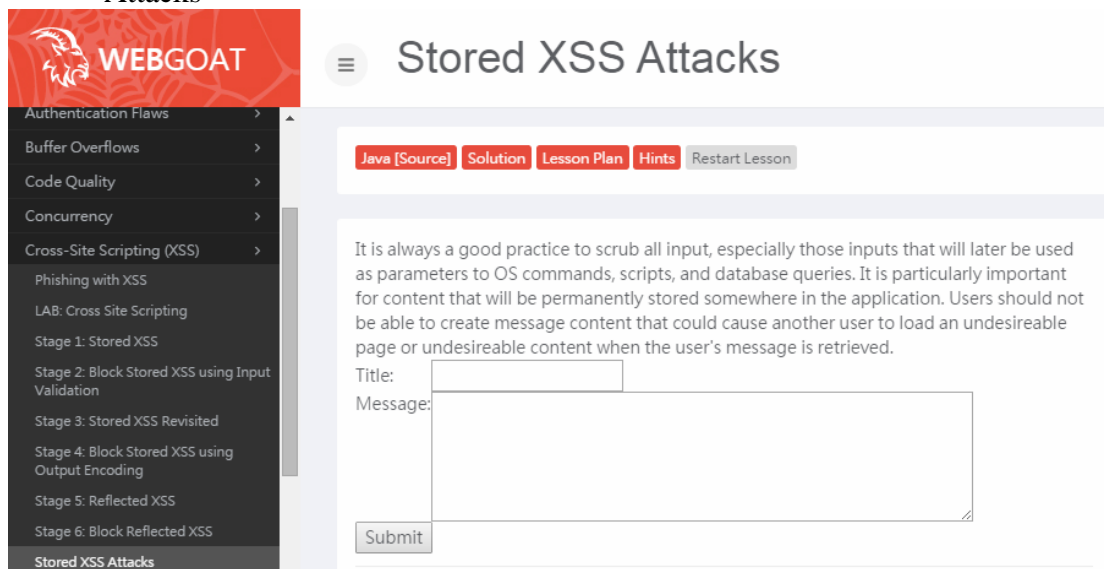
In this task, we will walk through some of the common web application vulnerabilities. You are expected to understand the how to identify and exploit vulnerabilities in the WebGoat lessons with Fiddler.

Task 5.1 Cross-site Scripting (XSS)

- Start Fiddler on the Windows 7 VM
- Access the WebGoat web interface via <http://<kali linux ip>:8080/WebGoat/> from the Windows 7 VM
- Login with the Webgoat User account
 - Username: guest
 - Password : guest
- Verify that your WebGoat traffic is passing through Fiddler

Task 5.1.1 Stored XSS Attacks

- WebGoat Lesson – Stored XSS Attacks
- On the WebGoat menu, select Cross-Site Scripting (XSS) → Stored XSS Attacks



- Observe the normal operation flow of the application, also observe what information is being sent in the HTTP request.
- Identify how many inputs/variables in the HTTP request
- Try modifying the inputs/variables with characters which have special or semantic meaning in HTML. Such as HTML tags `
`, ``, ``, e.t.c.
- Observe the returned HTTP response and web pages.
- Identify the vulnerable parameter
- Try modifying the parameter with characters which have special or semantic meaning in HTML and Javascript. Such as `<script>...</script>`, `onclick=""`, `onerror=""`, e.t.c.
- Try making hypothesis and test to see if it gets the expected outcome
- Complete the lesson objective

Task 5.1.2 Reflected XSS Attacks

- WebGoat Lesson – Reflected XSS Attacks
- On the WebGoat menu, select Cross-Site Scripting (XSS) → Reflected XSS Attacks

WEBGOAT

Authentication Flaws > Buffer Overflows > Code Quality > Concurrency > Cross-Site Scripting (XSS) > Phishing with XSS > LAB: Cross Site Scripting > Stage 1: Stored XSS > Stage 2: Block Stored XSS using Input Validation > Stage 3: Stored XSS Revisited > Stage 4: Block Stored XSS using Output Encoding > Stage 5: Reflected XSS > Stage 6: Block Reflected XSS > **Stored XSS Attacks** > **Reflected XSS Attacks** > Cross Site Request Forgery (CSRF) > CSRF Prompt By-Pass > CSRF Token By-Pass > HTTPOnly Test > Cross Site Tracing (XST) Attacks > Improper Error Handling >

Reflected XSS Attacks

Java [Source] Solution Lesson Plan Hints Restart Lesson

It is always a good practice to validate all input on the server side. XSS can occur when unvalidated user input is used in an HTTP response. In a reflected XSS attack, an attacker can craft a URL with the attack script and post it to another website, email it, or otherwise get a victim to click on it.

