Learn how easy is to bypass firewalls using DNS tunneling (and also how to block it)

By Roger Galobardes

Published: 2023-01-01 · Archived: 2026-04-29 07:20:49 UTC

9 min read

Oct 30, 2018

See how I played red team/blue team on this simple network security experiment.

Press enter or click to view image in full size

photo by Rob Bye on Unsplash

Many tutorials out there explain how to perform DNS tunneling but most of them feel like just a compilation of the commands needed to execute it, with almost no explanation on the networking background.

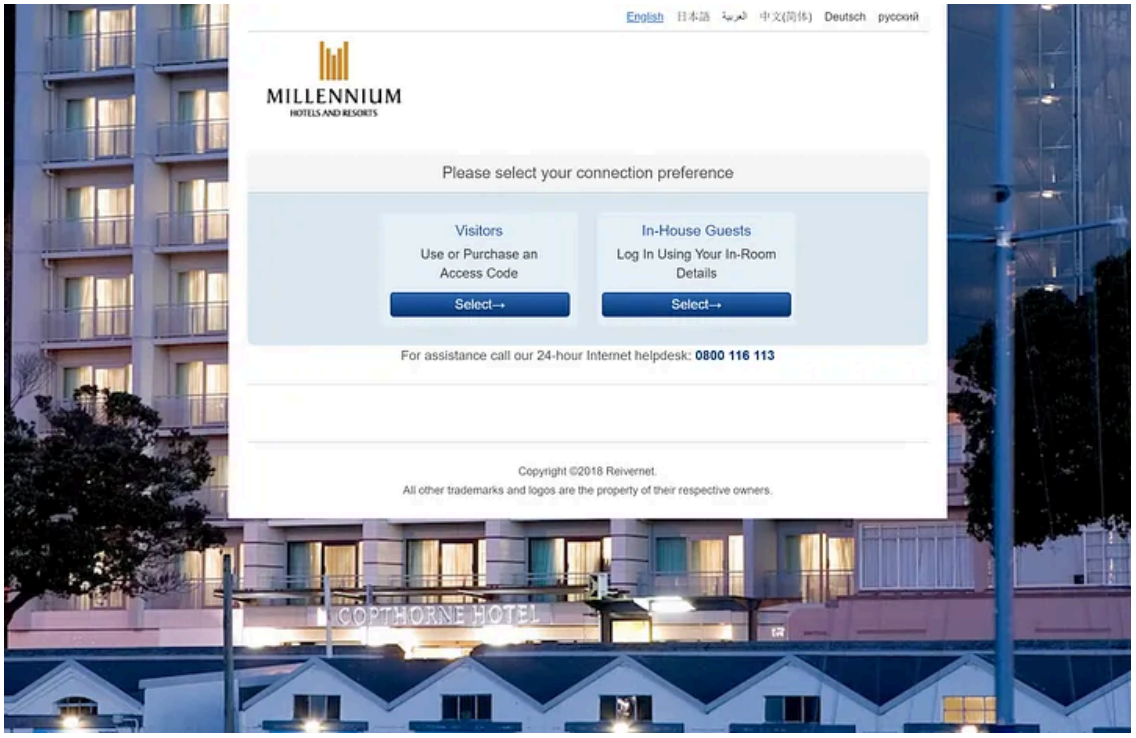
What's even worse: **No one seems to discuss how to prevent DNS tunneling from happening in your network.**

1. What is it and why would someone use it:

DNS tunneling is a method used to send data over the DNS protocol, a protocol which has never been intended for data transfer. Because of that, people tend to overlook it and **it has become a popular but effective tool in many attacks.**

Most popular use case for DNS tunneling is obtaining free internet through bypassing captive portals at airports, hotels, or if you feel patient the not-so-cheap in flight Wi-Fi.

Press enter or click to view image in full size



Hotels generally provide an access code if you ask… but we all have introvert days.

On those shared internet hotspots HTTP traffic is blocked until a username/password is provided, however DNS traffic is generally still allowed in the background: we can encode our HTTP traffic over DNS and voilà, we have internet access.

This sounds fun **but reality is, browsing anything on DNS tunneling is slow. Like, back to 1998 slow.**

Another more dangerous use of DNS tunneling would be **bypassing network security devices** (Firewalls, DLP appliances…) to set up a direct and unmonitored communications channel on an organisation's network.
Possibilities here are endless: Data exfiltration, setting up another penetration testing tool… you name it.

To make it even more worrying, there's a large amount of easy to use DNS tunneling tools out there.

There's even at least one **VPN over DNS provider** (warning: the design of the website is hideous, making me doubt on the legitimacy of it).

As a pentester all this is great, as a network admin not so much.

2. How does it work:

For those who know nothing about DNS but still made it here, I think you deserve a very brief explanation on what DNS does: **DNS is like a phonebook for the internet**, it translates URLs (human-friendly language, the person's name), into an IP address (machine-friendly language, the phone number). That helps us remember many websites, same as we can remember many people's names.

For those who know what DNS is I would suggest looking here for a quick refresh on DNS, but in short what you need to know is:

- **A Record:** Maps a domain name to an IP address.

example.com → 12.34.52.67

- **NS Record** (a.k.a. Nameserver record): Maps a domain name to a list of DNS servers, in case our website is hosted in multiple servers.

