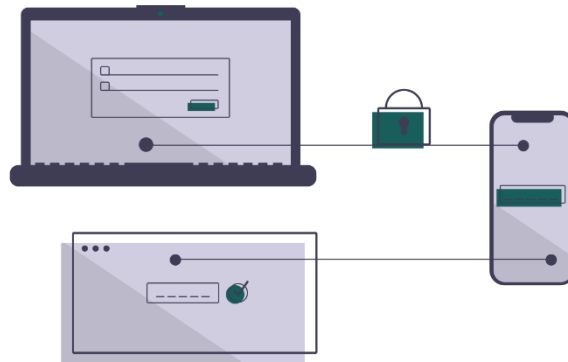


# Attacking Session Management and Authentication

Created by Colin McLean

Adapted by Jamie O'Hare

September 2023



**Aim:** This exercise is designed to delve into the essential concepts of web application session management and authentication. Throughout the exercise, you will be introduced to both manual techniques and advanced tools to achieve a thorough understanding of how to attack sessions and authentication mechanisms in target web applications.

**Note:** The information contained in this document is for educational purposes only.

If any link does not work, please attempt to use the Wayback Machine (linked below) to view the content prior to its inaccessibility.



The Wayback Machine

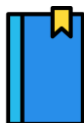
<https://archive.org/web/>



Caution!

Use of these tools and techniques found within may breach the Computer Misuse Act, therefore, ensure you have permission to interact with the target machines.

# 1 FANTASTIC TOOLS AND WHERE TO FIND THEM

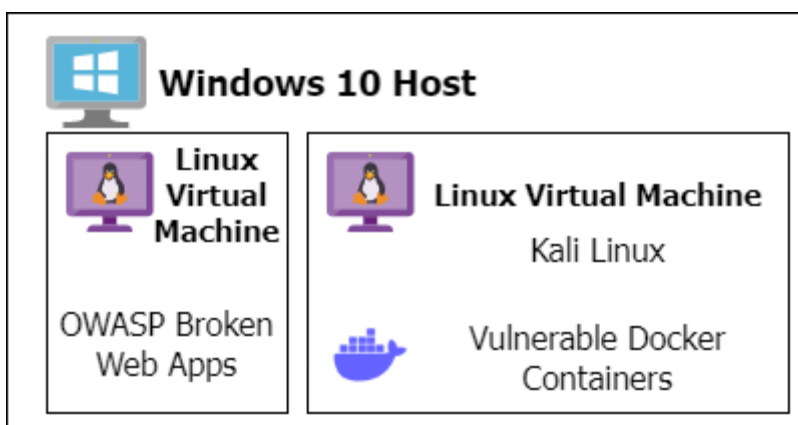


Please Note

This section is for reference.

## 1.1 LAB ENVIRONMENT

For this module's lab exercises, the network topology is as follows: A Windows 10 Host (192.168.1.254) operating 2 Linux Virtual Machines. One VM is running an altered version of the OWASP Broken Web Apps bundle (192.168.1.100), and another is a boutique installation of Kali Linux with several tools preinstalled, and various vulnerable docker containers (192.168.1.253). The figure below contains a visual representation of this topology. The following table presents the credentials of each system.



System	Username	Password
Windows 10 Host	admin	netlab
OWASP Broken Web Apps	root	owaspbwa
Kali Linux	kali	kali

### 1.1.1 Vulnerable Docker Containers

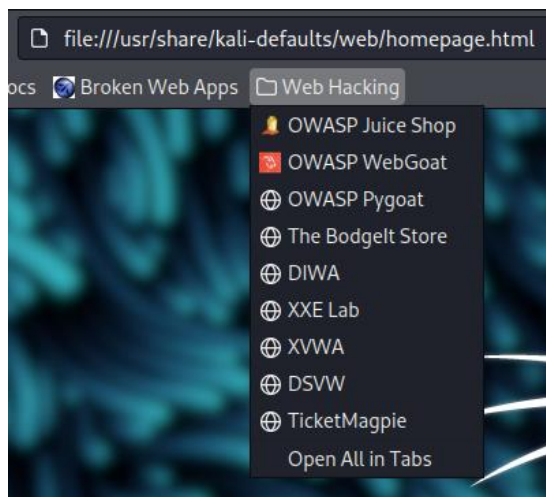
While some vulnerable applications exist on an isolated Virtual Machine, some are much closer to home! Through the power of containerisation (thanks to Docker!), there are multiple vulnerable web applications available on the Kali Linux Virtual Machine. You can find the corresponding docker-compose files in the docker\_targets directory found on the Desktop, as shown in the figure below.

```
(kali㉿kali)-[~/Desktop]
$ tree
.
├── docker_targets
│   ├── ag_targets
│   │   └── docker-compose.yml
│   ├── api_hacking
│   │   └── docker-compose.yml
│   └── web_hacking
│       └── docker-compose.yml
└── ...
```

To get these targets to run, use the command `docker-compose up`, when in the same directory as the `.yml` file, as seen below.

```
(kali㉿kali)-[~/Desktop/docker_targets/web_hacking]
$ sudo docker-compose up
[sudo] password for kali:
```

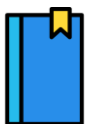
Afterwards, you can find links to these targets on Firefox’s bookmarks bar.



## 1.2 WINDOWS TOOLS

---

While there are not many tools which we will run on the Windows 10 host, there may be the odd time we do. You can find these in the “tools” folder.



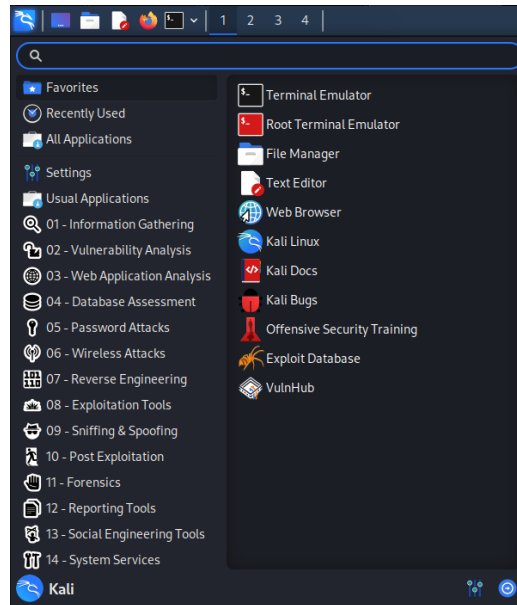
### Take note!

You may come across references to a “tools” folder, this is in the University’s lab environment and accessible there via the URI `\\hannah\tools\pentest`

## 1.3 KALI TOOLS

---

In contrast to Windows, there will many tools found installed on Kali which we will make ample use of. One way to find these tools is through Kali's search bar found by clicking the top left Kali logo icon. Alternatively, tools can be found through the command line interface.



### 1.3.1 What if the tools are not there?

If you are looking for a tool and it's not there, just install it! These machines are sandboxes for you do as you wish. This may require you to get familiar with some of the different installer packages such as apt-get install or go get. However, don't threat as there is plenty of tutorials out there to learn from.

# Contents

---

1	Fantastic Tools and Where to Find Them .....	2
1.1	Lab Environment .....	2
1.1.1	Vulnerable Docker Containers .....	2
1.2	Windows Tools .....	3
1.3	Kali Tools .....	4
1.3.1	What if the tools are not there? .....	4
2	Session Management & Authentication .....	7
2.1	Weakness in the Generation of Session Tokens.....	7
2.1.1	Meaningful Tokens.....	7
2.1.2	Common Encoding .....	8
2.1.3	Predictable Tokens.....	8
2.2	Weakness in Session Token Handling.....	10
2.2.1	Sniffing Tokens .....	10
2.2.2	Disclosure of Token in Log Files .....	11
2.2.3	Vulnerable Session Termination .....	11
2.3	A Sample of Countermeasures .....	11
3	Reverse engineering cookies .....	13
3.1	Decoding Cookies Using CyberChef.....	14
3.2	Decoding badstore's Cookies .....	15
3.3	Mutillidae 2 reversing cookie challenges .....	16
3.4	OWASP WebGoat Challenges .....	17
3.5	On-line Cookie Challenge .....	19
3.6	Cookie Analysis using Webscarab.....	19
4	Password cracking.....	25
4.1	Using OWASP ZAP.....	25
4.2	Using Burp Suite .....	27
4.3	Using Hydra .....	30
4.4	Using Medusa .....	31
5	Attacking Authentication Mechanisms.....	32
5.1	bWAPP Session Management Challenges .....	32

5.2	Abusing the Reset Password Functionality on OWASP Juice Shop .....	33
5.3	Various WebGoat Challenges .....	34
5.3.1	Authentication Bypasses .....	35
5.3.2	Password Resets.....	36
5.3.3	Secure Passwords.....	38
5.4	Bruteforcing One Time Passwords on PyGoat.....	39
5.4.1	With Burp Suite .....	41
5.4.2	With OWASP ZAP .....	42
6	Writing Vulnerability Reports .....	44
6.1	Scenario 1 - An Unprotected Login Page .....	44
6.2	Scenario 2 - Guessable Coupon Codes .....	44
6.3	Scenario 3 - Insufficient Session Expiration on Mobile Site .....	45
7	Appendices – optional exercises.....	46
	Appendix A - Using JHjack to Hijack a Session .....	46
	Appendix C - XVWA Challenges.....	48

## 2 SESSION MANAGEMENT & AUTHENTICATION

The session management mechanism (normally cookies) is a fundamental security component in most web applications. It enables the applications to uniquely identify a given user across several different requests, and to handle data that it accumulates about the state of that user's interaction with the applications. Where an application implements login functionality, session management is of particular importance, as it enables the applications to persist its assurance of any given user's identity beyond the requests in which they supply their credential.

When discussing session management vulnerabilities, for the most part, they largely fall into two distinct categories.

1. Weakness in the generation of session tokens.
2. Weakness in the handling of session tokens throughout their lifecycle.

### 2.1 WEAKNESS IN THE GENERATION OF SESSION TOKENS

---

#### 2.1.1 Meaningful Tokens

Some session tokens are created using a transformation of the username (Remember token must be unique). This information is then encoded in some way and may be combined with other data. Example:

**757365723d636f6c696e3b6170703d757365723b646174653d31302f30392f3135**

Convert from Hex to ASCII

**user=colin;app=user;date=10/09/15**

This can be exploited to attempt to guess the current sessions of other application users. This can be accomplished by enumerating usernames, then generate large numbers of potentially valid tokens, finally testing them with a cookie editor. Tokens that contain meaningful data often exhibit some structure (delimited) as in the previous case. Could user=colin;app=admin;date=10/09/15 work?

Additionally, components that may be encountered within structured tokens include:

- The account username.
- The numeric identifier used by the application to distinguish between accounts.
- The user's first/last human name.
- The user's email address.
- The user's group or role within the application.
- A date/time stamp.
- An incrementing or predictable number.
- The client's IP address.

Why? One reason – ease! It is easier to code, decode and extract components than performing a cookie lookup.

Easy but Insecure Method	Harder but Secure Method
Cookie arrives.	Cookie arrives.
Decode & Extract username, role (admin/user/guest) etc	Look up user, role etc.
Do stuff based on this.	Do stuff based on this.