Shopping Cart

Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	69.99	1	\$0.00
Dynex - Traditional Notebook Case	27.99	1	\$0.00
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	1	\$0.00
3 - Year Performance Service Plan \$1000 and Over	299.99	1	\$0.00

The total charged to your credit card: \$0.00 UpdateCart

Enter your credit card number:

Enter your three digit access code:

Purchase

- Observe the normal operation flow of the application, also observe what information is being sent in the HTTP request.
- Identify how many inputs/variables in the HTTP request
- Try modifying the inputs/variables with characters which have special or semantic meaning in HTML. Such as HTML tags `
`, ``, ``, e.t.c.
- Observe the returned HTTP response and web pages.
- Identify the vulnerable parameter
- Try modifying the parameter with characters which have special or semantic meaning in HTML and Javascript. Such as `<script>...</script>`, `onclick=""`, `onerror=""`, e.t.c.
- Try making hypothesis and test to see if it gets the expected outcome
- Complete the lesson objective

Task 5.1.3 Cross Site Request Forgery (CSRF)

- WebGoat Lesson – Cross Site Request Forgery (CSRF)
- On the WebGoat menu, select Cross-Site Scripting (XSS) → Cross Site Request Forgery (CSRF)

WEBGOAT

Cross-Site Scripting (XSS) >

- Phishing with XSS
- LAB: Cross Site Scripting
- Stage 1: Stored XSS
- Stage 2: Block Stored XSS using Input Validation
- Stage 3: Stored XSS Revisited
- Stage 4: Block Stored XSS using Output Encoding
- Stage 5: Reflected XSS
- Stage 6: Block Reflected XSS
- Stored XSS Attacks
- Reflected XSS Attacks
- Cross Site Request Forgery (CSRF)**
- CSRF Prompt By-Pass

Java [Source] Solution Lesson Plan Hints Restart Lesson

Your goal is to send an email to a newsgroup that contains an image whose URL is pointing to a malicious request. Try to include a 1x1 pixel image that includes a URL. The URL should point to the CSRF lesson with an extra parameter "transferFunds=4000". You can copy the shortcut from the left hand menu by right clicking on the left hand menu and choosing copy shortcut. Whoever receives this email and happens to be authenticated at that time will have his funds transferred. When you think the attack is successful, refresh the page and you will find the green check on the left hand side menu.
Note that the "Screen" and "menu" GET variables will vary between WebGoat builds. Copying the menu link on the left will give you the current values.

Title:

Message:

- Observe the normal operation flow of the application, also observe what information is being sent in the HTTP request.
- Identify how many inputs/variables in the HTTP request
- Try modifying the inputs/variables with characters which have special or semantic meaning in HTML. Such as HTML tags
, , , e.t.c.
- Observe the returned HTTP response and web pages.
- Identify the vulnerable parameter
- Try modifying the parameter with characters which have special or semantic meaning in HTML and Javascript. Such as <script>...</script>, onclick="'", onerror="'", e.t.c.
- Try making hypothesis and test to see if it gets the expected outcome
- Construct the required URL and script for this lesson.
- Complete the lesson objective

Task 5.1.4 Bypassing Filters on XSS

- Create a folder named tempwww on Desktop of Kali Linux. Change directory into it and run a temporary PHP web server at TCP 8088 port rooted in that folder:

```
mkdir ~/Desktop/tempwww
cd ~/Desktop/tempwww
php -S 0.0.0.0:8088
```

- Copy the file L8T514.php to the tempwww folder
- Browse the page http://<kali_linux_ip>:8088/L8T514.php, you will see a simple form with a text area

submit

- Try to input any text in the box and observe the behavior of the PHP page. You may also check the PHP source code on the logic.

- A function is used to filter the inputted text

```
3     function clean_args($arg){
4         $arg = preg_replace("/<script[^>]*>/", "", $arg);
5         $arg = preg_replace("/<\/script[^>]*>/", "", $arg);
6         return $arg;
7     }
```

- Try to bypass the filter by:
 - Case 1: using HTML DOM events
 - Case 2: Using only <script> elements

End of Lab