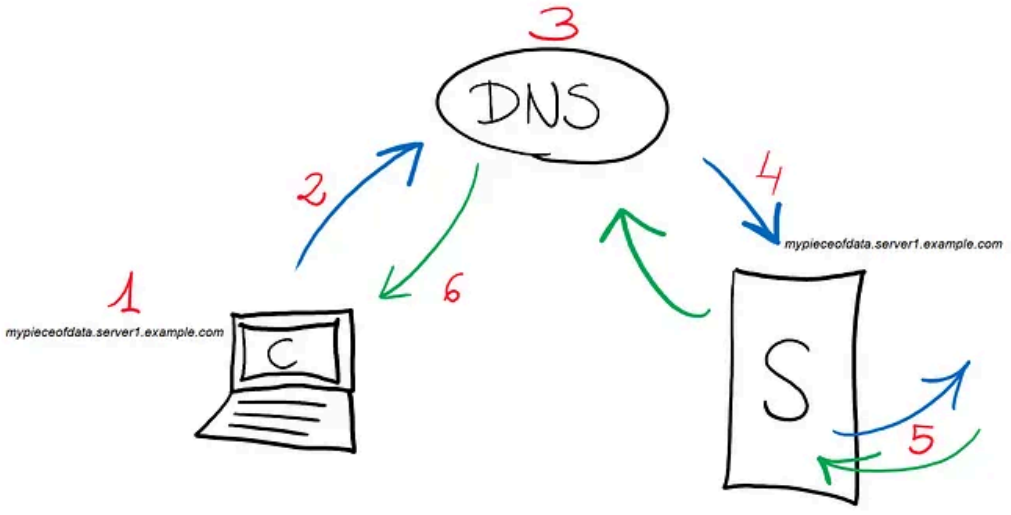
example.com → server1.example.com, server2.example.com

Who is involved in DNS tunneling?

- **Client**. Will launch DNS requests with data in them to a domain.
- One **Domain** that we can configure. So DNS servers will redirect its requests to a defined server of our own.
- **Server**. This is the defined nameserver which will ultimately receive the DNS requests.

The 6 Steps in DNS tunneling (simplified):

Press enter or click to view image in full size



I should have been an artist but I just loved subnetting too much.

1. The client encodes data in a DNS request. **The way it does this is by prepending a piece of data in the domain of the request**. For example: *mypieceofdata.server1.example.com*
2. The DNS request goes out to a DNS server.

3. The DNS server finds out the A register of your domain with the IP address of your server.
4. The request for *mypieceofdata.server1.example.com* is forwarded to the server.
5. The server processes whatever the *mypieceofdata* was supposed to do. Let's assume it was an HTTP request.
6. The server replies back over DNS and woop woop, we've got signal.

As you can imagine with this setup **the key to having a fast internet connection is low latency,** and when most airlines provide internet connections with around 1 second of ping, you might spend the whole 17 hours 40 minutes between Auckland and Doha to load this article.

3. Step by step example using iodine:

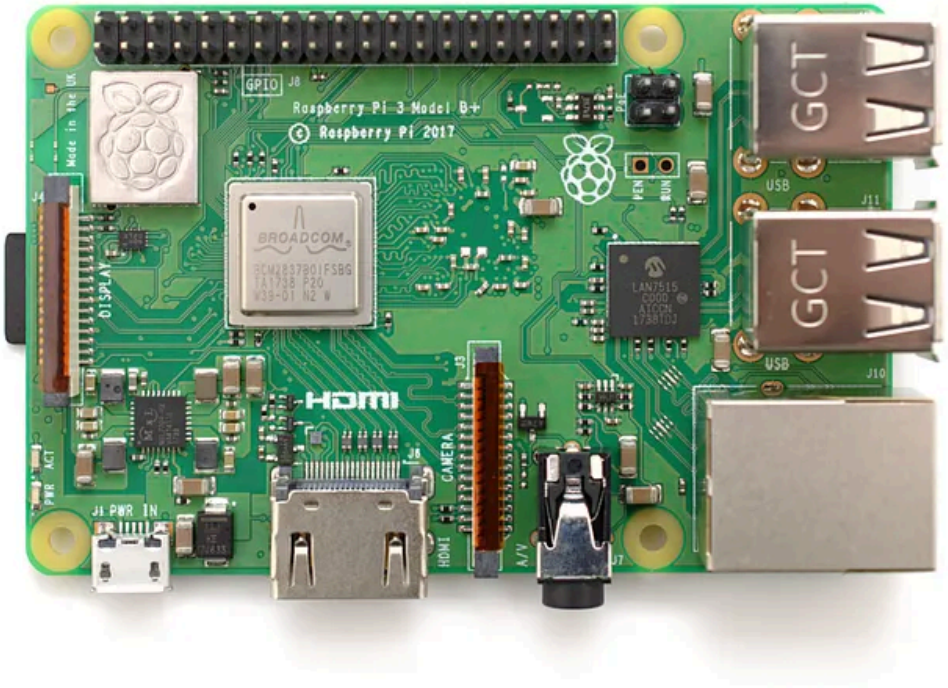
For this example we will use the famous tool iodine. From their website, the name came from this:

The name iodine was chosen since it starts with IOD (IP Over DNS) and since iodine has atomic number 53, which happens to be the DNS port number.