### 2.1.2 Common Encoding

Encoding schemes that are commonly encountered include:

- Hexadecimal representation using ASCII characters.
- XOR
- Base64
- MD5
- ROT

Some encoding can be determined from characteristics and characters of each encoding scheme. An example of this is '=' characters which is often a clue of the use of Base64. These are all used because they are simple to program, and thus quite often combined. Example:

**MD5(Base64(token))**

Therefore, it is (mostly) necessary to test various decoding and variations.

### 2.1.3 Predictable Tokens

Some session tokens do not contain any meaningful data associating them with a particular user but are guessable as they contain sequences or patterns that allow an attacker to extrapolate which a Hacker can gather a sample and guess. This extrapolation may involve an amount of trial and error, for example, one valid guess per 1,000 attempts.

Predictable session tokens commonly arise from three different sources:

1. Concealed sequences
2. Time dependency
3. Weak random number generation

The following sections look at real-life examples of each of these sources.

#### 2.1.3.1 Concealed Sequences

The developer has used a previously seen method like MD5(Base64(token)). Therefore, to get the token one must crack MD5 then decode Base64. Similar methods often encountered include ASCII to Hex or simple arithmetic such as the next token = last token + 355. Real-life example follows, consider the following series of session token values:



Original Token	Base64 Decoded	ASCII to Hex	Subtract Previous Value
lwjVJA	--Ö\$	9708D524	FF97C4EB6A
Ls3Ayg	.ÍÀŽ	2ECDC08E	97C4EB6A
xpKr+A	Æ'«ø	C692ABF8	FF97C4EB6A
XleXYg	^W-b	5E579762	FF97C4EB6A
9hyCzA	ö,Ì	F61C82CC	97C4EB6A
jeFuNg	?án6	8DE16E36	FF97C4EB6A

The algorithm used to generate tokens adds 0x97C4EB6A to the previous value, truncating the result to a 32-bit number, and Base64 encodes, allowing for it to be sent across HTTP protocol. The formula for this would be:

$$\text{Cookie} = \text{Base64} (\text{truncated} ((\text{token} + 0x97C4EB6A))$$

This can be scripted to produce the series of tokens that will be produced next, allowing the hacker to hijack the session.

### 2.1.3.2 Time Dependency

Some applications employ algorithms for generating session tokens that use the time of generation, although to the eye they may “appear” to be completely random. Tying session tokens may allow hackers to predict other users’ tokens. A scripted attack may succeed in identifying large numbers of other users’ tokens. An example follows:

16:00 Sample	16:15 Sample
3124538-1172764258718	3124553-1172764800468
3124539-1172764259062	3124554-1172764800609
3124540-1172764259281	3124555-1172764801109
3124541-1172764259734	3124556-1172764801406
3124542-1172764260046	3124557-1172764801703

Breaking the string into variables separated by the dash. In the 16:00 sample it appears, the 1<sup>st</sup> digit appears to increment by one, while the 2<sup>nd</sup> digit varies. 15 minutes later it, the 1<sup>st</sup> digit has increased by over 10, perhaps this is the number of unique logins in that time? While the 2<sup>nd</sup> digit is over 500,000, perhaps this is milliseconds? A Hacker can use Jhijack to accomplish hijacking via this deduction.

### 2.1.3.3 Weak Random Number Generation

Random number generators use algorithms to create sequences of numbers. If you know the Pseudo Random Number Generator, this is used to guess a number, perhaps with some automation. Additionally, some PRNGs are predictable and thus vulnerable.



Randomly failed! Weaknesses in Java Pseudo Random Number Generators (PRNGs)  
<https://armoredbarista.blogspot.com/2013/03/randomly-failed-weaknesses-in-java.html>



Insufficient Entropy for Random Values  
<https://phpsecurity.readthedocs.io/en/latest/Insufficient-Entropy-For-Random-Values.html>

## 2.2 WEAKNESS IN SESSION TOKEN HANDLING

---

If tokens are not handled and stored properly then the generation is irrelevant. If tokens are disclosed to an attacker via some means, then the attacker can hijack user sessions.

### 2.2.1 Sniffing Tokens

Using a sniffer such as CAIN or Wireshark, along with ARP poisoning tokens can be sniffed across a network. In the simplest case, unencrypted HTTP connection for communications allows for an attacker on the same network, or somewhere in between source and destination, to read all the packet details including login credentials. A more sophisticated attacker would use ARP Poisoning to accomplish this.



ARP Spoofing/Poisoning  
[https://www.wikiwand.com/en/ARP\\_spoofing](https://www.wikiwand.com/en/ARP_spoofing)

One countermeasure to sniffing tokens would be to use HTTPS as it would be encrypted therefore cannot be read, but implementation can often be poor. Some applications use HTTPS to protect user's credentials during login, but then revert to HTTP, therefore, the cookie is sent in clear text after authentication. Similarly, some applications use HTTP for the front page, but switch to HTTPS from the login page onwards, but the cookie is created on the first page.

Another problem arises from when a HTTP page is used but the request is sent using HTTPS protocol. This problem resulted in a famous tweet/quote from Troy Hunt. "Your logon form posts to HTTPS, but you blew it when you loaded it over HTTP". This is due to the possibility of an attack performing a MITM attack when the HTTP page is loaded, i.e., sending a keylogger along with the page, as seen in the following video.



Your logon form posts to HTTPS, but you blew it when you loaded it over HTTP  
<https://www.youtube.com/watch?v=nm-85-bDP6c>



Here is why you can't trust SSL logos on HTTP pages (even from SSL vendors)

<https://www.youtube.com/watch?v=Z3aCT4FAuYA>

Even if the application issues a fresh cookie after login and uses HTTPS from the login page onwards. Tokens can be disclosed if the user revisits any HTTP page (common are Help or About). The application may attempt to switch to HTTPS when the user clicks the login link, however, it may still accept a login over HTTP if the user modifies the URL accordingly (social engineering email?).

### 2.2.2 Disclosure of Token in Log Files

A cause of session tokens appearing in system logs is where an application uses the URL query string for transmitting tokens. Example:

<https://www.webjunction.org/do/Navigation;jsessionid=F27ED2A6AAE4C6DA409A3044E79B8B48?category=327>

These tokens will appear in various system logs to which unauthorized parties may have access, for example:

- Users' browser logs.
- Web Server logs.
- Logs of corporate or ISP proxy servers.
- Logs of proxies "down-the-line".

### 2.2.3 Vulnerable Session Termination

Some applications do not enforce effective session expiration. A session may remain valid after the last request is received. An attacker may be able to capture the tokens of users who has accessed the application in the recent past. This can be accomplished locally on a machine by browsing to C:\Users\Colin\AppData\Local\Microsoft\Windows\Temporary Internet Files (or similar) and grabbing the cookies! Even worse is the possible case, when a user clicks "Logout", the server performs no action whatsoever.

## 2.3 A SAMPLE OF COUNTERMEASURES

---

The secure generation, transmission, and termination of cookies and other session tokens is paramount to web application security. To achieve this, practitioners should look to adopt the following countermeasures.

- Strong Token Generation.
- Only transmitted over HTTPS.
- Secure and Robust logout and deauthentication functionality.
- Tokens should be timed out.
- Single logins (no concurrency)
- Per page tokens (as seen in high street banking)

You may wish to look at the following resources for more substantial countermeasures.



Secure Session Management with Cookies for Web Applications

<https://crypto.stanford.edu/cs142/papers/web-session-management.pdf>




OWASP Application Security Verification Standard (2014)

[https://www.owasp.org/images/5/58/OWASP\\_ASVS\\_Version\\_2.pdf](https://www.owasp.org/images/5/58/OWASP_ASVS_Version_2.pdf)

### 3 REVERSE ENGINEERING COOKIES

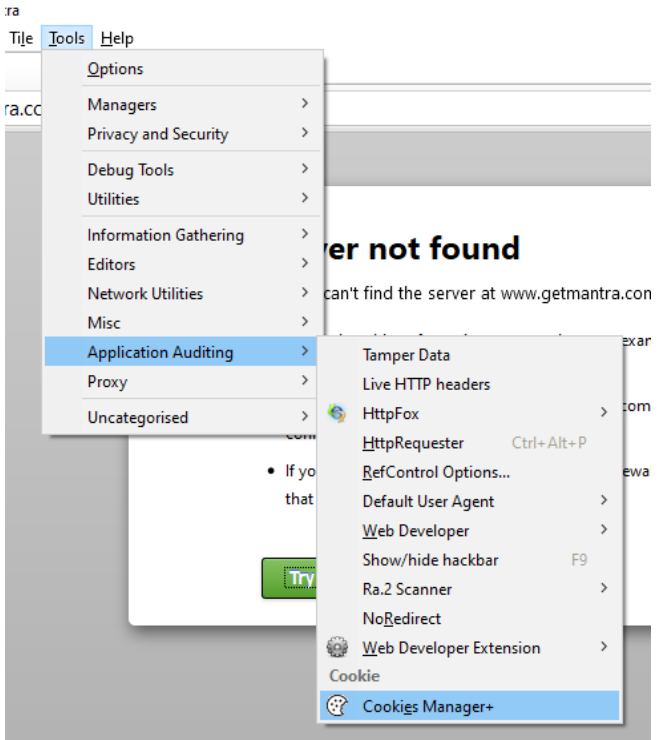
The following exercises focus on the reverse engineering of cookie values. This may include trying to determine the generation pattern and being able to predict values or reversing the entire value and making the information human readable.

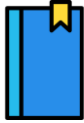
Across many of the exercises you will need to use a Cookie inspector/editor tool. You have several choices.



**Take note!**

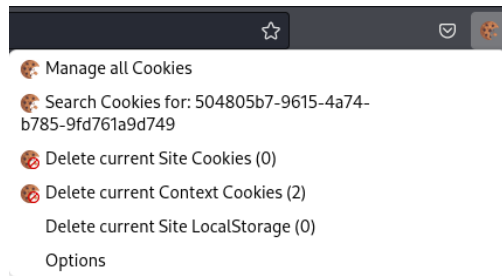
**Cookies manager+** is installed under OWASP Mantra (D drive in the specialist labs).





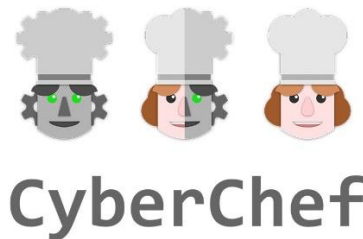
### Take note!

On the Firefox browser under Kali, there is a **Cookie Editor** plugin installed. Next to the URL bar, you will find a cookie icon as seen in the Figure below.



## 3.1 DECODING COOKIES USING CYBERCHEF

CyberChef is an open-source tool created by GCHQ, which allows values to be coded and decoded in many ways. We can use this tool to decode encoded values such as Cookies.



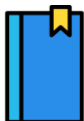
The CyberChef tool accessible through GCHQ's GitHub hosted Website

<https://gchq.github.io/CyberChef/>

- Use CyberChef to decode the following cookies.

53 47 56 73 62 47 38 67 64 47 68 6c 63 6d 55 3d

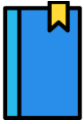
NzlgMTAxIDewOCAxMDggMTExIDMyIDExNiAxMDQgMTAxIDExNCAxMDEgMzI=



### Take note!

Experiment with CyberChef - it may be useful for your coursework!

