Personal Challenge Project - DNS tunneling

last modified by Daniel Baird

on 2020/11/24 02:42

Table of Contents

1. DNS basics 3
2. History 3
3. What's it for 4
4. Stories 4
5. How 5
6. Tools 5
7. My experimentation 6
8. How to detect it 11
  8.1 Snort and Suricata 13
  8.2 DNS (BIND) logs in Splunk 14
9. How to stop it 15
10. Ok, so how to REALLY stop it 19
11. The bigger strategic picture 20
12. References 20
  13. Articles and discussions on DNS tunneling 21
Marking 21
  Kristian Mansfield 21
  Richard Buckland 22

1. 1. DNS basics
2. 2. History
3. 3. What's it for
4. 4. Stories
5. 5. How
6. 6. Tools
7. 7. My experimentation
8. 8. How to detect it
    1. 8.1 Snort and Suricata
    2. 8.2 DNS (BIND) logs in Splunk
9. 9. How to stop it
10. 10. Ok, so how to REALLY stop it
11. 11. The bigger strategic picture
12. 12. References
    1. 13. Articles and discussions on DNS tunneling
13. Marking
    1. Kristian Mansfield
    2. Richard Buckland

link to blog post to help with marking. https://www.openlearning.com/unswcourses/courses/cyberfoundations/group-pages/5f8c950056f62dc6a5c34ebe/project-proposals/

This project is a personal education one, about DNS tunneling; connecting out of a network when direct connections are blocked using TCP over DNS.

1. ✓ learn how it works. enough to get up and running
2. ✓ get iodine working
3. ✓ look at logs (BIND into Splunk etc)
4. ✓ play with snort
5. ✓ play with packet captures in wireshark
6. ✓ read blog posts, articles, whitepapers. learn what others that went before figured out. stop before you fall victim to analysis-paralysis
7. ✓ try again to detect with splunk/snort.
8. ✓ ponder how to detect and/or block it, architectural defenses etc.
9. implement a method of blocking the traffic
10. This blog post explaining it all

A large amount of research work has already been done in this space already. To push the envelope, not just repeat what's already been done, a lot of reading is required. As this is an introductory subject, and there's no expectation that this project will push any envelopes of understanding, I focused on self-education. I read some material, but then got stuck in with some hands on lab testing. I alternated between reading and trying things out for myself.

1. DNS basics

DNS is one of the most foundational and fundamental protocols used on the Internet and IP networks in general. The phone book analogy gets used a lot. If you want to talk to something on the internet, you generally know it by a name, e.g. www.abc.net.au Then the owner of this domain name can configure resources that use it without users needing to know the details. Users just need to remember the www.abc.net.au name, and the DNS system takes care of the lookup so your computer/device knows how to connect to the site and show you the contents.

- https://en.wikipedia.org/wiki/Domain_Name_System
- https://blog.apnic.net/2019/11/04/dns-wars/ - Paul Vixie's keynote presentation at NANOG 77.

2. History

- The first discussion I've found of this attack was from Oskar Pearson on the Bugtraq mailing list in April of 1998.
- Tunneling over DNS mentioned during training by Rex di Bona while at Optus ~1999

- Slashdot article Sept 2000 "IP tunneling through nameservers"
- NSTX (the Nameserver Transfer Protocol) ~2002
- OzymanDNS ~2004
- many tools started coming out after this (see below)
- iodine in active development 2006 - 2014 (last release 2014-06-16: 0.7.0 "Kryoptonite") and some updates to "master" since then

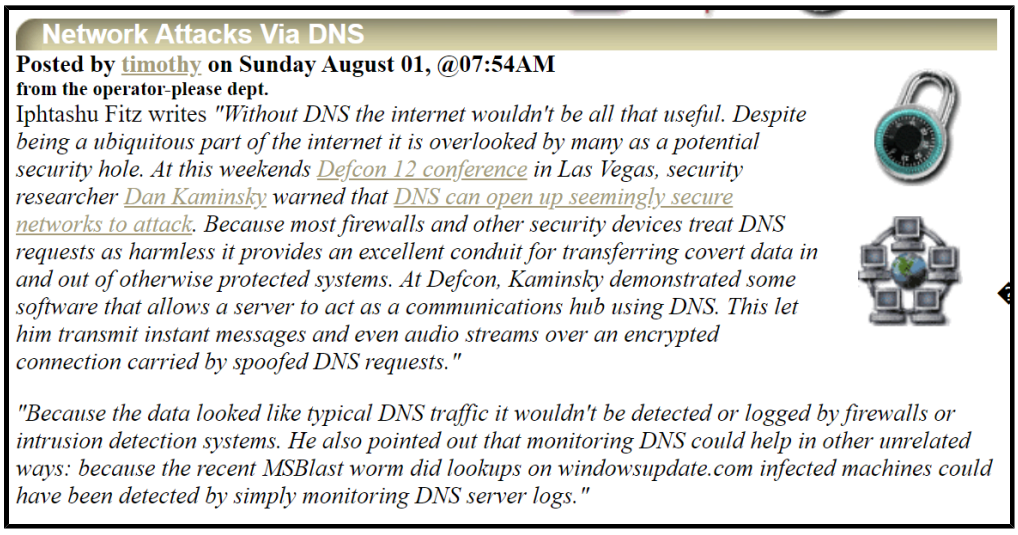
3. What's it for

DNS tunneling is a mechanism that enables bad things to happen. What kind of bad things?

- **Data Exfiltration** – Hackers sneak sensitive data out over DNS. It's certainly not the most efficient way to get data from a victim's computer— with all the extra overhead and encoding—but it can work and it's stealthy!
- **Command and Control (C2)** – Hackers use the DNS protocol to send simple commands to, say, a remote access trojan (RAT).
- **IP-Over-DNS Tunneling** — This sounds crazy, but there are utilities that have implemented the IP stack on the DNS query-response protocol. That would make it relatively easy to transfer data using standard communications software like FTP, Netcat, ssh, etc. Powerfully evil!

4. Stories

To the contrary of popular belief, DNS tunneling is not new. It was first suggested in this Bugtraq posting in April of 1998. Its real break to fame though was in Dan Kaminsky's 2004 Defcon presentation(see below). In between, there was also a tool called NTSX, which allowed you to do DNS tunneling without any difficulty. However, Dan added a new thing to the mix: entertainment. His tool allowed one to segment audio in small chunks and distribute it, and have it cached, in DNS servers. After this presentation, everything changed, and suddenly DNS tunneling was a major concern. Not so much for the radio though, but the tunneling itself - which wasn't new.



Network Attacks Via DNS

Posted by timothy on Sunday August 01, @07:54AM
from the operator-please dept.

Iphtashu Fitz writes *"Without DNS the internet wouldn't be all that useful. Despite being a ubiquitous part of the internet it is overlooked by many as a potential security hole. At this weekends Defcon 12 conference in Las Vegas, security researcher Dan Kaminsky warned that DNS can open up seemingly secure networks to attack. Because most firewalls and other security devices treat DNS requests as harmless it provides an excellent conduit for transferring covert data in and out of otherwise protected systems. At Defcon, Kaminsky demonstrated some software that allows a server to act as a communications hub using DNS. This let him transmit instant messages and even audio streams over an encrypted connection carried by spoofed DNS requests."*

"Because the data looked like typical DNS traffic it wouldn't be detected or logged by firewalls or intrusion detection systems. He also pointed out that monitoring DNS could help in other unrelated ways: because the recent MSBlast worm did lookups on windowsupdate.com infected machines could have been detected by simply monitoring DNS server logs."

5. How



DNS Tunneling works by abusing DNS records to traffic data in and out of a network. In principle, every type of record can be used, but the speed of the connection differs by the amount of data that can be stored in a single record. Below are some obvious ones:

- **TXT records** allow free-form data and can even include spaces. You can as such store information in it encoded with base64, allowing 220 bytes of data per record. TXT records are intended for "generic" use within the DNS framework. Users can place whichever data in it, as long as it meets the protocol requirements: a maximum length of 255 octets according to RFC 1035. In the recent past, TXT records had virtually been depreciated, which would have enabled highly security conscious environments to discard them at the enterprise DNS server. Today however, they have regained use as the carrier of the antispam-related SPF record.
- **CNAME records** allow only the characters A through Z, digits 0-9 and the hyphen. As such you require base32 encoding and are limited to about 110 bytes per record
- **EDNS0 messages** can be larger than the 512 byte maximum for UDP DNS, and can carry a 1280 byte payload by default. OzyManDNS uses a 768 byte payload for stability
- **A and MX records** can be used as well, but with more limitations. These cannot store all types of data.