1. Checklist before you start — What you need:

- A domain that you can configure. Godaddy allows this for their cheap .xyz domains.
- A device that acts as a server which has a public IP address and if you have it behind a firewall, make sure it is reachable through UDP 53. For this example I am using an old Raspberry Pi but you can run iodine on a plethora of devices. (If you're reading a specific article like this I assume you know how to run linux on a Raspberry and how to install iodine!).
- A client with iodine installed. For this I will use my Kali VM but again, you can even run iodine on Windows if you want.

  Press enter or click to view image in full size



It can do more than emulating Street Fighter 2 turbo apparently.

2. Register a domain on GoDaddy and configure it like this:

A record called dnsa.exampledomain.xyz → your public IP address

NS record called t.exampledomain.xyz → dnsa.exampledomain.xyz

→ **Protip**: Having a short subdomain name like “t” will allow you to pack in more data in every single request.

You can do this with a dynamic IP but you will need to use Dynamid DNS provider and point the NS record to it.

3. Wait for it to propagate.

This could go from 1 to 24 hours.

4. Run iodine on your server:

$ sudo iodined -c -f 10.0.0.1 -P password t.exampledomain.xyz

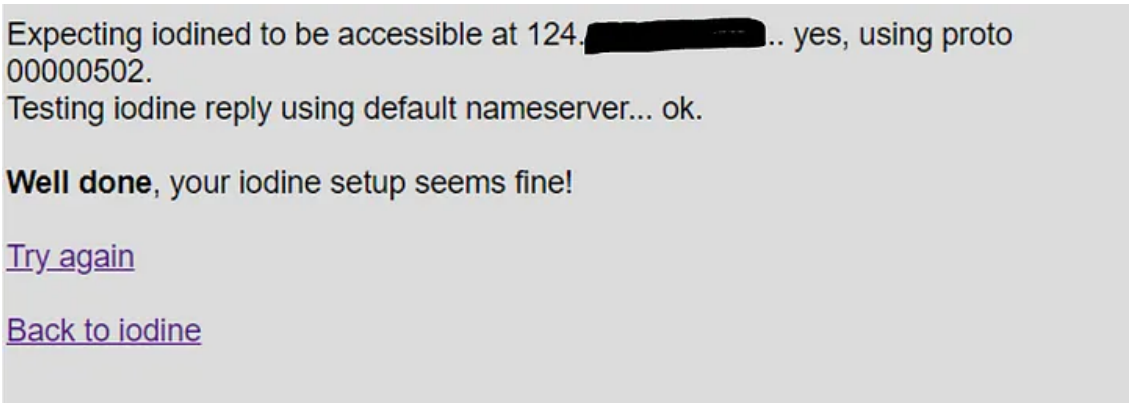
…If you copy paste that password you deserve whatever happens due to it.

5. Verify that it works:

At this point you can verify your DNS setup on the following tool from the iodine team:
https://code.kryo.se/iodine/check-it/

Iodine must be running on the server and be reachable from the internet!

Press enter or click to view image in full size



If you’re in the right path you should see this message.

It is also a good moment to check what is your client’s public IP address and note it down, that will help verify later that DNS tunneling works.

Press enter or click to view image in full size

It is important to use tools that make you look professional.

6. Run iodine on the client and establish the DNS tunnel:

Get Roger Galobardes’s stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

On the client:

root@kali:~#iodine -I 50 -f -P password t.exampledomain.xyz.

Press enter or click to view image in full size

```
Autodetecting DNS query type (use -T to override).
Using DNS type NULL queries
Version ok, both using protocol v 0x00000502. You are user #0
Setting IP of dns0 to 10.0.0.2
Setting MTU of dns0 to 1130
Server tunnel IP is 10.0.0.1
Testing raw UDP data to the server (skip with -r)
Server is at 172.22.22.7, trying raw login: ....failed
Using EDNS0 extension
Switching upstream to codec Base128
Server switched upstream to codec Base128
No alternative downstream codec available, using default (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 ok.. will use 1188-2=1186
Setting downstream fragment size to max 1186...
Connection setup complete, transmitting data.
```

iodine tries many encoding methods to see which is more suitable.