## 3.2 DECODING BADSTORE'S COOKIES



### Take note!

This exercise uses the badstore application found on the Broken web apps VM.

- Using a browser that will give cookies (e.g. Mantra), browse to the badstore application located on the broken web apps VM, found at the URL below.

`http://192.168.1.100/badstore/htdocs/index.html`

- Create a user, perhaps use simple credentials such as **user** with a password of **user**, such as seen in the Figure below.

### Register for a New Account

Full Name:

Email Address:

Password:

Password Hint - What's Your Favorite Color?:

*(The Password Hint is used as a security measure to help recover a forgotten password. You will need both your email address and this hint to access your account if you forget your current password.)*

- Once authenticated as this new user, examine the cookies created using a cookie inspector/editor. Of particular interest is the **SSOid** cookie. It should look like the cookie pictured in the Figure below.

### Cookies

**SSOid:**dXNlckB1c2VyLmNvbTplZTEyY2JiMTkwNTJlNDBiMDdhYWwY2Ew...  
%3D%0A

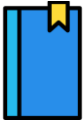


### Challenge!

Decode the SSOid for the account you created.

Note that it does not have == at the end, typical of Base64 encoding.

### 3.3 MUTILLIDAE 2 REVERSING COOKIE CHALLENGES



#### Take note!

This exercise uses the Mutillidae 2 application found on the Broken web VM.

- Browse to the Mutillidae 2 application located on the Broken web VM, found at the URL below.

`http://192.168.1.100/mutillidae/index.php`

- Register an account with the credentials as **bob / bob**. Then login as this new user. Examining the cookies created using a cookie inspector/editor. The cookies found should resemble the Figure provided below.

#### Cookies

PHPSESSID:6845mei6se8fticmhg798i1vv3
showhints:1
username:bob
uid:25

- Next register a user with the credentials **sam / sam**, then examine this new set of cookies. They should resemble those found in the Figure below.

#### Cookies

PHPSESSID:6845mei6se8fticmhg798i1vv3
showhints:1
username:sam
uid:26



#### Challenge!

Reverse engineer the cookie logic, then use a Cookie inspector/editor to successfully authenticate as the user Bob.



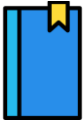
For Reference and Solution.

<https://www.youtube.com/watch?v=jDazgB7N9i8>



## 3.4 OWASP WEBGOAT CHALLENGES

---

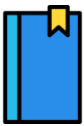


### Take note!

This exercise uses the OWASP WebGoat application found on the Broken web VM.

- Browse to the OWASP WebGoat application located on the Broken web VM, found at the URL below.

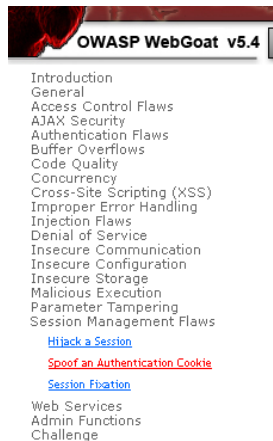
`http://192.168.1.100/WebGoat/`



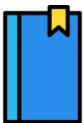
### Take note!

The OWASP WebGoat credentials are **guest / guest**.

- Select the Session Management Flaws from the Menu on the left-hand side and choose the Spoof an Authentication Cookie challenge.



- On this **Session mangement flaws/Spoof an Authentication Cookie page**, login as the webgoat user.



### Take note!

The user webgoat credentials are **webgoat / webgoat**.

Solution Videos Restart this Lesson

The user should be able to bypass the authentication check. Login using the webgoat/webgoat account to see what happens. You may also try aspect/aspect. When you understand the authentication cookie, try changing your identity to alice.

Sign in  
Please sign in to your account. See the OWASP admin if you do not have an account.

\*Required Fields

\*User Name

\*Password

**ASPECT SECURITY**  
Application Security Experts

OWASP Foundation | Project WebGoat | Report Bug

- After authenticating with the webgoat user, inspect the cookie created. Take a closer look at the **AuthCookie** created when authenticating on this exercise page. It should resemble something like the Figure provided below.

Cookies	
JSESSIONID	B6F8E7F9E2C1FE3BACD140589FCBE29D
AuthCookie	65432ubphcfx

- Return to the exercise page, and logout. Then login as the user **aspect/aspect** and examine the generated cookie. It should resemble something like the Figure provided below.

Cookies	
JSESSIONID	B6F8E7F9E2C1FE3BACD140589FCBE29D
AuthCookie	65432udfqtb

- You should notice similarities between the cookie for both users.



### Challenge!

Reverse engineer the cookie logic to successfully authenticate as the user **Alice**.



For Reference and Solution.

<https://www.youtube.com/watch?v=A93wniN1T7w>



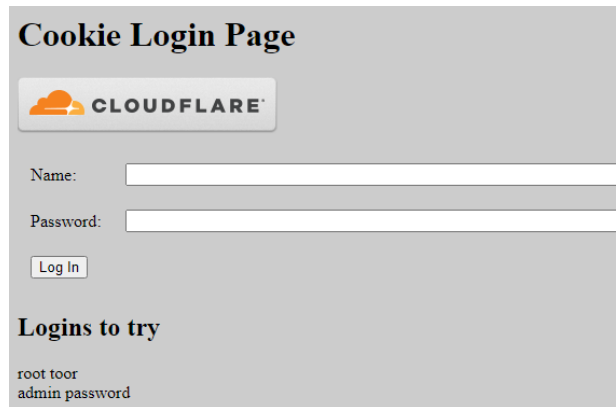
### Supplementary Content

Further session management exercises, including those available on the more modern version of the WebGoat application, are found in Appendix.

### 3.5 ON-LINE COOKIE CHALLENGE

---

Try to solve the following challenge.



The screenshot shows a web page titled "Cookie Login Page". At the top, there is a Cloudflare logo. Below it, there are two input fields: "Name:" and "Password:". A "Log In" button is positioned below the password field. Underneath the button, the text "Logins to try" is displayed, followed by two lines of text: "root toor" and "admin password".

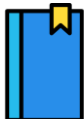


#### Challenge!

<https://attack.samsclass.info/cookielogin/>

### 3.6 COOKIE ANALYSIS USING WEBCARAB

---



#### Take note!

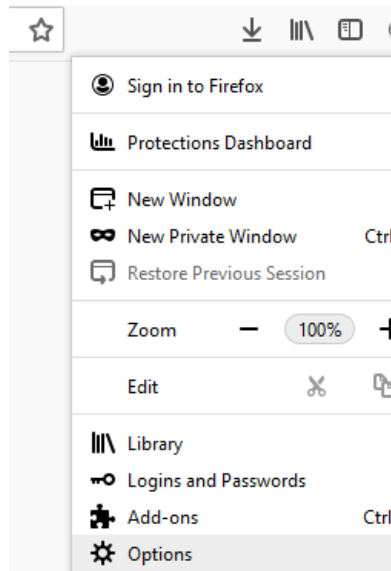
This exercise uses the WackoPICKO application found on the Broken web apps VM.

WebScarab was one of the original proxies, however, has since fell out of favour but it still has some novel uses. For this following exercise, we will be exploring cookie analysis using webScarab.

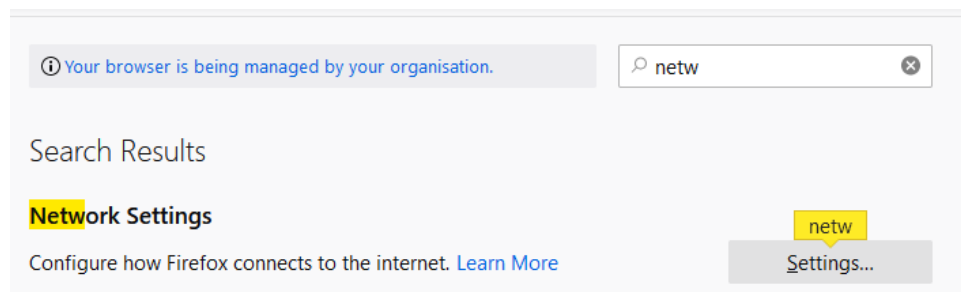
- Under Kali, run **WebScarab** by entering the following on a terminal You may have to install it).

`webScarab`

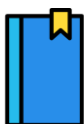
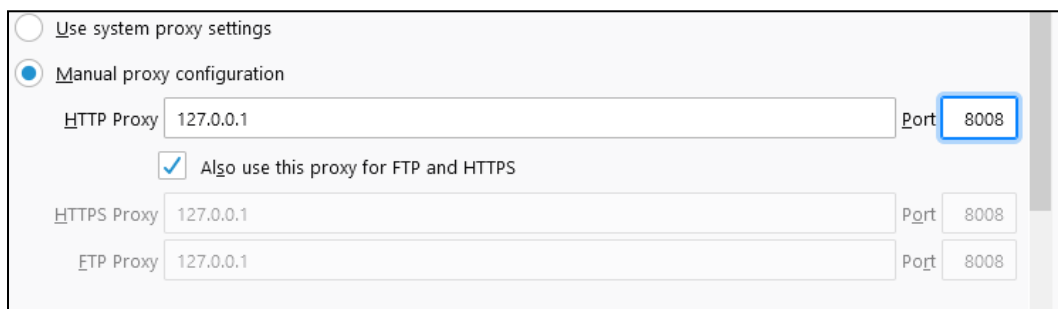
- Afterwards, **configure the Firefox browser under Kali** to point towards webScarab's proxy location.
- I.e. From Kali's Firefox browser, change the settings to point towards ZAP's proxy. (Look for **options**)



- Then search for **network**



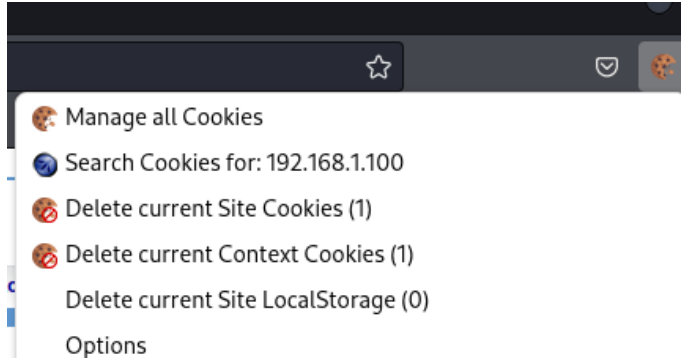
- Then configure the settings



**Take note!**

Webscarab's proxy is located at port **8008** - not 8080 as typical with OWASP ZAP and Burpsuite.

- Also, clear all the cookies etc from the cookies manager in the right hand top corner.



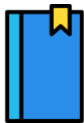
- After configuring the proxy correctly, navigate to the WackoPicko site found at the URL below.

<http://192.168.1.100/WackoPicko/index.php>

- Authenticate on the WackoPicko application as the user **bryce** (password **bryce**).
- Examine the cookies in **Webscarab Summary tab** and examine the **set-cookie** column. You should see a random Cookie ID being set similar to the snippet and Figure below.