6. Tools

Tool Name	Description
NSTX	tunneling network-packets over DNS
dns2tcp	dns2tcp was written by Olivier Dembour and Nicolas Collignon. It is written in C and runs on Linux. The client can run on Windows. It supports KEY and TXT request types.
tcp-over-dns	tcp-over-dns (TCP-over-DNS) was released in 2008. It has a Java based server and a Java based client. It runs on Windows, Linux and Solaris. It supports LZMA compression and both TCP and UDP traffic tunneling.
OzymanDNS	OzymanDNS is written in Perl by Dan Kaminsky in 2004. It is used to setup an SSH tunnel over DNS or for file transfer. Requests are base32 encoded and responses are base64 encoded TXT records.

iodine	iodine is a DNS tunneling program first released in 2006 with updates as recently as 2010. It was developed by Bjorn Andersson and Erik Ekman. Iodine is written in C and it runs on Linux, Mac OS X, Windows and others. Iodine has been ported to Android. It uses a TUN or TAP interface on the endpoint.
SplitBrain	SplitBrain is a variant of OzymanDNS.
DNScat-P / dnscat2	DNScat (DNScat-P) was originally released in 2004 and the most recent version was released in 2005. It was written by Tadeusz Pietraszek. DNScat is presented as a “Swiss-Army knife” tool with many uses involving bi-directional communication through DNS. DNScat is Java based and runs on Unix like systems. DNScat supports A record and CNAME record requests (Pietraszek, 2004). Since there are two utilities named DNScat, this one will be referred to as DNScat-P in this paper to distinguish it from the other one.
DNScapy	DNScapy was developed by Pierre Bienaime. It uses Scapy for packet generation. DNScapy supports SSH tunneling over DNS including a Socks proxy. It can be configured to use CNAME or TXT records or both randomly.
TUNS	TUNS, an IP over DNS tunnel, was developed by Lucas Nussbaum and written in Ruby. It does not use any experimental or seldom used record types. It uses only CNAME records. It adjusts the MTU used to 140 characters to match the data in a DNS request. TUNS may be harder to detect, but it comes at a performance cost.
PSUDP	PSUDP was developed by Kenton Born. It injects data into existing DNS requests by modifying the IP/UDP lengths. It requires all hosts participating in the covert network to send their DNS requests to a Broker service which can hold messages for a specific host until a DNS request comes from that host. The message can then be sent in the response.
Your Freedom	HTTPS/UDP/FTP/DNS/ECHO VPN & tunneling solution for Windows, Mac OSX, Linux and Android. Bypass proxies and access the Internet anonymously
Hexify	A tool is developed by Infoblox to do the penetrating test for DNS tunneling.
others?	*... there's bound to be more out there*

7. My experimentation

https://code.kryo.se/iodine/ # In the interest of time, I just picked an implementation that I already knew about for the lab setup. It's well known and easy to use from what I hear and read. I've never actually tried to use it before though.

Two machines, a mini-PC as client, running Ubuntu 20



and a server instance (Ubuntu 20) in a container on one of my Intel NUCs



a quickstart guide found on [github]

DNS tunneling with iodine.md

Raw

##Domain

We need some records on our domain (mydomain.com) DNS for connections. Add these records:

```
t1              IN      NS      t1ns.mydomain.com. ; note final the dot!
t1ns            IN      A       OUR_SERVER_IP
```

##Server

Install the `iodine` package, run the server with a password and an unused range ip address.

```
sudo aptitude install iodine
sudo iodined -f -c -P apassword 10.0.1.1 t1.mydomain.com &
```

##Linux client

Install iodine and establish a connection with the server via DNS tunnelling. Then forward the 9999 port via ssh your server to act as a proxy for connections (connect with your server user and password).

```
sudo iodine -f -P apassword t1.mydomain.com &
ssh user@10.0.1.1 -D 9999
```

##Browser

Configure your proxy settings for socks 5 proxy localhost port 9999

set up a new subdomain "tun.home" on my internal DNS servers pointing to the new container with hostname "iodine-svr"

```
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-52-generic x86_64)

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

  System information as of Sun 01 Nov 2020 03:20:54 AM UTC

  System load:  0.0                Temperature:          49.0 C
  Usage of /:   14.0% of 116.12GB  Processes:            282
  Memory usage: 18%                Users logged in:      0
  Swap usage:   0%                 IPv4 address for br0: 192.168.101.3

Last login: Fri Oct 30 10:49:10 2020 from 192.168.101.148
daniel@nuc03:~$ lxc list
+------------+---------+-----------------------+------+-----------+-----------+
|    NAME    |  STATE  |          IPV4         | IPV6 |   TYPE    | SNAPSHOTS |
+------------+---------+-----------------------+------+-----------+-----------+
| gravwell   | STOPPED |                       |      | CONTAINER | 0         |
+------------+---------+-----------------------+------+-----------+-----------+
| iodine-svr | RUNNING | 192.168.101.195 (eth0) |      | CONTAINER | 0         |
+------------+---------+-----------------------+------+-----------+-----------+
| test01     | RUNNING | 192.168.101.153 (eth0) |      | CONTAINER | 0         |
+------------+---------+-----------------------+------+-----------+-----------+
| test02     | RUNNING | 192.168.101.192 (eth0) |      | CONTAINER | 0         |
+------------+---------+-----------------------+------+-----------+-----------+
| xwiki      | RUNNING | 192.168.101.228 (eth0) |      | CONTAINER | 0         |
+------------+---------+-----------------------+------+-----------+-----------+
daniel@nuc03:~$
```

iodine installed on both from Ubuntu's package system "apt install iodine"

Looking at the help for iodine and iodined, there seems to be a lot of configuration options. This gives a lot of flexibility to avoid preventative and detective controls.

```
root@iodine-svr:~# iodine -v
iodine IP over DNS tunneling client
version: 0.7.0 from 2014-06-16
root@iodine-svr:~# iodine -h
iodine IP over DNS tunneling client
Usage: iodine [-v] [-h] [-f] [-r] [-u user] [-t chrootdir] [-d device] [-P password] [-m maxfragsize] [-M maxlen] [-T type]
[-O enc] [-L 0|1] [-I sec] [-z context] [-F pidfile] [nameserver] topdomain
Options to try if connection doesn't work:
  -T force dns type: NULL, PRIVATE, TXT, SRV, MX, CNAME, A (default: autodetect)
  -O force downstream encoding for -T other than NULL: Base32, Base64, Base64u,
     Base128, or (only for TXT:) Raw  (default: autodetect)
  -I max interval between requests (default 4 sec) to prevent DNS timeouts
  -L 1: use lazy mode for low-latency (default). 0: don't (implies -I1)
  -m max size of downstream fragments (default: autodetect)
  -M max size of upstream hostnames (~100-255, default: 255)
  -r to skip raw UDP mode attempt
  -P password used for authentication (max 32 chars will be used)
Other options:
  -v to print version info and exit
  -h to print this help and exit
  -f to keep running in foreground
  -u name to drop privileges and run as user 'name'
  -t dir to chroot to directory dir
  -d device to set tunnel device name
  -z context, to apply specified SELinux context after initialization
  -F pidfile to write pid to a file
nameserver is the IP number/hostname of the relaying nameserver. if absent, /etc/resolv.conf is used
topdomain is the FQDN that is delegated to the tunnel endpoint.
root@iodine-svr:~# iodined -h
iodine IP over DNS tunneling server
Usage: iodined [-v] [-h] [-c] [-s] [-f] [-D] [-u user] [-t chrootdir] [-d device] [-m mtu] [-z context] [-l ip address to listen
on] [-p port] [-n external ip] [-b dnsport] [-P password] [-F pidfile] tunnel_ip[/netmask] topdomain
  -v to print version info and exit
  -h to print this help and exit
  -c to disable check of client IP/port on each request
  -s to skip creating and configuring the tun device, which then has to be created manually
  -f to keep running in foreground
  -D to increase debug level
     (using -DD in UTF-8 terminal: "LC_ALL=C luit iodined -DD ...")
  -u name to drop privileges and run as user 'name'
  -t dir to chroot to directory dir
  -d device to set tunnel device name
  -m mtu to set tunnel device mtu
  -z context to apply SELinux context after initialization
  -l ip address to listen on for incoming dns traffic (default 0.0.0.0)
  -p port to listen on for incoming dns traffic (default 53)
  -n ip to respond with to NS queries
  -b port to forward normal DNS queries to (on localhost)
  -P password used for authentication (max 32 chars will be used)
  -F pidfile to write pid to a file
  -i maximum idle time before shutting down
tunnel_ip is the IP number of the local tunnel interface.
   /netmask sets the size of the tunnel network.
topdomain is the FQDN that is delegated to this server.
root@iodine-svr:~#
```