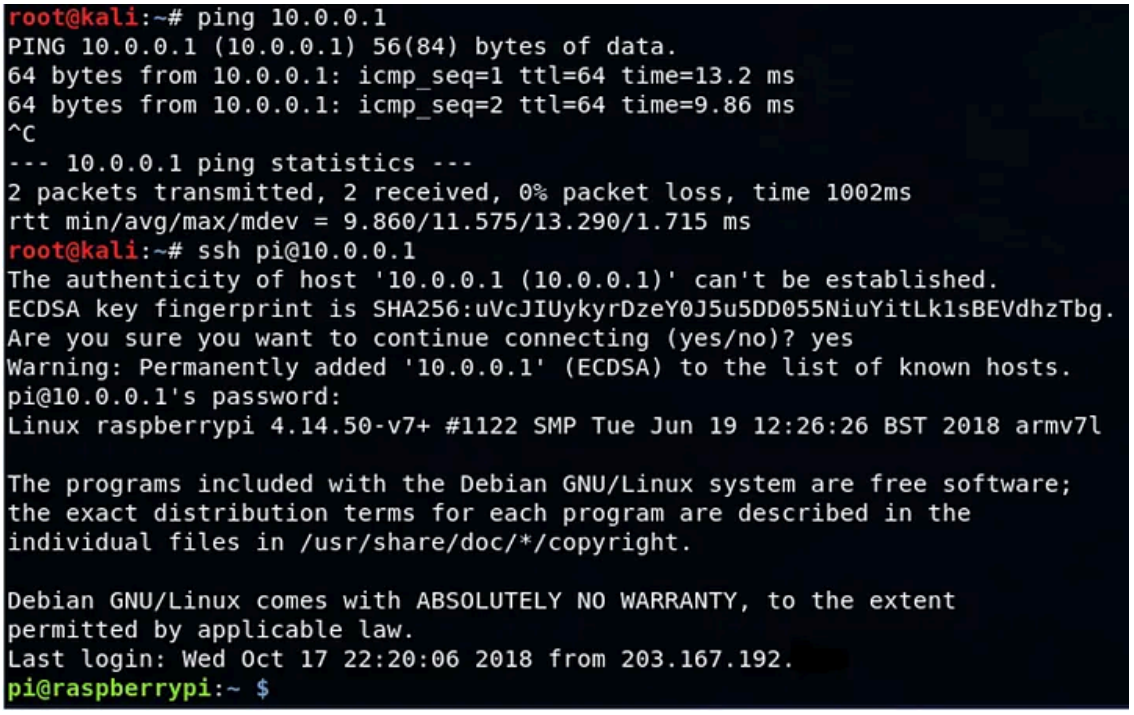
A new network interface should appear in your client, **and you should be able to ping the server's tunnel IP address we configured before.**