PHPSESSID=75947cjqtjqnt10iomd8982ov1

13	11:31:40	GET	http://19... /WackoPl...	200 OK	Proxy		3135	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
12	11:31:40	GET	http://19... /WackoPl...	200 OK	Proxy		3482	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	PHPSESSI...
11	11:31:40	GET	http://19... /WackoPl...	301 Move...	Proxy		240	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
9	11:31:11	GET	http://19... /favicon.ico	200 OK	Proxy		3638	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	



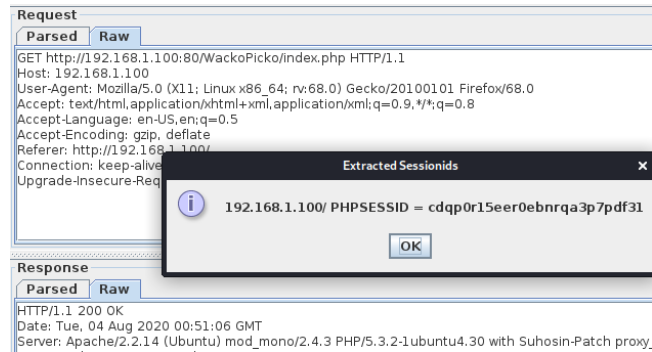
### Take note!

Note the ID number in the above case is 12 (yours will be different).

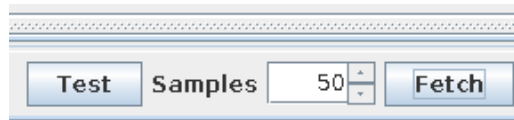
- Navigate to the **SessionID Analysis** tab and select this ID from the list of requests.

SessionID Analysis	Scripted	Fragments	Fuzzer	Compare	Search	SAML	OpenID	W
Summary	Messages		Proxy		Manual Request			
Collection	Analysis	Visualisation						
Previous Requests :								
From message body		11 - GET http://192.168.1.101:80/WackoPICKO 301 Moved Permanently						
Regex		12 - GET http://192.168.1.101:80/WackoPICKO/ 200 OK						
		13 - GET http://192.168.1.101:80/WackoPICKO/css/stylings.php 200 OK						
Request		14 - GET http://192.168.1.101:80/WackoPICKO/css/blueprint/print.css 200 OK						

- You may wish to examine the **RAW Request and Responses** to make the digests easier to see, as seen in the following Figure.



- Select **Test** and a pop up with a new **PHPSESSID** should appear as seen in the Figure above, and fetch 50 samples as seen in the Figure below.



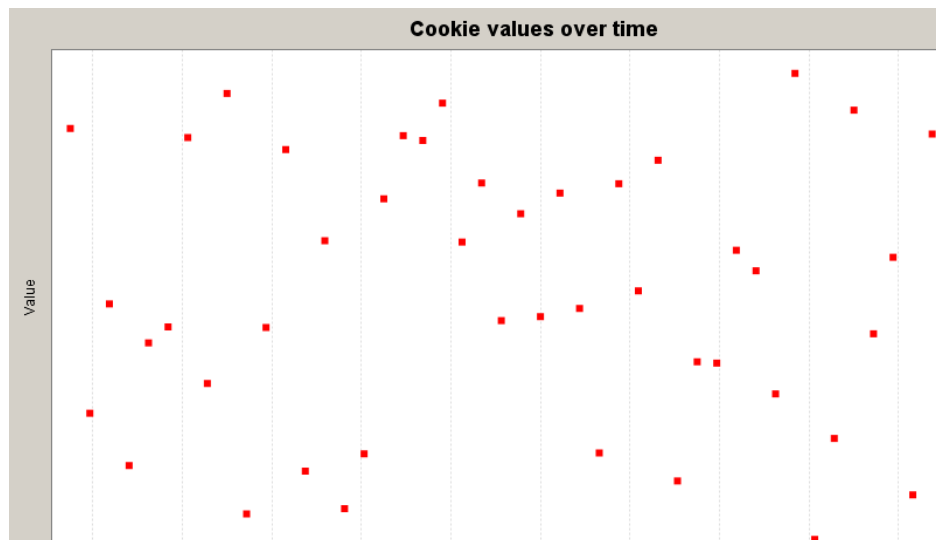
- Fetch doesn't appear to do anything but **Select the Analysis Tab** to view the cookies. Both steps are pictured in the following figures.

**NOTE** You must select the **Session identifier**.

Collection	Analysis	Visualisation
Session Identifier : 192.168.1.100/ PHPSESSID		
192.168.1.100/ PHPSESSID		
2023/01/16 07:05:54.697		

Collection	Analysis	Visualisation
Session Identifier : 192.168.1.100/ PHPSESSID		
Date	Value	
2023/01/16 07:05:54.697	9v8jv7tq0c5ubf0pu38f5k6l27	
2023/01/16 07:05:54.812	bn9q3bl1v9ndpk1a8bjtqf8m0	
2023/01/16 07:05:54.911	qvmsmc6i0lvos2u8piel0losn2	
2023/01/16 07:05:55.5	mncmonmbfavnsnn3d5ecalusl6	
2023/01/16 07:05:55.109	3nc98jhafghk0q8ld6f3b3o4j5	
2023/01/16 07:05:55.209	2f5lm1on5m0ld5rjt40ca30of1	
2023/01/16 07:05:55.318	sm957h9q6ravs9k7gvnha5end7	
2023/01/16 07:05:55.414	66q433ospc8ktak9qp2sph6j45	
2023/01/16 07:05:55.519	q14lt7odvfmlohb0qbmhdggke0	
2023/01/16 07:05:55.618	rrf2sie28ths45j35l9frtrae5	
2023/01/16 07:05:55.715	5fqaj50na6rpppajtq3n2g1j56	
2023/01/16 07:05:55.825	arc1mheoqof7m6t9ibrblglh14	
2023/01/16 07:05:55.920	atoe6hhour76hqcuo0t45mscbe4	
2023/01/16 07:05:56.20	r3fn79gdr68emk43kl9mruitn5	

- Then select the **visualization** tab to view the associated scatter graph (as shown below).



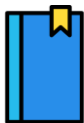
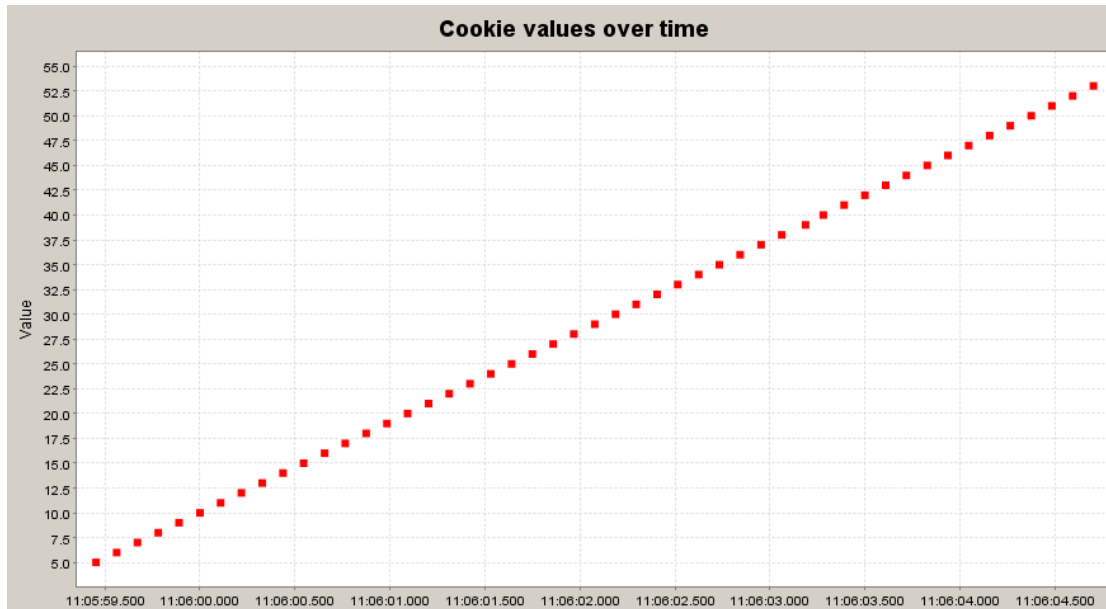
- Back to the browser, on the Wackopicko application select the **admin tab** at the bottom of the screen, as seen in Figure below, and log in as the **admin/admin**.

[Home](#) | [Admin](#) | [Contact](#) | [Terms of Service](#)

- Back on **webscarab**, you should see that authenticating with the admin credentials has created a second cookie named **session**.

ID	Date	Method	Host	Path	Para...	Status	Origin P...	XSS	CRLF	
32	2012/10/14...	GET	http://192.168.1.100:80	/Wacko...	?pag...	200 OK	Proxy	<input type="checkbox"/>	<input type="checkbox"/>	
31	2012/10/14...	POST	http://192.168.1.100:80	/Wacko...	?pag...	303 S...	Proxy	<input checked="" type="checkbox"/>	<input type="checkbox"/>	session=33
30	2012/10/14...	GET	http://192.168.1.100:80	/Wacko...	?pag...	200 OK	Proxy	<input type="checkbox"/>	<input type="checkbox"/>	

- Do a Session-ID analysis of this cookie, like you done above, and you should find that this is incrementing by one each time.



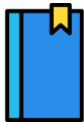
**Take note!**

The hack would be to create a cookie “session” with a predicted value and send it using Cookies Manager+, you will be logged in as admin.



For Reference.

<http://www.aldeid.com/wiki/WackoPicko/SessionID-vulnerability>



**Take note!**

While the possibility of reversing a modern application’s secrets is quite low, the underlying principles are applicable to other use against other tokens or logic.



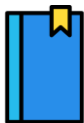
## 4 PASSWORD CRACKING

The following exercises focus on cracking passwords. Throughout this section, you'll make ample use of dictionary files. Kali comes preinstalled with several useful dictionaries. You can find most of the lists we will use in the `/usr/share/wordlists` directory, as shown in the Figure below.

```
(kali@kali)-[~]
$ ls -l /usr/share/wordlists
total 52108
lrwxrwxrwx 1 root root 26 May 31 08:52 amass -> /usr/share/amass/wordlists
lrwxrwxrwx 1 root root 25 May 31 08:52 dirb -> /usr/share/dirb/wordlists
lrwxrwxrwx 1 root root 30 May 31 08:52 dirbuster -> /usr/share/dirbuster/wordlists
lrwxrwxrwx 1 root root 41 May 31 08:52 fasttrack.txt -> /usr/share/set/src/fasttrack/wordlist.txt
lrwxrwxrwx 1 root root 45 May 31 08:52 fern-wifi -> /usr/share/fern-wifi-cracker/extras/wordlists
lrwxrwxrwx 1 root root 28 May 31 08:52 john.lst -> /usr/share/john/password.lst
lrwxrwxrwx 1 root root 27 May 31 08:52 legion -> /usr/share/legion/wordlists
lrwxrwxrwx 1 root root 46 May 31 08:52 metasploit -> /usr/share/metasploit-framework/data/wordlists
lrwxrwxrwx 1 root root 41 May 31 08:52 nmap.lst -> /usr/share/nmap/nmaplib/data/passwords.lst
-rw-r--r-- 1 root root 53357329 May 31 04:31 rockyou.txt.gz
lrwxrwxrwx 1 root root 39 May 31 08:52 sqlmap.txt -> /usr/share/sqlmap/data/txt/wordlist.txt
lrwxrwxrwx 1 root root 25 May 31 08:52 wfuzz -> /usr/share/wfuzz/wordlist
lrwxrwxrwx 1 root root 37 May 31 08:52 wifite.txt -> /usr/share/dict/wordlist-probable.txt
```

Here is some information about a subsection of the files.