NOTE: I just noticed the **-b** option, which is very clever. If there is a DNS server running on the host already, you can move it to run on a different port (e.g. 5353) and have iodine forward all normal queries it receives to that port.

```
root@iodine-svr:~# sudo iodined -f -c -P apassword 10.0.1.1 tun.home &
[1] 2137
root@iodine-svr:~# Opened dns0
Setting IP of dns0 to 10.0.1.1
Setting MTU of dns0 to 1130
Opened IPv4 UDP socket
Listening to dns for domain tun.home
```

```
daniel@LarkBox:~$ sudo iodine -r -f -P apassword tun.home &
[1] 6817
daniel@LarkBox:~$ Opened dns0
Opened IPv4 UDP socket
Sending DNS queries for tun.home to 127.0.0.53
Autodetecting DNS query type (use -T to override).iodine: Got NOTIMP as reply: server does not support our request
...iodine: Got NOTIMP as reply: server does not support our request
..iodine: Got NOTIMP as reply: server does not support our request
.
Using DNS type TXT queries
Version ok, both using protocol v 0x00000502. You are user #0
Setting IP of dns0 to 10.0.1.2
Setting MTU of dns0 to 1130
Server tunnel IP is 10.0.1.1
Skipping raw mode
Using EDNS0 extension
Switching upstream to codec Base128
Server switched upstream to codec Base128
Autodetecting downstream codec (use -O to override)
Switching downstream to codec Raw
Server switched downstream to codec Raw
Switching to lazy mode for low-latency
Server switched to lazy mode
Autoprobing max downstream fragment size... (skip with -m fragsize)
768 ok.. 1152 ok.. ...1344 not ok.. ...1248 not ok.. ...1200 not ok.. 1176 ok.. ...1188 not ok.. will use 1176-2=1174
Setting downstream fragment size to max 1174...
Connection setup complete, transmitting data.
```

OK, so it seems to be working, but what has it set up?

```
daniel@LarkBox:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: wlo2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 34:cf:f6:46:65:d1 brd ff:ff:ff:ff:ff:ff
    inet 192.168.101.194/24 brd 192.168.101.255 scope global dynamic noprefixroute wlo2
       valid_lft 934sec preferred_lft 934sec
    inet6 fe80::4a52:235c:2b55:b0f3/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: dns0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1130 qdisc fq_codel state UNKNOWN group default qlen 500
    link/none
    inet 10.0.1.2/27 scope global dns0
       valid_lft forever preferred_lft forever
```

```
root@iodine-svr:~# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
5: dns0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1130 qdisc fq_codel state UNKNOWN group default qlen 500
    link/none
    inet 10.0.1.1/27 scope global dns0
       valid_lft forever preferred_lft forever
27: eth0@if28: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 00:16:3e:c6:89:43 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.101.195/24 brd 192.168.101.255 scope global dynamic eth0
       valid_lft 1146sec preferred_lft 1146sec
    inet6 fe80::216:3eff:fec6:8943/64 scope link
       valid_lft forever preferred_lft forever
```

It was a surprise how it just worked. I expected it to be harder to set up. Next a bit of testing

```
daniel@LarkBox:~$ ping 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=6.81 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=8.59 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=6.09 ms
^C
--- 10.0.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 6.089/7.162/8.591/1.051 ms
daniel@LarkBox:~$ ssh daniel@10.0.1.1
daniel@10.0.1.1's password:

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Could not chdir to home directory /home/daniel: No such file or directory
$ bash
daniel@iodine-svr:/$ uname -a
Linux iodine-svr 5.4.0-52-generic #57-Ubuntu SMP Thu Oct 15 10:57:00 UTC 2020 x86
daniel@iodine-svr:/$
```

and check that this traffic is actually going out over DNS

- check what the iodine processes are doing (lsof)
- check what packets are going back and forth (tcpdump)

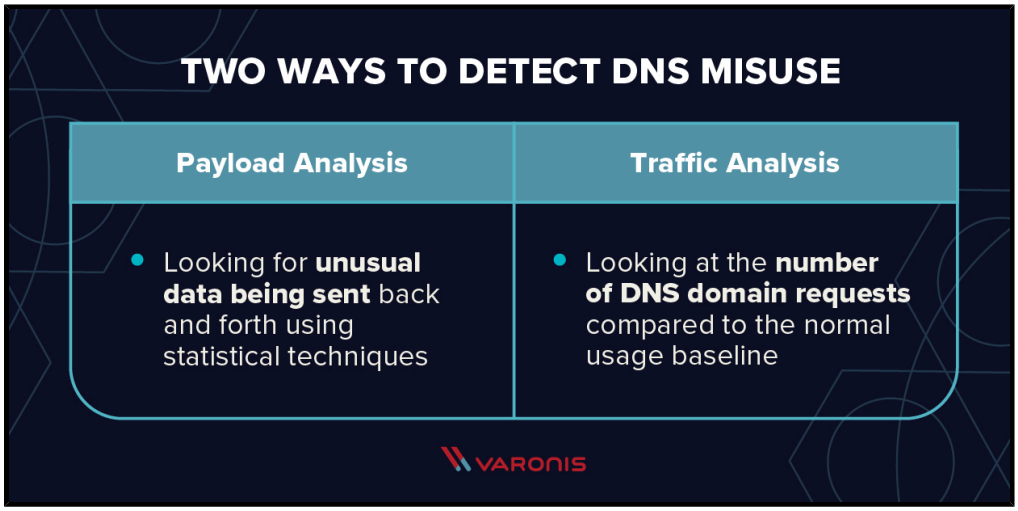
tcpdump showed lots of port 53 DNS TXT queries and responses between each host and my name servers, but nothing directly between them.

lsof showed the iodine processes using /dev/net/tun for the tunnel interface, a bunch of system libraries and UDP sockets; port 53 on the server node and a high port on the client node.

So it works. Now to see how it looks in Splunk, where my DNS servers (BIND) are logging all queries into

8. How to detect it

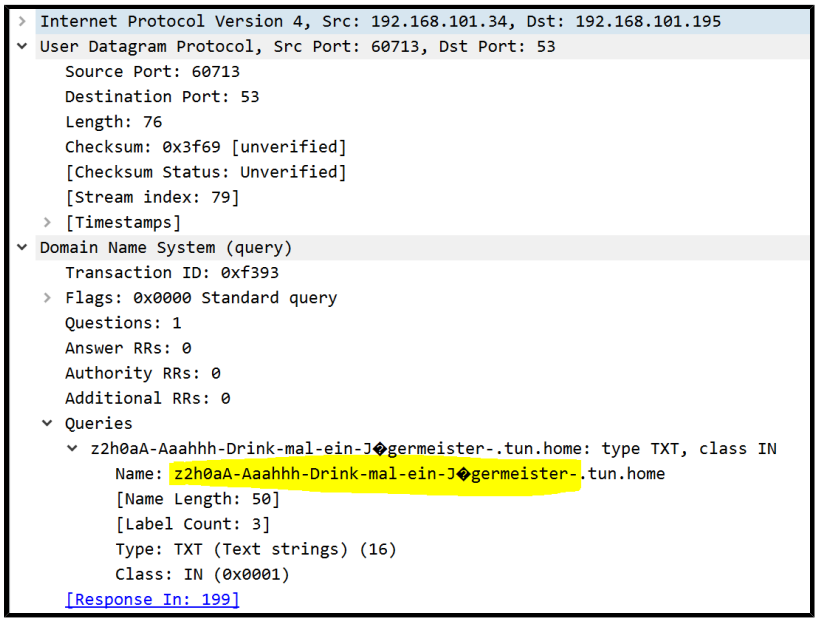
Short answer: difficult. If an attacker is using this, they're likely to be fairly sophisticated, and if so, can tweak and randomise the traffic to blend in. Detections I found were rather brittle.