Press enter or click to view image in full size

```
root@kali:~# ifconfig
dns0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST>  mtu 1130
        inet 10.0.0.2  netmask 255.255.255.224  destination 10.0.0.2
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 500
(UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1  bytes 48 (48.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

dns0 will now appear under your client's network interfaces

Press enter or click to view image in full size



You should be able to ping the server, even SSH into it. All through your fresh DNS tunnel.

Now, at this point you could set up a route for your traffic to be sent to 10.0.0.1 and you'd be good to go. However DNS traffic is not encrypted so an extra step we can take is…

7. Establish an SSH tunnel over the DNS tunnel:

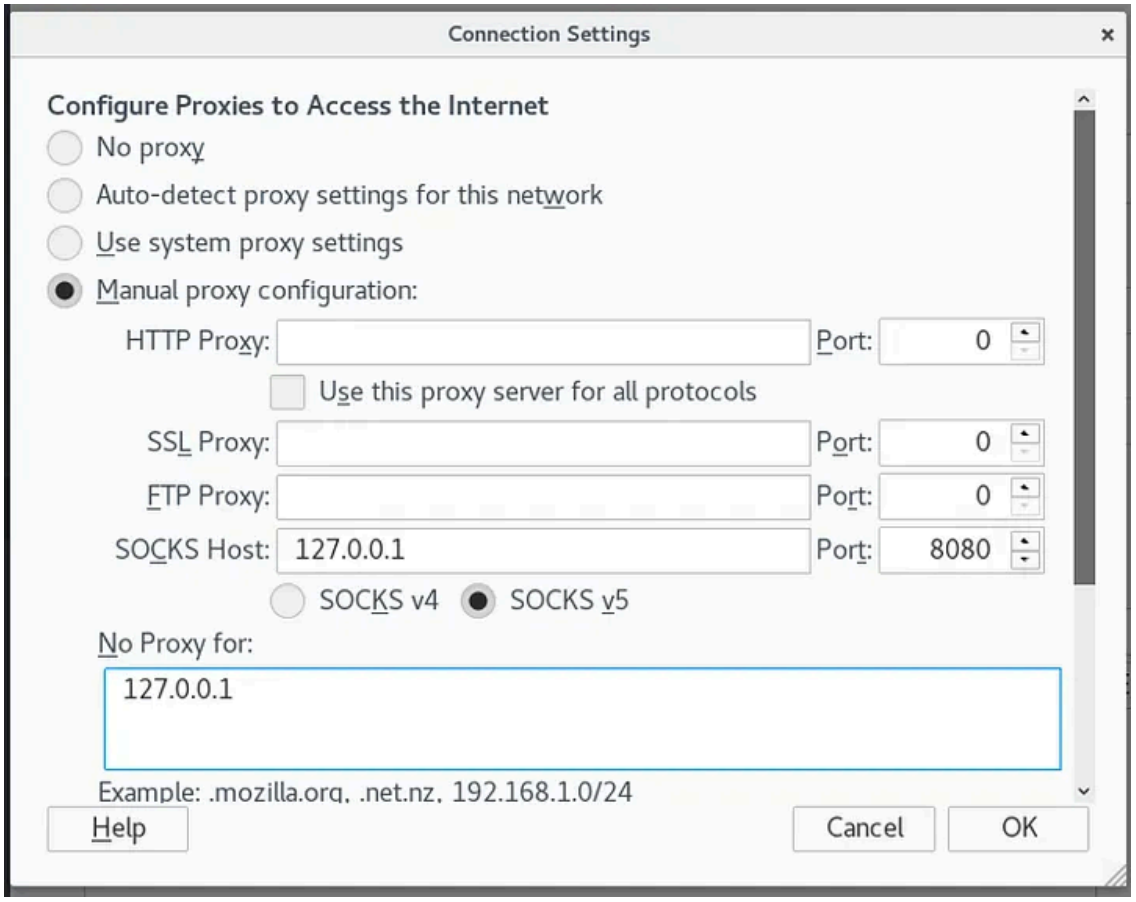
On the client establish an SSH tunnel against the server:

ssh -D 8080-N 10.0.0.1

-N makes it not execute a remote command, -D creates a socket to listen on this port and whenever a connection is done to it it forwards traffic over the SSH tunnel.

8. Change proxy settings accordingly on your browser:

Press enter or click to view image in full size



9. Enjoy your slow, yet free, internet access.

Good way to test it is checking that your public IP address on the client is now the same as on the server… Congratulations!

4. Blocking DNS tunneling on your network (with an example!):

Now this is the bit where I've struggled to find more information on.