- **common passwords** is small but is useful for simple passwords.
- **wordlist.txt** is a medium sized dictionary.
- **rockyou.txt** is a very large dictionary and would only be used as a last resort.
- **best15**, **best110** and **best1050** contain the top used passwords.

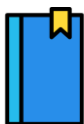


### Take note!

These files are actually symbolic links to their respective locations; however, you can still use this directory when importing them into tools.

### 4.1 USING OWASP ZAP

Automated password cracking can be an extremely effective method of hacking an application. While it is standard practice for applications to mitigate against this type of attack, mitigations can be circumvented if the security measures are not comprehensively implemented. It is somewhat common for automated cracking to remain possible in peripheral behaviour such as password reset or 2-factor authentication.



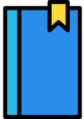
### Take note!

This exercise uses the WackoPicko application found on the Broken Web apps VM.

- Under Windows, run **OWASP Zap** as a proxy, and configure the **Mantra browser**.
- Navigate to the WackoPicko application at the URL provided below.

<http://192.168.1.100/WackoPicko/index.php>

- On the WackoPicko application, **navigate to the admin login page**.
- Attempt to authenticate on this page with the credentials try **admin / test**.



#### Take note!

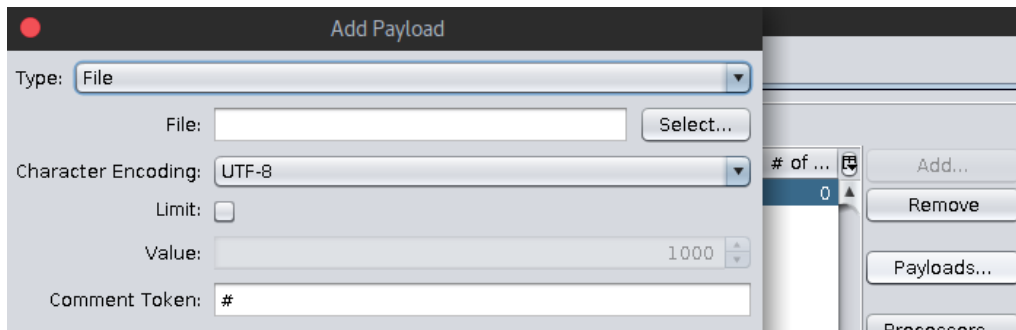
On the WackoPicko application, the admin login page is accessible from the link found at the bottom of the home page, as seen in the Figure below.

[Home](#) | [Admin](#) | [Contact](#) | [Terms of Service](#)

- Return to Zap and find the corresponding authentication request. It should resemble something like the Figure below.

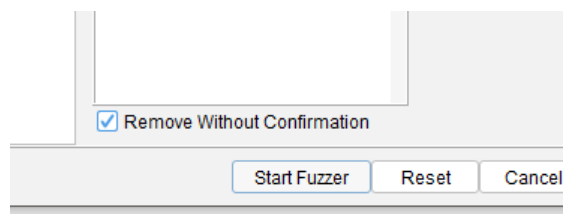


- Select **test** to highlight it and then **Right-click** on the word test and **choose Fuzz**.
- After selecting Fuzz, you will be prompted with a **Payloads** menu, from here add a file.



There are two dictionary files that you can use choose `/usr/share/wordlists/dirb/small.txt`

After importing the password file, start the fuzzer, and watch the responses come.



You are looking for an unusual response. Usually denoted with a different HTTP code, size, or response time, as seen in the Figure below.

- Click on the **Size Resp. Body** field to sort by size. You should see that only one response has a body size of 0. i.e. the password is **admin**.

Size Resp. Header	Size Resp. Body ^	Highest Alert	State	Payloads
0 bytes				admin
277 bytes		Medium		
277 bytes				12345
277 bytes				abc123
277 bytes			Reflected	password
277 bytes				computer

## 4.2 USING BURP SUITE

Like OWASP ZAP, Burp Suite can also be used to crack passwords. It is very similar to OWASP ZAP in this functionality, however, as we only have access to the Community Edition of Burp Suite it is not as effective as ZAP.

Start by closing OWASP ZAP, launching Burp Suite and intercept the same request to authenticate on the admin panel of WackoPicko. Right click this request and send it to Intruder. Navigate to Intruder, and ensure the password value test is the only value bookended by \$ character.

**Choose an attack type**

Attack type:

---

**Payload Positions**

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target:

```

1 POST /WackoPicko/admin/index.php?page=login HTTP/1.1
2 Host: 192.168.1.100
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 29
9 Origin: http://192.168.1.100
10 Connection: close
11 Referer: http://192.168.1.100/WackoPicko/admin/index.php?page=login
12 Cookie: session=5; PHPSESSID=5hvv84qi8lds5gflbh86laonq6
13 Upgrade-Insecure-Requests: 1
14
15 adminname=admin&password=5test5

```

The next step is to navigate to the Payloads panel and load the same password list as previously used. If you use john.lst you should remove the comments located at the start of the panel using the Burp Suite interface. After doing so, launch the attack!

**Payload Options [Simple list]**

This payload type lets you configure a simple list of strings that are used as payloads.

*Enter a new item*

#!comment: This list has been compiled by Solar Des...

#!comment: in 1996 through 2011. It is assumed to b...

#!comment:

#!comment: This list is based on passwords most co...

#!comment: systems in mid-1990's, sorted for decre...

#!comment: (that is, more common passwords are li...

#!comment: revised to also include common websit...

#!comment: of "top N passwords" from major com...

#!comment: occurred in 2006 through 2010.

#!comment:

After launching the attack, you will notice the requests trickling away. Take note of any response which is different from the rest. An abnormal response usually means we've cracked the password!

Request ^	Payload	Status	Error	Timeout	Length
0		200	<input type="checkbox"/>	<input type="checkbox"/>	813
1	123456	200	<input type="checkbox"/>	<input type="checkbox"/>	813
2	12345	200	<input type="checkbox"/>	<input type="checkbox"/>	813
3	password	200	<input type="checkbox"/>	<input type="checkbox"/>	813
4	password1	200	<input type="checkbox"/>	<input type="checkbox"/>	813
5	123456789	200	<input type="checkbox"/>	<input type="checkbox"/>	813
6	12345678	200	<input type="checkbox"/>	<input type="checkbox"/>	813
7	1234567890	200	<input type="checkbox"/>	<input type="checkbox"/>	813
8	abc123	200	<input type="checkbox"/>	<input type="checkbox"/>	813
9	computer	200	<input type="checkbox"/>	<input type="checkbox"/>	813
10	tigger	200	<input type="checkbox"/>	<input type="checkbox"/>	813
11	1234	200	<input type="checkbox"/>	<input type="checkbox"/>	813
12	qwerty	200	<input type="checkbox"/>	<input type="checkbox"/>	813
13	money	200	<input type="checkbox"/>	<input type="checkbox"/>	813

Request	Response
Pretty	Raw
7	Content-Type: application/x-www-form-urlencoded
8	Content-Length: 31
9	Origin: http://192.168.1.100
10	Connection: close
11	Referer: http://192.168.1.100/WackoPicko/admin/index.php?page=login
12	Cookie: session=5; PHPSESSID=5hvv84qi81ds5gflbh861aonq6
13	Upgrade-Insecure-Requests: 1
14	
15	adminname=admin&password=123456

? ⚙ ← →

16 of 3546

## 4.3 USING HYDRA

Hydra is a flexible password cracker, it can bruteforce user/password combinations for many different protocols. However, to use Hydra effectively, you will have to provide the tool with a good quality wordlist. Thankfully, Kali comes with some wordlists preinstalled.

```
(kali@kali)-[~]  
$ hydra -h  
Hydra v9.3 (c) 2022 by van Hauser/THC & David Maciejak - Please do not use in military  
llegal purposes (this is non-binding, these *** ignore laws and ethics anyway).
```

There are several dictionaries dedicated to individual cracking tasks. To use hydra to brute-force an attack, the following syntax can be used: -

```
hydra 192.168.1.20 http-form-post -l admin -P  
/usr/share/wordlists/passwords.txt  
"/processlogin.php:txtusername=^USER^&txtpassword=^PASS^:F=failedtext" -f  
-V
```

Where:

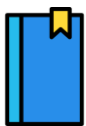
- **admin** is the username to bruteforce.
- **processlogin.php** is the page that the logon is submitted to.
- **txtusername** is the username variable being sent
- **txtpassword** is the password variable being sent.
- **F=failedtext** is a section of text that appears on the page if a request fails.
  - **S=successtext** can alternatively be used to provide text seen on a successful attempt.
- **-f** will stop when a valid account appears.
- **-V** means be verbose i.e. show us all the attempts.



### Challenge!

Starting the command below, get Hydra to crack WackoPicko's admin password.

```
hydra 192.168.1.100 http-form-post -l admin -P /usr/share/wordlists/dirb/small.txt  
"/WackoPicko/admin/index.php?page=login:adminname=^USER^&password=^PASS^:F=Admin Area"  
-f -V
```



### Take note!

You might want to check out these notes found on GitHub:

[https://github.com/gnebbia/hydra\\_notes](https://github.com/gnebbia/hydra_notes)

## 4.4 USING MEDUSA

---

Medusa is like Hydra, here's an example of how to run it.

```
(kali㉿kali)-[~]  
└─$ medusa  
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>
```

```
medusa -h 192.168.1.20 -u admin -P /usr/share/wordlists/passwords.txt -M  
web-form -m FORM:"processlogin.php" -m DENY-SIGNAL:"not found" -m  
FORM-DATA:"post?txtusername=&txtpassword="
```

Where:

- **-h** specifies the host
- **-u** specifies the username
- **-P** specifies the password list to use
- **-M** specifies the Module to execute.
- **-m** specifies the Module parameter to use. (This can be done multiple times as seen above)
- **DENY-SIGNAL:"not found"** specifies text found on a failed attempt's page
- **-f** will stop when a valid account appears



### Challenge!

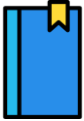
Starting the command below, get Medusa to crack WackoPicko's admin password.

```
medusa -h 192.168.1.100 -u admin -P /usr/share/wordlists/dirb/small.txt -M web-form -m  
FORM: "/WackoPicko/admin/index.php?page=login" -m DENY-SIGNAL:"Admin Area" -m FORM-  
DATA:"post?adminname=&password=" -f
```

# 5 ATTACKING AUTHENTICATION MECHANISMS

## 5.1 BWAPP SESSION MANAGEMENT CHALLENGES

---



### Take note!

This exercise uses the bWAPP application found on the Broken Web apps VM.

Browse to the bWAPP web application and try to solve their Session Management exercises found in **Section A2**.