1. put snort in between to detect the DNS tunnel traffic as it passes over ethernet.
2. capture port 53 traffic on the endpoints and/or the in-line box running snort.
3. experiment with the DNS query logs that go into Splunk from my two DNS resolvers 192.168.101.34 and 192.168.101.36

- High volume of DNS requests from internal clients where little usually take place;
- Significant difference in the format of these lookups as compared to regular ones. Each of the requests is usually long and contains higher entropy than usual. Hostnames are usually formatted using a language. All languages contain redundancy. Base64 or Base3 "looks" different
- The total amount of data transferred over port 53 is much higher than usual. This is much easier to detect on the internal network than it is on the perimeter (from main DNS relays to outside)

Looking at the pcap files in Wireshark, some almost readable stuff stood out.



then greping through the source it seems there are strings sent to see if they come through intact, thus determining what's supported by the DNS resolvers that the traffic is passing through.

```
static int
handshake_upenc_autodetect(int dns_fd)
/* Returns:
   0: keep Base32
   1: Base64 is okay
   2: Base64u is okay
   3: Base128 is okay
*/
{
    /* Note: max 59 chars, must start with "aA".
       pat64: If 0129 work, assume 3-8 are okay too.

       RFC1035 par 2.3.1 states that [A-Z0-9-] allowed, but only
       [A-Z] as first, and [A-Z0-9] as last char _per label_.
       Test by having '-' as last char.
     */
    const char *pat64 = "aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ+0129-";
    const char *pat64u = "aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ_0129-";
    const char *pat128a = "aA-Aaahhh-Drink-mal-ein-J\344germeister-";
    const char *pat128b = "aA-La-fl\373te-na\357ve-fran\347aise-est-retir\351-\340-Cr\350te";
    const char *pat128c = "aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ";
    const char *pat128d = "aA0123456789\274\275\276\277"
        "\300\301\302\303\304\305\306\307\310\311\312\313\314\315\316\317";
    const char *pat128e="aA"
            "\320\321\322\323\324\325\326\327\330\331\332\333\334\335\336\337"
            "\340\341\342\343\344\345\346\347\350\351\352\353\354\355\356\357"
            "\360\361\362\363\364\365\366\367\370\371\372\373\374\375";
    int res;
```

If we try to detect these fixed handshake strings it should work for anyone using this particular version of iodine, and isn't sophisticated enough to think to change the src before using it. So, rather unreliable.

8.1 Snort and Suricata

- in-line IDS/IPS. detect and block in real-time. downside is the practicality of integrating into the network and the computer power required.
- tried OPNsense, pfSense, ntop-ng, vanilla Ubuntu with multiple eth ports in a bridge
- OPNsense and pfSense make it easier to work with Snort/Suricata configs, but I had problems getting traffic flowing through the bridge. Gave up troubleshooting in the interests of time
- Tried vanilla Ubuntu, using https://www.snort.org/documents/snort-3-on-ubuntu-18-19 , which worked in that I got it all up and running, but I hit the wall when I tried to create custom Snort v3 rules to detect the DNS tunnelling traffic. Turning on all the built in rules for DNS didn't trigger when the Iodine traffic passed through. confirmed with packet captures that the traffic WAS passing through and that Snort could detect large DNS packets. Much more intelligent rules are needed to avoid alarming on normal traffic but still catch the tunneling traffic. This is where I got stuck.
- ntop-ng was easier to work with, but compute intensive. It overloaded the Intel i5 CPU in my test PC, even with under 100mbps of traffic passing through.



8.2 DNS (BIND) logs in Splunk

Experiment with what to search for to detect the tunneling traffic. BIND only logs queries, not replies. To check reply content, in-inline IPS (snort etc) tech is needed.

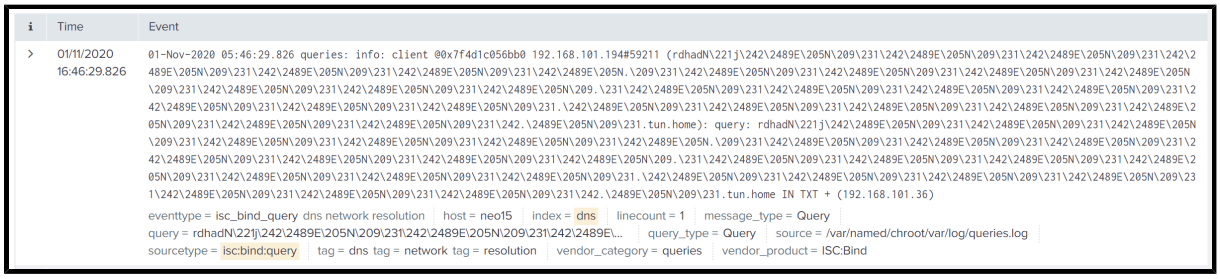
- size of requests(bigger than legit ones) ?
- frequency of requests (bursts of requests) ?
- unusual request types (TXT, TYPE65399, SRV) ?
- request content that's different from legit traffic (resource record being requested, special characters) ?

While looking through the query logs in Splunk , I noticed these TYPE65399 queries, which isn't a legitimate DNS resource record type you can query.



Time	Event
26/10/2020 10:41:50.793	25-Oct-2020 23:41:50.793 queries: info: client @0x7f38867c23a0 192.168.101.194#59784 (yrb2k5.tun.home): query: yrb2k5.tun.home IN TYPE65399 +E(0) (192.168.101.34) eventtype = isc_bind_query dns network resolution host = neo13 index = dns linecount = 1 message_type = Query query = yrb2k5.tun.home query_type = Query source = /var/named/chroot/var/log/queries.log sourcetype = isc:bind:query tag = dns tag = network tag = resolution vendor_category = queries vendor_product = ISC:Bind
26/10/2020 10:41:50.787	25-Oct-2020 23:41:50.787 queries: info: client @0x7f38940f72c0 192.168.101.194#43413 (yrb2k3.tun.home): query: yrb2k3.tun.home IN TYPE65399 +E(0) (192.168.101.34) eventtype = isc_bind_query dns network resolution host = neo13 index = dns linecount = 1 message_type = Query query = yrb2k3.tun.home query_type = Query source = /var/named/chroot/var/log/queries.log sourcetype = isc:bind:query tag = dns tag = network tag = resolution vendor_category = queries vendor_product = ISC:Bind
26/10/2020 10:41:50.779	25-Oct-2020 23:41:50.779 queries: info: client @0x7f387d3a2ec0 192.168.101.194#58778 (yrb2k0.tun.home): query: yrb2k0.tun.home IN TYPE65399 +E(0) (192.168.101.34) eventtype = isc_bind_query dns network resolution host = neo13 index = dns linecount = 1 message_type = Query query = yrb2k0.tun.home query_type = Query source = /var/named/chroot/var/log/queries.log sourcetype = isc:bind:query tag = dns tag = network tag = resolution vendor_category = queries vendor_product = ISC:Bind
26/10/2020 10:39:02.751	25-Oct-2020 23:39:02.751 queries: info: client @0x7f387d3a2ec0 192.168.101.194#35864 (yrbaik.tun.home): query: yrbaik.tun.home IN TYPE65399 +E(0) (192.168.101.34) eventtype = isc_bind_query dns network resolution host = neo13 index = dns linecount = 1 message_type = Query query = yrbaik.tun.home query_type = Query source = /var/named/chroot/var/log/queries.log sourcetype = isc:bind:query tag = dns tag = network tag = resolution vendor_category = queries vendor_product = ISC:Bind
26/10/2020 10:39:02.747	25-Oct-2020 23:39:02.747 queries: info: client @0x7f3889092a90 192.168.101.194#49494 (yrbaii.tun.home): query: yrbaii.tun.home IN TYPE65399 +E(0) (192.168.101.34) eventtype = isc_bind_query dns network resolution host = neo13 index = dns linecount = 1 message_type = Query query = yrbaii.tun.home query_type = Query source = /var/named/chroot/var/log/queries.log sourcetype = isc:bind:query tag = dns tag = network tag = resolution vendor_category = queries vendor_product = ISC:Bind
26/10/2020 10:39:02.737	25-Oct-2020 23:39:02.737 queries: info: client @0x7f388ceec0c0 192.168.101.194#44353 (yrbaif.tun.home): query: yrbaif.tun.home IN TYPE65399 +E(0) (192.168.101.34) eventtype = isc_bind_query dns network resolution host = neo13 index = dns linecount = 1 message_type = Query query = yrbaif.tun.home query_type = Query source = /var/named/chroot/var/log/queries.log sourcetype = isc:bind:query tag = dns tag = network tag = resolution vendor_category = queries vendor_product = ISC:Bind