There's many approaches you can follow to block DNS tunneling, which one to choose will depend on your current deployment:

1. Detection of known malicious domains:

This is done by analyzing the DNS queries from your network and if one of them is for a known bad domain, it is dropped.

The easiest way to implement this is by using a public DNS resolver that provides a layer of security. Free examples of this are Cloudfare (just check you're not using 1.1.1.1 in your network…), or OpenDNS (free for home use). Enterprise solutions include some of the Infoblox products or Cisco Umbrella to name the most common ones.

Alternatively some firewalls provide URL security and might be able to block known domains.

For this method you are relying on a blacklist of known malicious domains, so if you are victim of a targeted or small scale attack, chances are this is going to be useless.

2. Traffic Analysis:

More of a reactive measure, however if we have an idea of the average DNS requests in your network or hosts you should be able to detect when there is a sudden increase in DNS activity.

Again, there are many tools you can use for this. From firewalls that have application visibility (most NGFW should do this), network flow analysis tools, or dedicated DNS security appliances like the ones from Infoblox.

3. Intrusion Prevention Systems:

Some IPS systems such as Snort will include detect packets unique to iodine:

> *# alert udp $EXTERNAL_NET 53 -> $HOME_NET any (msg:"APP-DETECT iodine dns tunnelling handshake server ACK"; flow:to_client; byte_test:1,&,0x80,2; content:"|00 01 00 01 00|"; depth:5; offset:4; content:"v"; within:1; distance:4; content:"VACK"; within:200; fast_pattern; metadata:service dns; reference:url,code.kryo.se/iodine/README.html; classtype:policy-violation; sid:27046; rev:2;)*

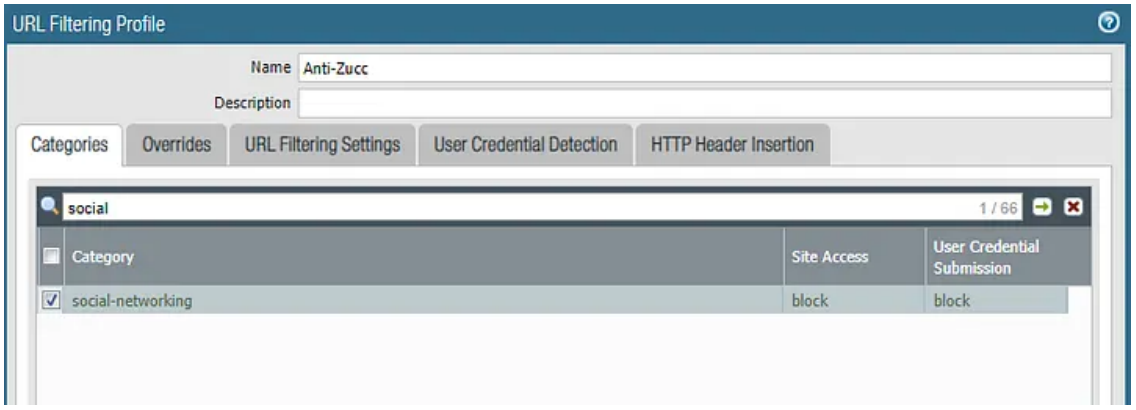
Some other more generic rules will detect long URLs in DNS requests, **these rules that can easily be bypassed** by shortening the payload in every request (although slowing down the tunnel speed).

Blocking DNS Tunnel — An example:

During my Iodine deployment I was sending all traffic through a Palo Alto Networks firewall in my environment. The plan was to play first the attacker role and later the defender.

First I configured a basic URL filtering rule to block social network websites:

Press enter or click to view image in full size



I blocked social networking sites on the URL filtering engine and soon I was browsing them again.

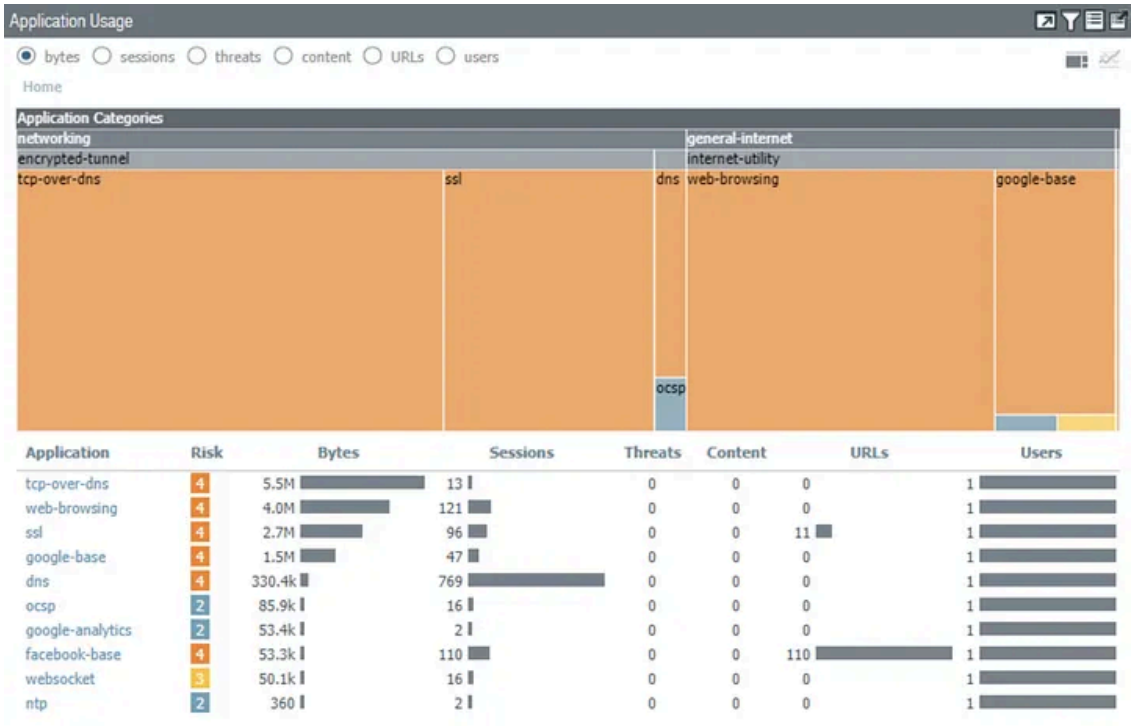
In seconds I established my DNS tunnel… and it worked! Soon I was loading Facebook.

I was surprised on how easy it was to bypass the URL filtering engine, and then I proceeded to switch teams and see how could I block this.

The starting point would be traffic analysis, I expected a big amount of DNS traffic and then some convoluted solution to block sudden changes in traffic patterns.

However this is what I found to my surprise:

Press enter or click to view image in full size

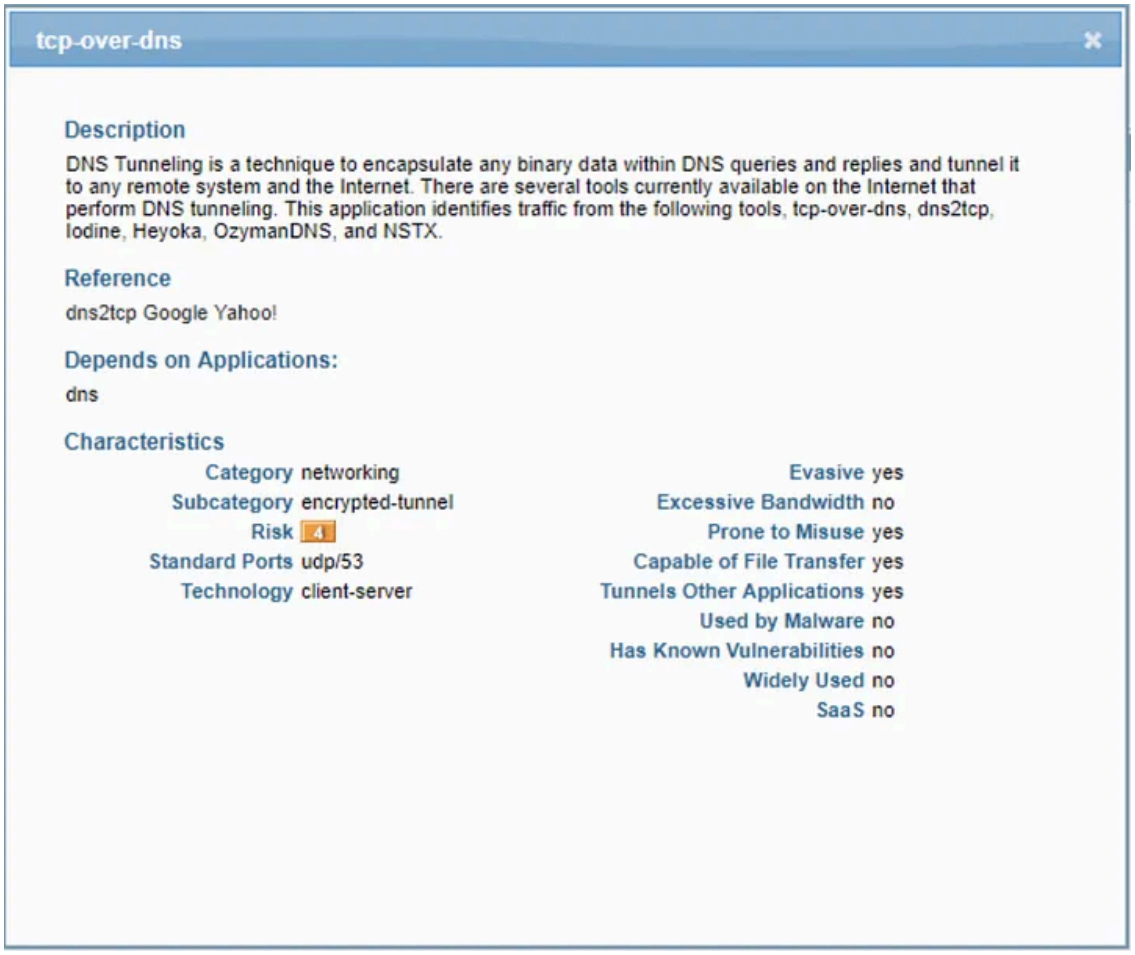


Basic application usage analysis on a Palo Alto Networks firewall