<http://192.168.1.100/bWAPP/login.php>

```
Session Management - Administrative Portals
Session Management - Cookies (HTTPOnly)
Session Management - Cookies (Secure)
Session Management - Strong Sessions
Session Management - Session ID in URL
```

### 5.1.1.1 Administrative Portals Challenge



#### Challenge!

With the security level set to low, how do you get to the admin portal?



#### Challenge!

With the security level set to medium, how do you get to the admin portal?

### 5.1.1.2 Cookies (Secure) Challenge

Make sure to change the Security level back to low!



#### Challenge!

Identify what is wrong, and what risk this introduces.



### 5.1.1.3 Strong Sessions Challenge



#### Challenge!

Identify what is wrong, and what risk this introduces.

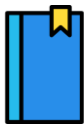
### 5.1.1.4 Session ID in URL Challenge



#### Challenge!

Identify what is wrong, and what risk this introduces.

## 5.2 ABUSING THE RESET PASSWORD FUNCTIONALITY ON OWASP JUICE SHOP



#### Take note!

This exercise uses the OWASP Juice Shop application found in the web\_hacking docker containers.

This exercise will look to at completing one of Juice Shop's 3 Star difficulty challenges named "Bjoern's Favorite Pet". The description of this challenge gives a clue where to start.

"Reset the password of Bjoern's OWASP account via the Forgot Password mechanism with the original answer to his security question"

To accomplish this task, start by navigating to the Forgot Password page, which is found at the link below.

<http://localhost:3000/#/forgot-password>

Here is a freebie, Bjoern's email is bjoern@owasp.org. Once you've supplied this email, you'll noticed the security question box will autofill with the security question "Name of your favorite pet?".

Thankfully, this information is easily obtainable through Bjoern's twitter linked below. Spend time scrolling through his account, and you'll find the name of a rather cute cat. Provide its name to answer

the security question and reset the password to complete the challenge. If you do not have a Twitter account, the name can be found in Bjoern's talk for BeNeLux Day 2018.



Bjoern's Twitter Account

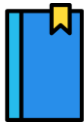
<https://twitter.com/bkimminich>



Bjoern's BeNeLux Day 2018 Juice Shop Talk

<https://www.youtube.com/watch?v=Lu0-kDdtVf4&t=239s>

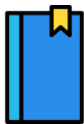
## 5.3 VARIOUS WEBGOAT CHALLENGES



### Take note!

This exercise uses the WebGoat application found in the web\_hacking docker containers.

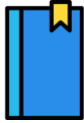
The (newer) WebGoat training application has many exercises relating to broken authentication. Different vulnerabilities are explored in each exercise. This section will explore the most relevant exercises. Once authenticated, you can find the relevant exercises from the menu on the left-hand side.



### Take note!

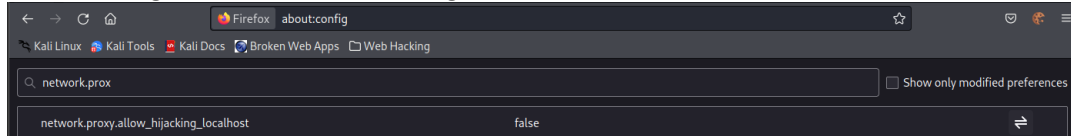
The user aspect credentials are **goats1 / shepherd1**.





### Take note!

In the following exercises, you may need to make use of a proxy, however, you may struggle with intercepting traffic destined for localhost. To solve this issue, follow these steps. On Firefox, browse to `about:config` and search for the `network.proxy.allow_hijacking_localhost` setting. It is currently configured to be false, set it to True using the button on the far right.



## 5.3.1 Authentication Bypasses

The first exercise to look at is Authentication Bypasses. Start by reading the information provided, then navigate to the next page via the arrow at the top.



### Authentication Bypasses

This new page has an exercise for you to do, hence the red colour in comparison to the grey. However, don't skip over the reading in this section. It includes some clues of how to accomplish the task.



### Challenge!

Successfully verify the account without verifying it - allowing you to change the password.

To start this task, populate the fields, accordingly, as seen in the Figure below. Then hit submit, however, you'll want to intercept this request with a proxy of your choice.

#### The Scenario

You reset your password, but do it from a location or device that questions are also stored on another device (not with you), as you have already provided your username/email and opted for

With the request intercepted, it should look like the Figure below. You'll want to send this to a feature which allows you to manipulate then forward the request. For Burp Suite this is Repeater. Of this request, the only part of interest is the functions found on line 18. This is where you want to edit.

```

1 POST /WebGoat/auth-bypass/verify-account HTTP/1.1
2 Host: 127.0.0.1:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0)
4 Gecko/20100101 Firefox/91.0
5 Accept: */*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
9 X-Requested-With: XMLHttpRequest
10 Content-Length: 98
11 Origin: http://127.0.0.1:8080
12 Connection: close
13 Referer: http://127.0.0.1:8080/WebGoat/start.mvc
14 Cookie: JSESSIONID=L1RcJZRZuL_BEQXBEIugWze9zr9mGAOdOUzkG-F;
15 csrfToken=
16 StZeG4hiWwRL0QvE7B4vMNNxtVyOGOTJbsE79BaLS27v8lLSVvPSenMKjQwKX;
17 language=en; welcomebanner_status=dismiss; cookieconsent_status=
18 dismiss; continueCode=
19 oWj7vSLVYb5K70nL0xNm2Mrvj4GLvABw1eE2zgXokMPaRyJ6QBp9EVqr2e
20
21 Sec-Fetch-Dest: empty
22 Sec-Fetch-Mode: cors
23 Sec-Fetch-Site: same-origin
24
25 secQuestion=Jamie&secQuestion1=Bel1+Street&jsEnabled=16
26 verifyMethod=SEC_QUESTIONS&userId=12309746

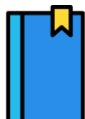
```

You will know if you edit it correctly if you receive a response found in the Figure below.

```

1 HTTP/1.1 200 OK
2 Connection: close
3 X-XSS-Protection: 1; mode=block
4 X-Content-Type-Options: nosniff
5 X-Frame-Options: DENY
6 Content-Type: application/json
7 Date: Thu, 22 Sep 2022 13:20:32 GMT
8
9 {
10   "lessonCompleted":true,
11   "feedback":
12     "Congrats, you have successfully verified the account without ac
13     tually verifying it. You can now change your password!",
14   "output":null,
15   "assignment":"VerifyAccount",
16   "attemptWasMade":true
17 }

```



### Take note!

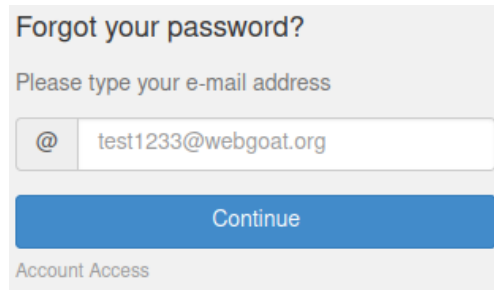
If you aren't making any progress, then use the hint function at the top of the exercise page.

## 5.3.2 Password Resets

The second relevant exercise looks Password Reset functionality. There is a fair bit of content in this section, with 7 pages, 4 of which are activities. However, don't worry as only one of them is intricate as the previous exercise. Start by reading the information provided, then navigate to the next page.



The first activity is found on page 2. This activity is to introduce the WebWolf application (also running when you launch web\_hacking Docker containers. WebWolf is an email client allowing activities to mimic a real-world application which emails you correspondence like password reset emails. For this exercise, start by providing your email ([goats1@webgoat.com](mailto:goats1@webgoat.com)) in the forgot your password functionality, and hitting continue.



Forgot your password?

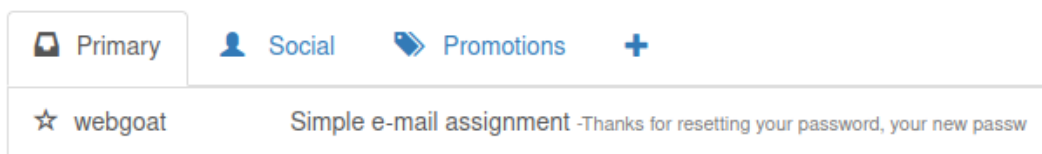
Please type your e-mail address

@ test1233@webgoat.org

Continue

Account Access

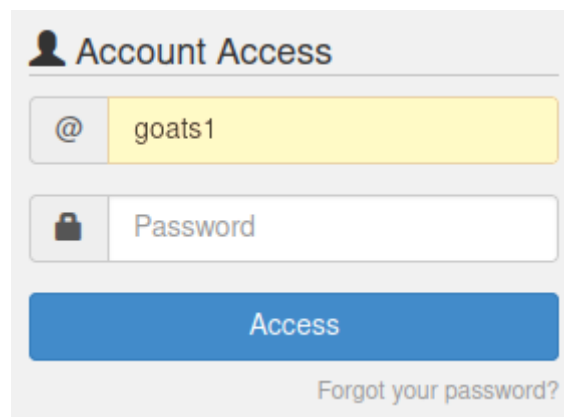
Next step is to navigate to the WebWolf application. There is a handy hyperlink just above the activity box. Inside the WebWolf application you will need to authenticate using the same credentials as the WebGoat application. Afterwards, navigate to the mailbox and you will be greeted with an email like the Figure below.



Primary Social Promotions +

☆ webgoat Simple e-mail assignment -Thanks for resetting your password, your new passw

This email details a new password for your account. Return to the original Account Access pane of the activity and submit your username, and new password to complete the exercise.



Account Access

@ goats1

Password

Access

Forgot your password?

Continue with the exercise, reading the information found within. The next activity is found on page 4. In this activity, you are tasked to retrieve the password of another user by accurately guessing their weak security question answer. The 3 accounts to guess are tom, admin, and larry. Populate the username accordingly, then try different colours for the answer. While you may want to try and bruteforce this using a proxy tool, that would be overkill as the the security question is very weak. Try and fill in the incomplete Table below.

**WebGoat Password Recovery**

**Your username**

**What is your favorite color?**

**Submit**

Username	Favourite Colour
tom	
admin	
larry	

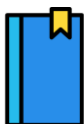
Continue with the exercise, reading the information found within. The next activity is found on the following page. This activity just requires you to look at two security questions. The application will provide reasoning on why these questions are suitable or impracticable.

In comparison the activity on page 6 requires more effort. In this activity you are tasked to reset Tom's password and then successfully authenticate as him. Rather than support you through this exercise, challenge yourself to complete the exercise yourself.