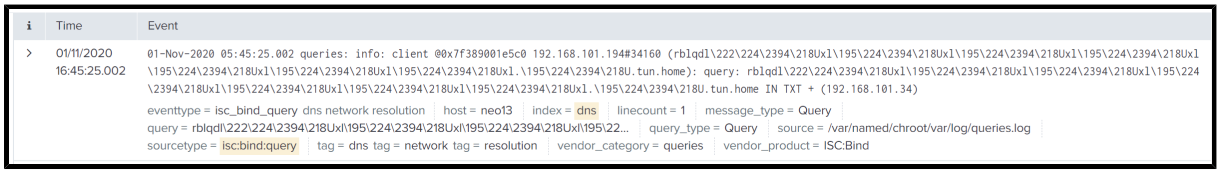
Trying various searches in Splunk, default iodine use resulted in large queries, which stood out from normal ones. Searching for any query larger than 120 characters had no false positives when searching through my DNS logs for a week(over 700,000 queries). Splunk SPL: index=dns | where len(query) >=120



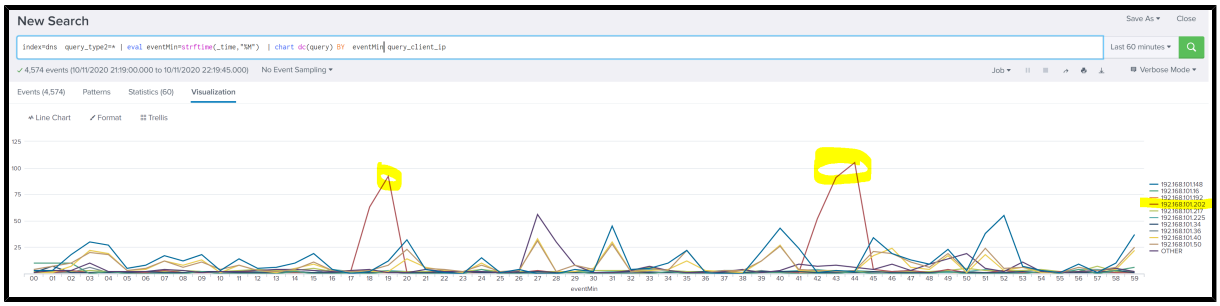
However I expect this isn't too hard to change and thus hide within the range of legit traffic.

e.g. this query was tracked down to a company using O365: selector2-TallentireOFlynnMovsowitz-onmicrosoft-com._domainkey.TallentireOFlynnMovsowitz.onmicrosoft.com

So I reduced upstream and downstream fragments/hostnames and the resulting BIND queries were a lot smaller (under 200 down from over 600) but iodine broke when I tried to set the values low enough to get the hostnames down under 120 characters.



I also tried looking at query frequency over small time periods for single client IP addresses.



I thought some basic anomaly detection might work

9. How to stop it

I gave up on creating rules in Snort/Suricata in the interest of time and lab practicalities. I switched to focusing on a logging and DNS resolver solution. I already block direct DNS traffic out to the internet from my lab network, only the 2 DNS resolvers are allowed out. This forces all DNS resolution to go via these two servers. If I can detect the tunneling, I should be able to dynamically update the DNS resolvers to block it.

I recently set up some free DNS RPZ feeds for my resolvers, which have already been blocking suspect sites/domains that I've accidently encountered (following links to dark corners of the internet, malicious embedded ads etc)



Time	Event
14/11/2020 13:17:59.331	14-Nov-2020 02:17:59.331 rpz: info: client @0x7fee240b7e40 192.168.101.40#37952 (www.powr.io): rpz QNAME NXDOMAIN rewrite www.powr.io via www.powr.io.basic-dnsfw.rpz.threatstop.local host = neo13 index = dns linecount = 1 source = /var/named/chroot/var/log/rpz.log sourcetype = isc:bind:rpz
13/11/2020 23:25:22.722	13-Nov-2020 12:25:22.722 rpz: info: client @0x7f03b00016a0 127.0.0.1#42389 (www.pubyun.com): rpz QNAME NXDOMAIN rewrite www.pubyun.com via www.pubyun.com.coinblocker.srv host = dns-master index = dns linecount = 1 source = /var/log/named/rpz.log sourcetype = isc:bind:rpz
10/11/2020 09:57:20.274	09-Nov-2020 22:57:20.274 rpz: info: client @0x7f7ebc1d7080 192.168.101.148#61390 (www.woodmann.com): rpz QNAME NXDOMAIN rewrite www.woodmann.com via www.woodmann.com.basic-dnsfw.rpz.threatstop.local host = neo13 index = dns linecount = 1 source = /var/named/chroot/var/log/rpz.log sourcetype = isc:bind:rpz
05/11/2020 16:54:52.549	05-Nov-2020 05:54:52.549 rpz: info: client @0x7f387d1ca010 192.168.101.50#39782 (rb.gy): rpz QNAME NXDOMAIN rewrite rb.gy via rb.gy.basic-dnsfw.rpz.threatstop.local host = neo13 index = dns linecount = 1 source = /var/named/chroot/var/log/rpz.log sourcetype = isc:bind:rpz
03/11/2020 14:42:00.648	03-Nov-2020 03:42:00.648 rpz: info: client @0x7f4d1c1143d0 192.168.101.148#64948 (pdfzon.com): rpz IP NXDOMAIN rewrite pdfzon.com via 32.197.117.54.198.rpz-ip.basic-dnsfw.rpz.threatstop.local host = neo15 index = dns linecount = 1 source = /var/named/chroot/var/log/rpz.log sourcetype = isc:bind:rpz
15/10/2020 15:38:42.767	15-Oct-2020 04:38:42.767 rpz: info: client @0x7fc30c0f72c0 192.168.101.148#59851 (network.bazaarvoice.com): rpz QNAME NXDOMAIN rewrite network.bazaarvoice.com via network.bazaarvoice.com.basic-dnsfw.rpz.threatsto host = neo13 index = dns linecount = 1 source = /var/named/chroot/var/log/rpz.log sourcetype = isc:bind:rpz
04/10/2020 19:01:08.305	04-Oct-2020 08:01:08.305 rpz: info: client @0x7f2e4415c7e0 192.168.101.148#59878 (www.bannister.fr): rpz IP NXDOMAIN rewrite www.bannister.fr via 32.19.33.186.213.rpz-ip.basic-dnsfw.rpz.threatstop.local host = neo15 index = dns linecount = 1 source = /var/named/chroot/var/log/rpz.log sourcetype = isc:bind:rpz
02/10/2020 08:23:18.510	01-Oct-2020 21:23:18.510 rpz: info: client @0x7f94540c0480 192.168.101.212#30684 (zpr.io): rpz QNAME NXDOMAIN rewrite zpr.io via zpr.io.basic-dnsfw.rpz.threatstop.local host = neo13 index = dns linecount = 1 source = /var/named/chroot/var/log/rpz.log sourcetype = isc:bind:rpz
15/09/2020 12:38:17.737	15-Sep-2020 01:38:17.737 rpz: info: client @0x7ff6c4028500 192.168.101.50#58709 (www.99counters.com): rpz IP NXDOMAIN rewrite www.99counters.com via 32.197.117.54.198.rpz-ip.basic-dnsfw.rpz.threatstop.local host = neo13 index = dns linecount = 1 source = /var/named/chroot/var/log/rpz.log sourcetype = isc:bind:rpz
08/09/2020 22:03:07.806	08-Sep-2020 11:03:07.806 rpz: info: client @0x7fec6f1d9000 192.168.101.212#37652 (www.sharebutton.co): rpz IP NXDOMAIN rewrite www.sharebutton.co via 32.153.242.59.199.rpz-ip.basic-dnsfw.rpz.threatstop.local host = neo13 index = dns linecount = 1 source = /var/named/chroot/var/log/rpz.log sourcetype = isc:bind:rpz
07/09/2020 15:24:51.153	07-Sep-2020 04:24:51.153 rpz: info: client @0x7f34768ec4d0 192.168.101.148#58899 (pubmlisher.com): rpz IP NXDOMAIN rewrite pubmlisher.com via 32.169.194.94.85.rpz-ip.basic-dnsfw.rpz.threatstop.local host = neo15 index = dns linecount = 1 source = /var/named/chroot/var/log/rpz.log sourcetype = isc:bind:rpz
06/09/2020 08:06:42.194	05-Sep-2020 21:06:42.194 rpz: info: client @0x7fec5ac17e40 192.168.101.34#44561 (mirror.nodesdirect.com): rpz IP NXDOMAIN rewrite mirror.nodesdirect.com via 20.0.80.92.23.rpz-ip.drop.rpz.spamhaus.org host = neo13 index = dns linecount = 1 source = /var/named/chroot/var/log/rpz.log sourcetype = isc:bind:rpz
29/08/2020 17:41:39.837	29-Aug-2020 06:41:39.837 rpz: info: client @0x7fc64d2c0340 192.168.101.40#48547 (www.9bis.net): rpz IP NXDOMAIN rewrite 9bis.net via 32.19.33.186.213.rpz-ip.basic-dnsfw.rpz.threatstop.local host = neo13 index = dns linecount = 1 source = /var/named/chroot/var/log/rpz.log sourcetype = isc:bind:rpz
18/08/2020 11:49:26.804	18-Aug-2020 00:49:26.804 rpz: info: client @0x7f103c04c7f0 192.168.101.148#51585 (snip.ly): rpz QNAME NXDOMAIN rewrite snip.ly via snip.ly.basic-dnsfw.rpz.threatstop.local host = neo15 index = dns linecount = 1 source = /var/named/chroot/var/log/rpz.log sourcetype = isc:bind:rpz
11/08/2020 00:38:14.916	10-Aug-2020 13:38:14.916 rpz: info: client @0x7f77244d68f0 192.168.101.40#49451 (1drv.ms): rpz QNAME NXDOMAIN rewrite 1drv.ms via 1drv.ms.basic-dnsfw.rpz.threatstop.local host = neo13 index = dns linecount = 1 source = /var/named/chroot/var/log/rpz.log sourcetype = isc:bind:rpz
09/08/2020 13:57:09.183	09-Aug-2020 02:57:09.183 rpz: info: client @0x7f77288a30f0 192.168.101.40#49451 (apps.bazaarvoice.com): rpz QNAME NXDOMAIN rewrite apps.bazaarvoice.com via apps.bazaarvoice.com.basic-dnsfw.rpz.threatstop.local host = neo13 index = dns linecount = 1 source = /var/named/chroot/var/log/rpz.log sourcetype = isc:bind:rpz