The firewall was detecting that I was sending TCP traffic over DNS. But how?

Clicking on the application details I could see the following:

Press enter or click to view image in full size

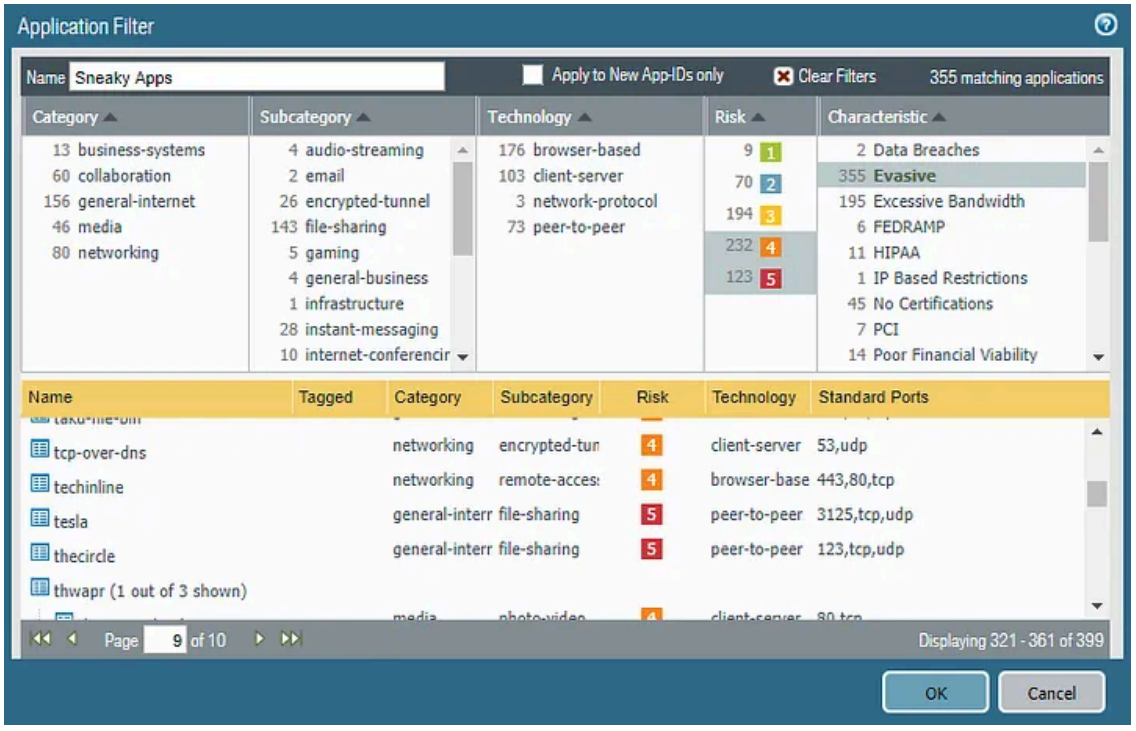


It looks like the application detection engine detects traffic from most DNS tunneling tools, in a similar way as we saw that Snort has a couple of rules to detect Iodine traffic, and puts them under the same category.

Grouping all these types of apps into one application group makes it more easy to manage than having a single Snort alert.

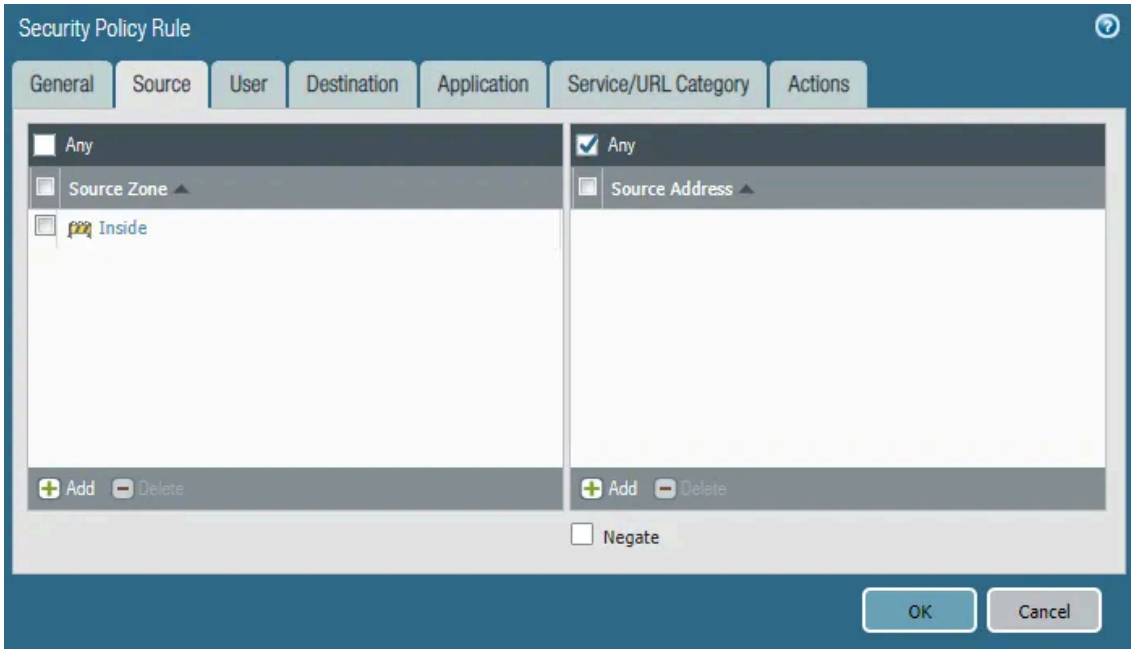
Let’s create a rule now that will block these type of applications:

Press enter or click to view image in full size

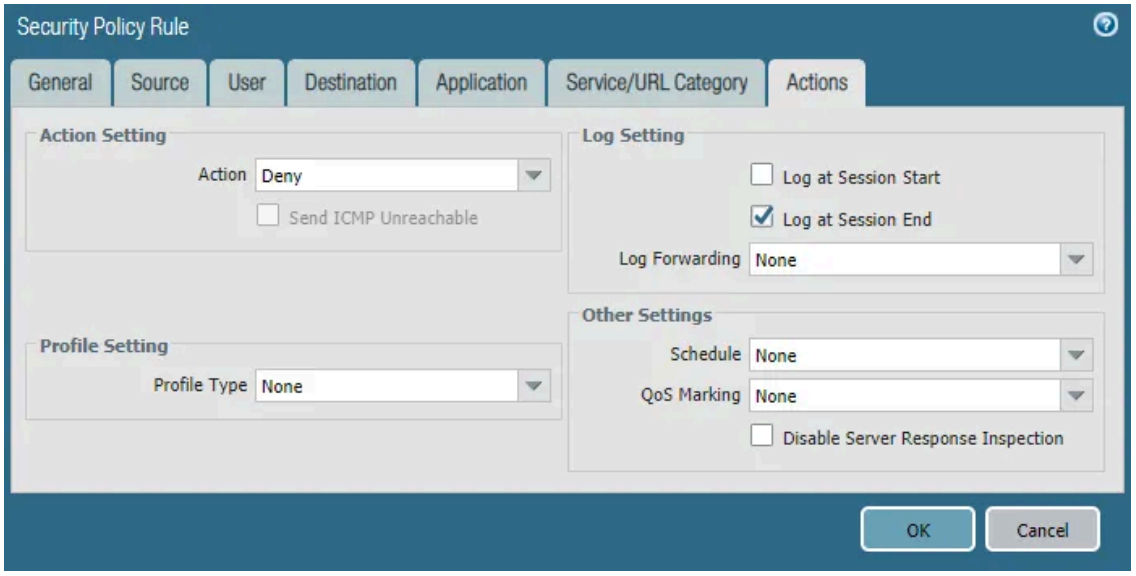


First I create an Application Filter for Evasive apps with a high risk rating. Notice tcp-over-dns is in the list.

Press enter or click to view image in full size



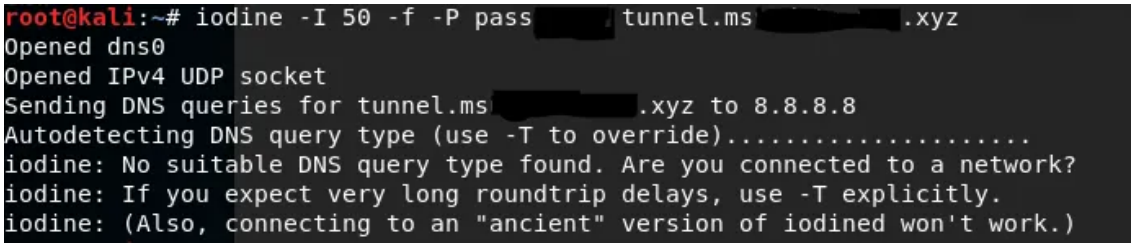
Second I define that this will be for traffic sourced from inside my network.



And finally, we make the rule Deny all this traffic. [Check this article to see what is the difference between a Deny or a Drop](#).

Easy! Let's see if it works:

First we run Iodine…



.. and apparently iodine can't establish a tunnel anymore!

Job's done! Our network should now be secured against most DNS tunneling attacks.

To simulate a true real world scenario I should switch sides again and try to bypass again this firewall but being honest, that would be way more complex.

Anything you didn't like? Please tell me why!