#### Challenge!

Authenticate as Tom



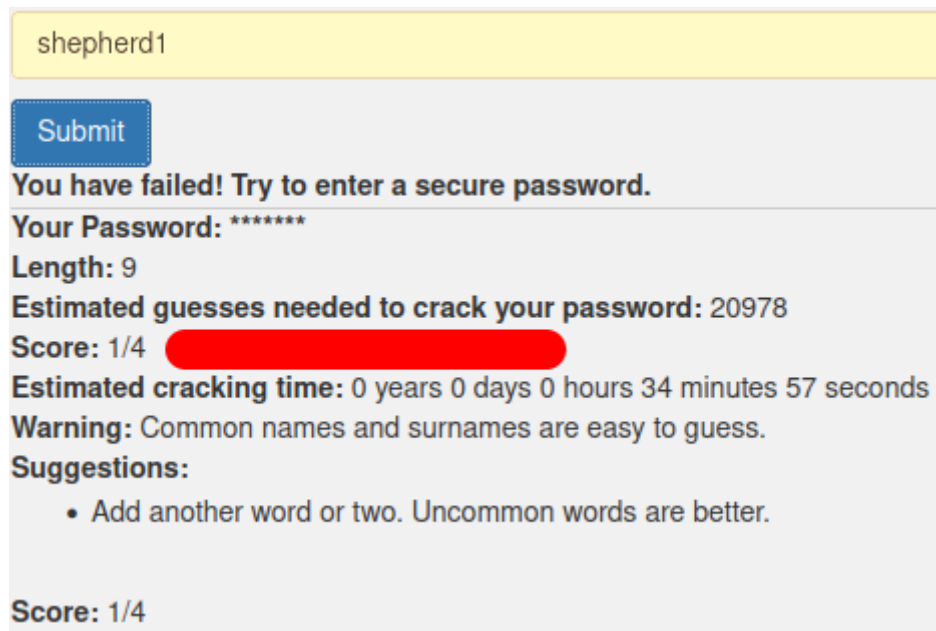
#### Take note!

If you require assistance, make use of the hints found at the top of page, or use google to find guides!

### 5.3.3 Secure Passwords

The last relevant exercise looks at Secure Passwords. There is only one activity in this section, located on page 4. However, don't skip over the rest. There is plenty of ace information found within, especially as the information is based on the NIST standards (US government guidance). Perhaps, compare this information with the most recent NCSC guidance (UK government).

The activity on page 4 is rather simple. The page presents a password cracking calculator, the task is to provide a strong password meeting the calculators' 4/4 standard. The Figure below presents an example using the user credentials for the application.



shepherd1

Submit

**You have failed! Try to enter a secure password.**

Your Password: \*\*\*\*\*

Length: 9

Estimated guesses needed to crack your password: 20978

Score: 1/4

Estimated cracking time: 0 years 0 days 0 hours 34 minutes 57 seconds

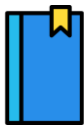
Warning: Common names and surnames are easy to guess.

Suggestions:

- Add another word or two. Uncommon words are better.

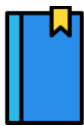
Score: 1/4

## 5.4 BRUTEFORCING ONE TIME PASSWORDS ON PYGOAT



### Take note!

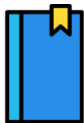
This exercise doesn't seem to work; however, I kept it in as the method behind the vulnerability is intriguing (and correct!). Feel free to just read the following section.



### Take note!

This exercise uses the PyGoat application found in the web\_hacking docker containers.

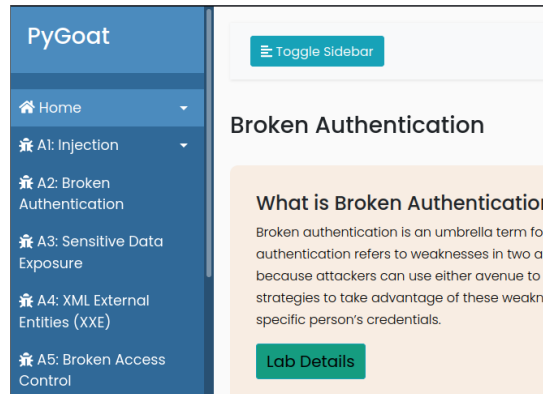
Navigate to the PyGoat Application, creating a new account or authenticating with the credentials found below.




### Take note!

PyGoat credentials are **goats/shepherd1**

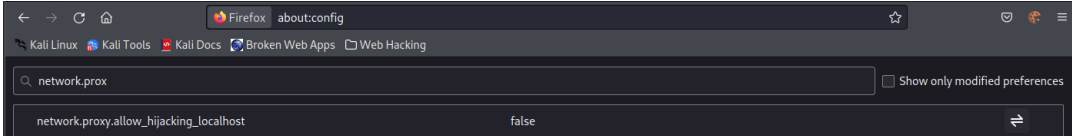
After authenticating, navigate to the Broken Authentication exercise. Make sure to read the information provided.



You will see that we must use a proxy to complete this exercise. PyGoat suggests Burp Suite. We will begin by using it; however, we will also touch on the alternative OWASP Zap. Before we get there though, we need to change a setting in Kali's Firefox to allow for proxying on Locally hosted applications. Refer to the box below for information.

**Take note!**

On Firefox, browse to `about:config` and search for the `network.proxy.allow_hijacking_localhost` setting. It is currently configured to be false, set it to True using the button on the far right.



After configuration, launch the lab, then start your proxy and configure the browser to use it. Once you've followed the instructions to the OTP login page, you'll be greeted with a page like below.

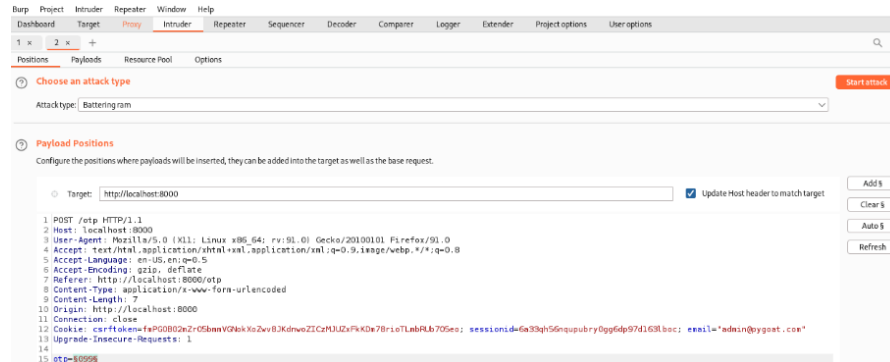
### Login Through Otp

Enter Your OTP:

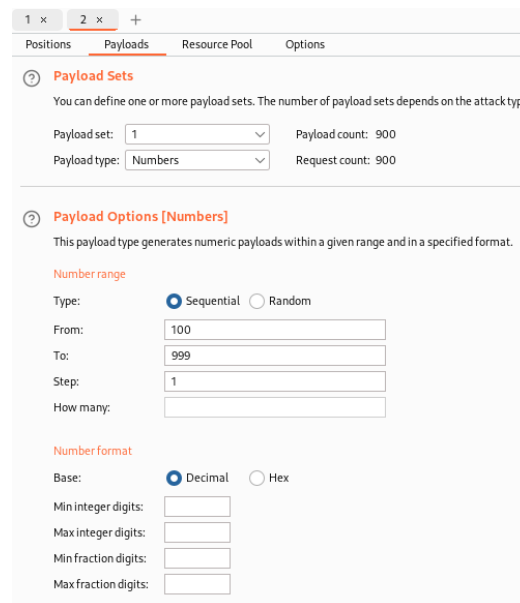


### 5.4.1 With Burp Suite

Insert the admin email (found on the page source) and a random 3-digit OTP value. Before clicking the Log In button, ensure Burp Suite is has enabled intercept. Once enabled, click login and you'll be greeted with the request on Burp Suite. Right click the request and send it to Intruder.



Navigate to the Intruder page, and you'll be greeted with a request like above. We only want to play around the otp value found on line 15. Therefore, if any other value bookended by the \$ characters should be removed via the buttons on the right-hand side of the pane. Afterwards head to the Payloads tab and configure the attack to use settings like those pictured below.



With everything set, start the attack! However, in doing so you'll be prompted saying Burp Suite community edition throttles intruder attacks. This is an unfortunate downside to using the community edition.



### Caution!

The max speed Burp Suite Community Edition allows means this step can take multiple hours! (Don't actually spend hours watching it, instead use OWASP ZAP).

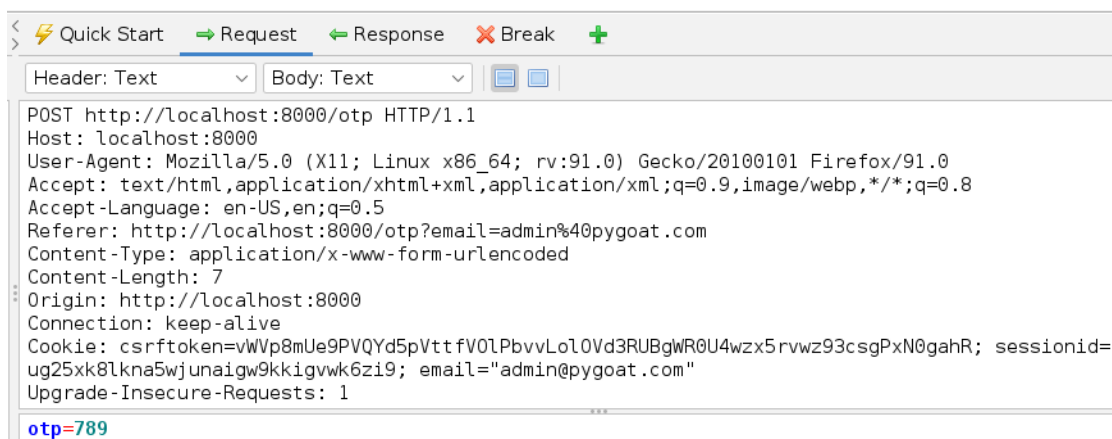
As the results trickle in, you should see output like below. To find the correct OTP, look for a different length. This is due to when we successfully authenticate, we should be greeted with a different page.

2. Intruder attack of http://localhost:8000 - Temporary attack - Not saved to project file						
Attack Save Columns						
Results Positions Payloads Resource Pool Options						
Filter: Showing all items						
Request ^	Payload	Status	Error	Timeout	Length	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	12904	
1	100	200	<input type="checkbox"/>	<input type="checkbox"/>	12904	
2	101	200	<input type="checkbox"/>	<input type="checkbox"/>	12904	
3	102	200	<input type="checkbox"/>	<input type="checkbox"/>	12904	
4	103	200	<input type="checkbox"/>	<input type="checkbox"/>	12904	
5	104	200	<input type="checkbox"/>	<input type="checkbox"/>	12904	
6	105	200	<input type="checkbox"/>	<input type="checkbox"/>	12904	
7	106	200	<input type="checkbox"/>	<input type="checkbox"/>	12904	
8	107	200	<input type="checkbox"/>	<input type="checkbox"/>	12904	
9	108	200	<input type="checkbox"/>	<input type="checkbox"/>	12904	
10	109	200	<input type="checkbox"/>	<input type="checkbox"/>	12904	
11	110	200	<input type="checkbox"/>	<input type="checkbox"/>	12904	
12	111	200	<input type="checkbox"/>	<input type="checkbox"/>	12904	
13	112	200	<input type="checkbox"/>	<input type="checkbox"/>	12904	
14	113	200	<input type="checkbox"/>	<input type="checkbox"/>	12904	

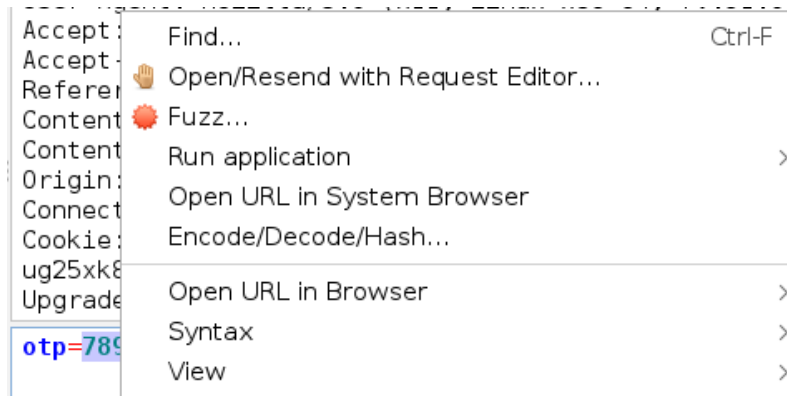
### 5.4.2 With OWASP ZAP

As seen above, Community Edition of Burp Suite throttles this type of attack. To perform the task faster, we can use OWASP ZAP as it does not throttle. This following section will outline how to use OWASP ZAP for this exercise.