I've been collecting all my DNS logs for years, so I have a large data set to search across. These searches had no false positives;

- backslash in the query string. Iodine queries have lots of them. I didn't find any other instances in my logs. some quick RFC searches (rfc952, rfc1123) showed numbers, letters, underscore and dash as valid characters, also dots to divide domains and subdomains. Backslash (\) forwardslash (//) comma (,) tilde (~) colon (:) exclamation point (!!) at sign (@) number sign (#) dollar sign ($) percent (%) caret (^) ampersand (&) apostrophe (') parentheses (()) braces ({}) white space (blank) should all be red flags.

index=dns sourcetype=isc:bind:query | regex query="[\\\/\,\~\:\ \!\@\#\$\%\^\&\'\(\)\{\}]+" | rex field=query "^[^\.]+\.(?P<domain_name>.+)" | table _time query domain_name | dedup domain_name

- query string of larger than 150 characters. Some Microsoft RRs were getting up close to 120 characters. The Iodine queries that carried tunnel data were all over 200 characters. This is a less reliable indicator as each component of a FQDN is allowed to be up to 63 characters and the whole FQDN can be up to 253characters. query string length could be use in conjunction with other indicators to help eliminate false positives or false negatives however.
- High entropy in hostname. Use URL Toolbox add on for Splunk

index=dns sourcetype=isc:bind:query | eval utlist = "custom" | `ut_shannon(query)` | search ut_shannon > 2.5 | table _time query ut_shannon | sort - ut_shannon
index=dns sourcetype=isc:bind:query | eval utlist = "custom" | `ut_parse_simple(query)` | `ut_shannon(query)` | search ut_shannon > 4.8 | rex field=query "^[^\.]+\.(?P<domain_name>.+)" | dedup domain_name | table _time query domain_name ut_shannon | sort - ut_shannon

_time	query	ut_shannon
2020-11-10 21:42:42.891	z1hyaAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ.tun.home	5.672292620687481
2020-11-10 21:43:30.280	zjz4aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ.tun.home	5.67229262068748
2020-11-10 22:18:51.750	ztdfaAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ.tun.home	5.64152338991825
2020-11-10 22:18:27.061	zzihaAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ.tun.home	5.629909736038813
2020-11-14 19:36:03.858	jfpwxrcomxc8ktnh2az506bqbgkss1605342963.nuid.imrworldwide.com	4.9341145913420466
2020-11-14 19:36:03.858	jfpwxrcomxc8ktnh2az506bqbgkss1605342963.nuid.imrworldwide.com	4.9341145913420466
2020-11-14 19:36:03.645	jfpwxrcomxc8ktnh2az506bqbgkss1605342963.nuid.imrworldwide.com	4.9341145913420466
2020-11-09 11:15:33.798	72k6ns9aftm7fajz5q4ihglcrublznbdnurtbp7e15364026b4e1ab43sac.d.aa.online-metrix.net	4.845195072077294
2020-11-09 11:15:33.797	72k6ns9aftm7fajz5q4ihglcrublznbdnurtbp7e15364026b4e1ab43sac.d.aa.online-metrix.net	4.845195072077294
2020-11-09 11:15:33.674	72k6ns9aftm7fajz5q4ihglcrublznbdnurtbp7e15364026b4e1ab43sac.d.aa.online-metrix.net	4.845195072077294

_time	query	domain_name	ut_shannon
2020-11-10 22:18:51.750	ztdfaAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ.tun.home	tun.home	5.64152338991825
2020-11-14 19:36:03.858	jfpwxrcomxc8ktnh2az506bqbgkss1605342963.nuid.imrworldwide.com	nuid.imrworldwide.com	4.9341145913420466
2020-11-09 11:15:33.798	72k6ns9aftm7fajz5q4ihglcrublznbdnurtbp7e15364026b4e1ab43sac.d.aa.online-metrix.net	d.aa.online-metrix.net	4.845195072077292
2020-11-13 07:09:08.835	p4-doobex2yqsgbg-7h2kb5q3y52r5xjx-691073-i1-v6exp3.v4.metric.gstatic.com	v4.metric.gstatic.com	4.820501175107986
2020-11-13 07:09:38.993	p4-doobex2yqsgbg-7h2kb5q3y52r5xjx-691073-s1-v6exp3-v4.metric.gstatic.com	metric.gstatic.com	4.811403646885019

💬 These other hostnames aren't tunnels, but I'm thinking I might want to block them too. They looks like advertising trackers.

⚠ WARNING: the URL Toolbox doesn't pick up the queries with backslashes in them

- 🔔 anomaly detection.

-> splunk alert

-> nsupdate script

/home/daniel/nsupdate-dynip.sh

https://docs.splunk.com/Documentation/SplunkCloud/latest/Alert/Configuringscriptedalerts

When you run a script as an alert action, positional arguments that capture alert information are passed to the script. The positional arguments are also available as environment variables. You can access information from each argument using the notation in the following table.

Arg	Env Variable	Value	Example
0	**SPLUNK_ARG_0**	Script name	/opt/splunk/bin/s
1	**SPLUNK_ARG_1**	Number of events returned	1
2	**SPLUNK_ARG_2**	Search terms	index=dns sourc
3	**SPLUNK_ARG_3**	Fully qualified query string	index=dns sourc
4	**SPLUNK_ARG_4**	Name of report	DNS Tunneling
5	**SPLUNK_ARG_5**	Trigger reason For example, "The number of events was greater than 1."	Saved Search [D
6	**SPLUNK_ARG_6**	Browser URL to view the report.	https://splunk01
7	**SPLUNK_ARG_7**	Not used for historical reasons.	
8	**SPLUNK_ARG_8**	File in which the results for the search are stored. Contains raw results in gzip file format.	/opt/splunk/var/r

then unpack the gzip'd CSV

```
root@splunk01:/var/tmp# gzip -d < tmp_0.csv.gz
```

```
"_time",query,"domain_name","ut_shannon","__mv__time","__mv_query","__mv_domain_name","__mv_ut_shannon"
"1605653046.360","zvn2aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ.tun.home","tun.home","5.672292620687481",,
```

strip the header

```
root@splunk01:/var/tmp# gzip -d < tmp_0.csv.gz | tail -n +2
"1605653046.360","zvn2aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ.tun.home","tun.home","5.672292620687481",,
```

read the file and loop through the rows

extract the hostname

```
nsupdate -k /home/daniel/Ktest-splunk-key.+163+35858.key -v /tmp/nsupdate
```

I ran out of time to complete the script. this was the start of it. it would log the hostname field from the Splunk alert. every time I set up the iodine tunnel it would reliably log the base hostname used.

```
#!/usr/bin/env bash

# RPZ dynamic update
#
# [TODO]
# - take argments to set variable for RR and RPZ action.
# - input validation for RR etc
# - capture failures; read file, write file, remove file etc.
# -

# simple script that writes parameters 0-7 to
# $SPLUNK_HOME/bin/scripts/echo_output.txt
# $SPLUNK_ARG_0 and $0 show how to use the long and short form.

read sessionKey
echo "'$sessionKey'" >>  "/var/tmp/echo_output.txt"
echo "'$SPLUNK_ARG_0' '$0' '$1' '$2' '$3' '$4' '$5' '$6' '$7' '$8' '$sessionKey'" >>  "/var/tmp/echo_output.txt"
gzip -d < $8  >>  "/var/tmp/echo_output.txt"

gzip -d < $8 | tail -n +2 | while read i
do
        echo "${i}" >>  "/var/tmp/echo_output.txt"
        variable=`echo "${i}" | awk '{split($0,a,","); print a[3]}'`
        echo "test test test: ${variable}" >>  "/var/tmp/echo_output.txt"
        ##echo "${i}" | awk '{split($0,a,","); print a[3]}' >>  "/var/tmp/echo_output.txt"
done
)
```

I created a manual input file for nsupdate, and used the key i set up previously to allow updates to the RPZ zone I set up for this purpose

```
nsupdate -k /home/daniel/Ktest-splunk-key.+163+35858.key -v /tmp/nsupdate
```

This worked, the new entry appeared in the RPZ zone file.

before

```
root@splunk01:~# dig rpz.splunk.local axfr @192.168.101.7

; <<>> DiG 9.11.3-1ubuntu1.13-Ubuntu <<>> rpz.splunk.local axfr @192.168.101.7
;; global options: +cmd
rpz.splunk.local.       300     IN      SOA     localhost. need.to.know.only. 201903130 60 60 432000 60
rpz.splunk.local.       300     IN      NS      LOCALHOST.
nuid.imrworldwide.com.rpz.splunk.local. 600 IN CNAME .
*.nuid.imrworldwide.com.rpz.splunk.local. 600 IN CNAME .
rpz.splunk.local.       300     IN      SOA     localhost. need.to.know.only. 201903130 60 60 432000 60
;; Query time: 1 msec
;; SERVER: 192.168.101.7#53(192.168.101.7)

root@splunk01:~#
```

after

```
root@splunk01:~# dig rpz.splunk.local axfr @192.168.101.7
```

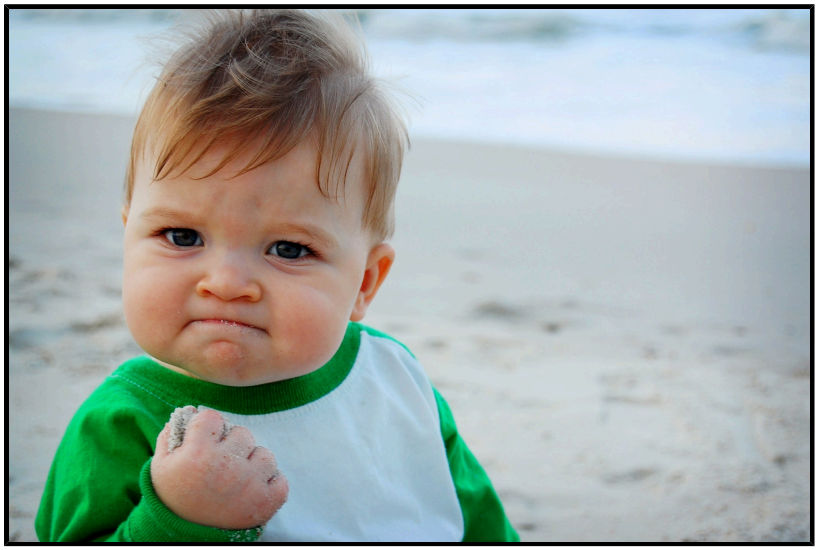
```
; <<>> DiG 9.11.3-1ubuntu1.13-Ubuntu <<>> rpz.splunk.local axfr @192.168.101.7
;; global options: +cmd
rpz.splunk.local.       300     IN      SOA     localhost. need.to.know.only. 201903130 60 60 432000 60
rpz.splunk.local.       300     IN      NS      LOCALHOST.
nuid.imrworldwide.com.rpz.splunk.local. 600 IN CNAME .
*.nuid.imrworldwide.com.rpz.splunk.local. 600 IN CNAME .
tun.home.rpz.splunk.local. 600 IN CNAME .
*.tun.home.rpz.splunk.local. 600 IN CNAME .
rpz.splunk.local.       300     IN      SOA     localhost. need.to.know.only. 201903130 60 60 432000 60
;; Query time: 1 msec
;; SERVER: 192.168.101.7#53(192.168.101.7)

root@splunk01:~#
```

After running the update, the tunnel didn't die. This puzzled me. After lots of frustration and googling, it seems the problem is that because these BIND instances are authoritative for the base tunnel domain AND I'm trying to block them with RPZ config, these two things clash. the authoritative config overrides the RPZ records. if i spent a LOT of time looking at BIND source code, i could probably find the evidence for this. so for now, it's just a theory.

After mucking about with setting up another BIND resolver separate from the two authoritatives ones for my tun.home domain, i was able to get the result i was after

```
64 bytes from 10.0.1.1: icmp_seq=65 ttl=64 time=8.21 ms
64 bytes from 10.0.1.1: icmp_seq=66 ttl=64 time=7.72 ms
64 bytes from 10.0.1.1: icmp_seq=67 ttl=64 time=7.33 ms
64 bytes from 10.0.1.1: icmp_seq=68 ttl=64 time=7.82 ms
iodine: Got SERVFAIL as reply: server failed or recursion timeout
iodine: Hmm, that's 1. Your data should still go through...
```



10. Ok, so how to REALLY stop it

As the old saying goes, *prevention is better than cure*. There are network architectural changes that can be made to make DNS tunneling impossible or at least extremely difficult. Defense in depth, these mechanisms overlap each other. You should not expect any mechanism to be 100% effective.