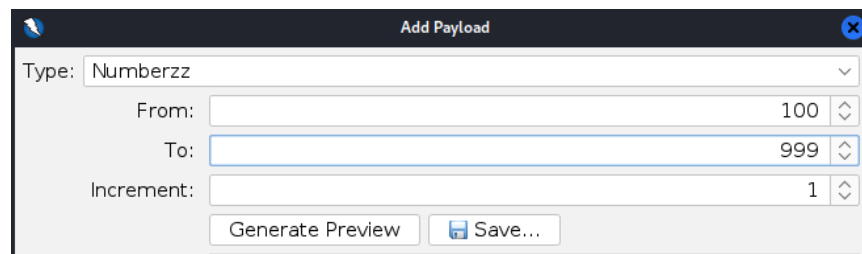
Close down Burp Suite and launch OWASP ZAP. Following the same steps shown above, however, translating them to the OWASP ZAP application. Start by intercepting the request we wish to modify, as seen in the Figure below.



Highlight the otp value, right click to reveal options. The one we want is Fuzz...



Add a new payload, using the Numberzz option, and fill in the fields accordingly. Then start the fuzzer. You will notice this is much faster than Burp Suite Community Edition. The solution is the response with a different body size.



## 6 WRITING VULNERABILITY REPORTS

Over the course of this exercise, you will have identified many different vulnerabilities. Many which could lead to severe consequences. However, finding these curiosities is only half the battle. The penetration tester or bug bounty hunter must then report this to the client.

Disclosing these vulnerabilities to the client can be difficult to write. Therefore, the following exercises provide you an opportunity to develop and refine your written communication skills.



### Consider...

Use this exercise to commence with independent personal research. Start by understanding each vulnerability, and then judge how best to present this to the respective client

### 6.1 SCENARIO 1 - AN UNPROTECTED LOGIN PAGE

---

As part of a penetration testing engagement of a local University, the login page for the retired research portal remains accessible. Thanks to efforts from the IT team towards system interoperability, this rather old login page works seamlessly with the modern infrastructure. However, this login page is not as hardened as others. Security measures such as an attempt limit, among others, are missing.



### Writing Exercise

Write a report detailing the security risk this unprotected login page poses, and the possible routes to resolving this issue.



### Take note!

It may be beneficial to ponder on how business critical the affected system is.

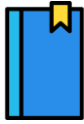
### 6.2 SCENARIO 2 - GUESSABLE COUPON CODES

---

To promote a new video game, the publisher and a wildly popular energy drink company have teamed up to provide exclusive in-game items to their customers. To claim in-game rewards, customers must enter codes found on the underside of can tab. From first impression, these 6-digit codes appear entirely random. However, after preserving with the small timeout feature when you enter incorrect codes, you have identified the codes can be easily reverse engineered and enumerated.

**Writing Exercise**

As a bug bounty hunter, write a vulnerability report for the issue identified above.

**Take note!**

While you may not think it, the vulnerability mentioned above has quite severe consequences!

## 6.3 SCENARIO 3 - INSUFFICIENT SESSION EXPIRATION ON MOBILE SITE

---

In a high-profile e-commerce website, an issue with insufficient session expiration was identified. Despite logging out, an access token used throughout the application is not expired properly. This is due to the log out process issuing many HTTP DELETE requests at once. One of these requests prematurely deletes a token required to invalidate the access token. As such, the access token remains active. This vulnerability online takes place on the mobile version of the e-commerce website, found at the m subdomain. While not pertaining to anything confidential, the sheer number of active access tokens may cause the e-commerce site issues.

**Writing Exercise**

Write a vulnerability report for the insufficient session expiration identified.

# 7 APPENDICES – OPTIONAL EXERCISES

## APPENDIX A - USING JHJACK TO HIJACK A SESSION

- From 192.168.1.100, run Webgoat (log in as **guest/guest**).
- Select **Hijack a session**.

The screenshot shows the OWASP WebGoat v5.4 web application. On the left is a sidebar with a list of topics: Introduction, General, Access Control Flaws, AJAX Security, Authentication Flaws, Buffer Overflows, Code Quality, Concurrency, Cross-Site Scripting (XSS), Improper Error Handling, Injection Flaws, Denial of Service, Insecure Communication, Insecure Configuration, Insecure Storage, Malicious Execution, Parameter Tampering, Session Management Flaws, Hijack a Session, Spoof an Authentication Cookie, Session Fixation, Web Services, Admin Functions, and Challenge. The 'Hijack a Session' option is highlighted. On the right, the 'Solution Videos' section contains text about session ID complexity. Below it, the 'General Goal(s):' section states 'Try to access an authenticated session belonging to some'. The 'Sign In' section has the prompt 'Please sign in to your account.', a note '\*Required Fields', and two input fields for '\*User Name:' and '\*Password:', followed by a 'Login' button.

Capture using WebScarab, look at the conversation where the **WEAKID** cookie is set.

- Note this conversation ID

187	2012/10/14...	GET	http://192.168.1.100:80	/WebGoat/css/layer...	304 N...	Proxy	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		WEAKID=1	
186	2012/10/14...	GET	http://192.168.1.100:80	/WebGoat/attack	?Scre...	200 OK	Proxy	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	acopendivids=swingset_jotto="", phpbb2=""...	JSESSION
185	2012/10/14...	GET	http://192.168.1.100:80	/WebGoat/images/...	304 N...	Proxy	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		JSESSION	

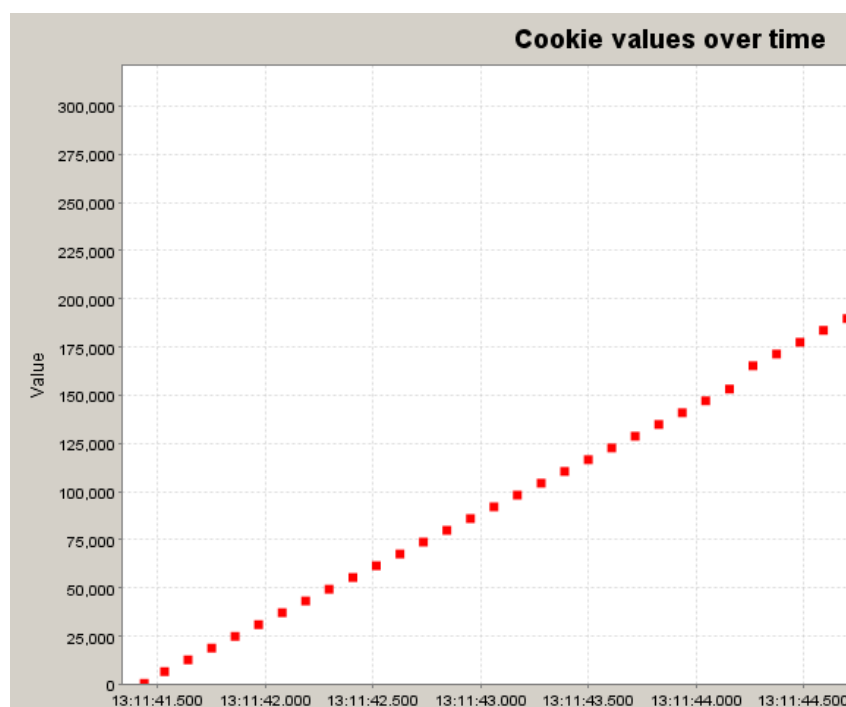
- In the SessionID analysis tab, select this conversation.

Previous	Next	186 - GET http://192.168.1.100:80/WebGoat/attack 200 OK
Parsed	Raw	
GET http://192.168.1.100:80/WebGoat/attack?Screen=452&menu=1800 HTTP/1.1 Host: 192.168.1.100 User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:12.0) Gecko/20100101 Firefox/12.0 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 Accept-Language: en-gb,en;q=0.5 Accept-Encoding: gzip, deflate Proxy-Connection: keep-alive Referer: http://192.168.1.100/WebGoat/attack?Screen=453&menu=1800 Cookie: JSESSIONID=B13873C105079D6A16B9F854E4186845; acopendivids=swingset,jott Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=		

- Hit the “Test” button to make sure that this is the correct ID.
- Now fetch 50 Samples

2012/10/14 13:39:50	19370-1350218390209
2012/10/14 13:39:50	19371-1350218390319
2012/10/14 13:39:50	19373-1350218390427
2012/10/14 13:39:50	19374-1350218390536

You should see that someplace the first digit will jump by 2. In this case 19372 is missing. This is simulating that case that someone else has logged in and a cookie is issued. Looking at the visualisation tab will illustrate this “jump” (as shown below).



We are guessing that the first digit of the cookie increments by 1 each time a user logs in and the second digit has something to do with time.

In the above example, the cookie will be “19372-” followed by 1350218390319 and 1350218390427.

We can use **JHijack** to try each of the possibilities.

- Extract the files from **jhijack** to your desktop
- Run the **.jar** file.
- Your task is gain a valid cookie so experiment!

**Note:** - “**Congratulations**” is the word to look for when we are successful

## APPENDIX C - XVWA CHALLENGES

---

The XVWA docker container also contains some relevant challenges.

XVWALoginAbout

Setup

Home

Instructions

Setup / Reset

Cryptography

A developer should understand which cryptography should be suitable for each required modules in application, it can be encoding, encrypting or hashing. Insecure implementation of cryptography can leads to sensitive data leakage.  
Read more about Cryptography  
[https://www.owasp.org/index.php/Guide\\_to\\_Cryptography](https://www.owasp.org/index.php/Guide_to_Cryptography)

Attacks

SQL Injection

SQL Injection (Blind)

OS Command Injection

XPATH Injection

Unrestricted File Upload

Enter your text here.

Enter Your Text

Submit Button

XVWAUserAbout

Setup

Home

Instructions

Setup / Reset

Session Flaws

Web applications require better session management to keep tracking the state of application and it's users' activities. Insecure session management can leads to attacks such as session prediction, hijacking, fixation and replay attacks.  
Read more about session management  
[https://www.owasp.org/index.php/Session\\_Management\\_Cheat\\_Shee](https://www.owasp.org/index.php/Session_Management_Cheat_Shee)

Attacks

SQL Injection

SQL Injection (Blind)

OS Command Injection

Welcome User

[Logout](#)