1. ***No public DNS resolution internally. Proxy servers do that.*** ☜
2. Block devices on trusted/sensitive networks from communicating directly with anything on the internet (or untrusted networks). Proxy all communications; web, dns, ntp, smtp etc.
3. No outbound Port-Address Translation "NAT" -> read about Samy Kamkar's work to see why (https://samy.pl/chownat/ /pwnat/ /slipstream/)

4. in-line IPS/IDS with very timely updates of IOCs and signatures. "Home grown" or reputable vendor. Test effectiveness with red team/pen-tester engagements.
5. DNS "firewalls" from a vendor like Infoblox or integrate existing DNS systems with protection config like RPZ that gets feeds from threat intel providers like https://www.farsightsecurity.com/
6. alerts and all DNS logs into central logging/SIEM platform to be used by Detect-&-Response and Threat Hunting teams

11. The bigger strategic picture

The ultimate goal is to be able to do away with DNS tunneling being a threat; remove the need to block tunneling.

Trend towards a Zero Trust network architecture; Google BeyondCorp , "Software Defined Perimeter" (Cloud Security Alliance), Implementing a Zero Trust security model at Microsoft (August 06, 2019) etc

"Identity is the perimeter". If there is no "inside" , then there is no network-centric exfiltration to protect from. Protection moves up to a higher layer, in the applications.

12. References

Resources I actually used/read

1. #SANS Institute: Using Splunk to Detect DNS Tunneling GIAC (GCIA) Gold Certification Author: Steve Jaworski, Advisor: Rick Wanner Accepted: May 31, 2016
2. #Iodine. http://code.kryo.se/iodine/
3. IETF: RFC 1034 Domain names – concepts and facilities, Internet standard, November 1987
4. IETF: RFC 1035 Domain names - Implementation and specification - Internet standard, November 1987
5. Pure Hacking: Reverse DNS Tunneling – Staged Loading Shellcode, Ty Miller, Blackhat (2008)
6. Ayaya: Black Ops of DNS, Dan Kaminsky, Blackhat (2004)Google Scholar
7. OzymanDNS – Dan Kaminsky (2004). https://dankaminsky.com/2004/07/29/51/
8. Dns2tcp - Hervé Schauer Consultants. http://www.hsc.fr/ressources/outils/dns2tcp/
9. Heyoka. http://heyoka.sourceforge.net/
10. DNScat. http://tadek.pietraszek.org/projects/DNScat/ (2004-2005) *The idea of DNScat was started by NSTX, which performs IP tunnelling over DNS.*
11. https://slashdot.org/story/00/09/10/2230242/ip-tunneling-through-nameservers (Sept 2000)
12. MagicTunnel. http://www.magictunnel.net/
13. Element53 – Sander Nijhof. https://nijhof.biz/element53/
14. VPN over DNS. https://www.vpnoverdns.com/
15. SANS Institute: Data Charging Bypass - How your IDS can help, Hassan Mourad, September 2014
16. SANS Institute: Detecting DNS Tunneling, Greg Farnham, February 2013
17. Bianco, D.: A traffic-analysis approach to detecting DNS tunnels. http://blog.vorant.com/2006/05/traffic-analysis-approach-to-detecting.html
18. Pietraszek, T.: Dnscat. http://tadek.pietraszek.org/projects/DNScat/
19. Heavy Reading: DNS Security for Service Providers: An Active Approach at L7 – White Paper – Patrick Donegan, October 2015
20. https://regexr.com/
21. RFC 1035 - Domain Names - Implementation and Specification
22. RFC 4034 - Resource records for the DNS Security Extensions
23. A guide to understanding covert channel analysis of trusted systems [Light Pink Book]

A raw list of resources I gathered via google searches. I arbitrarily stopped after a while, skimmed through them, and then got started with setting up Iodine

- https://trustfoundry.net/using-iodine-for-dns-tunneling-c2-to-bypass-egress-filtering/
- https://blogs.infoblox.com/community/analysis-on-popular-dns-tunneling-tools/
- https://unit42.paloaltonetworks.com/dns-tunneling-how-dns-can-be-abused-by-malicious-actors/
- https://labs.f-secure.com/blog/attack-detection-fundamentals-c2-and-exfiltration-lab-2/
- https://medium.com/@galolbardes/learn-how-easy-is-to-bypass-firewalls-using-dns-tunneling-and-also-how-to-block-it-3ed652f4a000
- https://navytitanium.github.io/DNSMasterChef/

- https://www.hitechnectar.com/blogs/prevent-dns-tunneling/
- https://help.zscaler.com/zia/about-dns-tunnel-detection
- https://my.socprime.com/en/integrations/dns-security-check-splunk
- https://www.daemon.be/maarten/dnstunnel.html
- https://hurricanelabs.com/blog/new-discoveries-for-dns-threat-hunting/

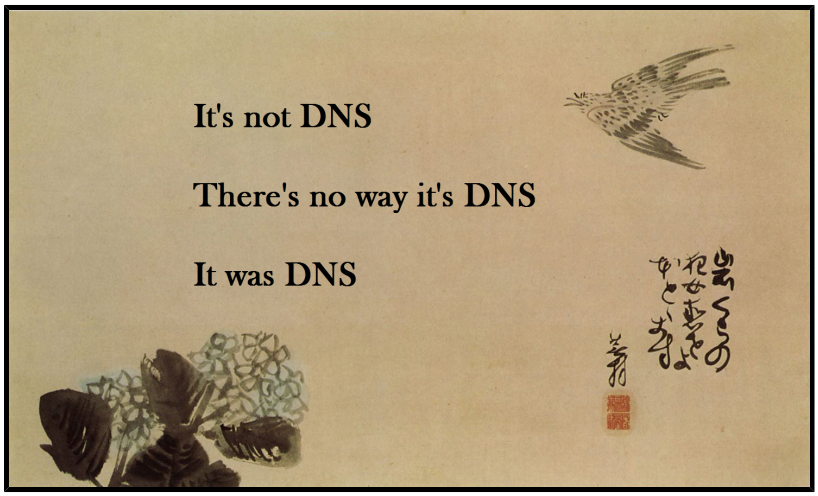
A Survey of DNS Tunnelling Detection Techniques using machine learning - paper13.pdf
Real-Time Detection of DNS Exfiltration and Tunneling from Enterprise Networks.pdf

- https://logrhythm.com/blog/detecting-dns-tunneling/
- https://www.hindawi.com/journals/scn/2018/6137098/
- https://www.elpanda.com.ar/?iodine
- https://infosectrek.wordpress.com/2016/01/18/setting-up-dns-tunneling-with-iodine-on-ubuntu-14-04/
- SlowDNS: A free VPN over DNS Tunneling Tool. http://slowdns.com/

13. Articles and discussions on DNS tunneling

Slashdot in 2000 on DNS tunneling
Information Security Magazine, May 2001
Compass Security releases DNS tunneling test tool
Discussion on the Firewall Wizards mailing list
Blog posting on the statistical detection of DNS tunneling
Blog posting on the statistical detection of DNS tunneling (ii)
Tutorial on how to use OzyManDNS
PPP over SSH over DNS Howto
NSTX howto

There's a meme for everything



Marking

copied comments from emails and openlearning posts

Kristian Mansfield

Your project sounds awesome! I've used iodine before and it's not too hard to set up, although DNS tunnelling can be difficult for some people just conceptually.

From what you've outlined already, this is a brilliant proposal. How I would mark it is as such:

- PS: Explain what DNS tunnelling is and show how to set it up
- CR: Showing examples of how to detect DNS tunnelling and what it looks like in Wireshark
- DN: Setting up methods to block DNS tunnelling without blocking legitimate requests
- HD: Detecting the traffic and automatically blocking through large amounts of network traffic.

I understand that this might come across as a lot of work – and it's meant to be to get the best marks. However, I think with the skillset you've demonstrated you have, it shouldn't be impossible for you to get the High Distinction level.

Richard Buckland

that's a very nice project daniel. work out what your deliverable will be and what your criteria for pass and for impressive are - i this this could be very fun (and useful for you in the future). i'd love it if part of your project could be explaining to others the significance of being able to do this and writing/finding some simple tools they could use themselves to experiment