# Troubleshooting DNS resolution problems

## Introduction

OpenVPN Access Server supports pushing an instruction to a connecting OpenVPN client to use a specific DNS server. Actually it supports pushing 2 DNS servers, in case the first one fails to respond. This can be configured in the Admin UI under VPN Settings. The Access Server also supports sending additional instructions for DNS Resolution Zones, which functions like a type of split-DNS where only queries for a specific DNS zone are sent to the VPN server, and DNS Default Suffix, which provides a hint to Windows to 'autocomplete' a partial hostname to a Fully Qualified Domain Name, or FQDN.

Unfortunately, not every operating system behaves the same in regards to DNS. Some systems will try all DNS servers at once, and accept the response from the first to respond. Others will be able to do split-DNS, and others will not. This can lead to certain problems. The guide below provides a way of checking to see if the DNS query you are doing from your OpenVPN client device, is actually making it through the VPN tunnel to the OpenVPN Access Server. And from there, of course, to the target DNS server. This information is valuable in determining whether or not the problem is at the client end, or at the server end.

## Testing DNS resolution from a client system

We are going to assume that you have a DNS server configured in the Admin UI of the Access Server, under VPN Settings. We are assuming you are not using the DNS Resolution Zones or the DNS Default Suffix fields. With this setting, all DNS request should be going from the OpenVPN client, through the OpenVPN Access Server, and then to the specified DNS server. In our example we are pushing the Google Public DNS server 8.8.8.8, and our test results will reflect this in the sample outputs as well.

Install your OpenVPN client program on your chosen client system. In our example we will be using a Windows 10 Professional client system with the OpenVPN Connect Client installed, and connected to the OpenVPN Access Server. Next open a console session or an SSH session to the OpenVPN Access Server, and obtain root privileges. We will be using the tool **tcpdump** to monitor activity on port 53 TCP and UDP, the default port where DNS queries are handled. We will be flushing the local DNS resolver cache on the client side, and then resolve a number of domains simply by pinging them by name. In our test situation, there are only a handful of clients connected, and the activity of DNS queries is very low, so we can monitor it easily. If you are testing on a production system and the **tcpdump** command gives too much output, you can append a grep filter by IP address, to filter queries coming only from your specific VPN client's IP address, to make reading and locating the DNS query results easier.

On the Access Server run these commands:

```
apt-get update
apt-get install tcpdump
```

With TCPdump installed, now run it with these parameters:

```
tcpdump -eni any port 53
```

Or, if you want to filter it by the IP address of your VPN client (adjust as needed):

```
tcpdump -eni any port 53 | grep "172.27.10.22"
```

With this running in the background, go to your VPN client's operating system, and open a command prompt. On Windows for example you can run the **cmd** program to open an old style DOS prompt. With that open, use the following commands to wipe the local DNS resolver cache, so it won't pull results from its own local memory, and then do an actual query.

Wipe local DNS resolver cache on Windows:

```
ipconfig /flushdns
```

Resolve some domain names:

```
ping www.google.com
ping www.openvpn.net
ping www.facebook.com
```

Each of these should yield results that look somewhat like this:

```
Pinging www.google.com [216.58.212.228] with 32 bytes of data:
Reply from 216.58.212.228: bytes=32 time=4ms TTL=56
Reply from 216.58.212.228: bytes=32 time=3ms TTL=56
Reply from 216.58.212.228: bytes=32 time=3ms TTL=56
Reply from 216.58.212.228: bytes=32 time=3ms TTL=56
Ping statistics for 216.58.212.228:
 Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
 Minimum = 3ms, Maximum = 4ms, Average = 3ms
```

On the OpenVPN Access Server you should be seeing results that look somewhat like this:

```
18:03:07.976553 In ethertype IPv4 (0x0800), length 76: 172.27.232.2.49531 > 8.8.8.8.53: 53268+ A?
www.google.com. (32)
18:03:07.976579 Out 00:0c:29:c7:60:e9 ethertype IPv4 (0x0800), length 76: 192.168.47.133.49531 >
8.8.8.8.53: 53268+ A? www.google.com. (32)
18:03:07.981162 In 34:31:c4:8e:b5:67 ethertype IPv4 (0x0800), length 92: 8.8.8.8.53 >
192.168.47.133.49531: 53268 1/0/0 A 216.58.211.100 (48)
18:03:07.981181 Out ethertype IPv4 (0x0800), length 92: 8.8.8.8.53 > 172.27.232.2.49531: 53268 1/0/0
A 216.58.211.100 (48)
```

The above result from tcpdump shows that a DNS request was received from the VPN client at 172.27.232.2, and that it was directed at the DNS server at 8.8.8.8, and the request was to find the A record (IP address) for the DNS name www.google.com. The first line shows that this request is coming in at the OpenVPN Access Server, from the VPN client. The second line shows the request leaving the Access Server through the network interface with MAC address 00:0c:29:c7:60:e9. In our test setup, this is the network interface of the Access Server that goes to the Internet, which makes sense, because the 8.8.8.8 DNS server is on the Internet. The third line shows that a DNS result has been received, and the fourth line shows that this result has been relayed back to the VPN client. In this case, DNS resolution is working.

## Split-DNS when using DNS Resolution Zones

Split-DNS is the principle of resolving only certain zones (domains) through a DNS server pushed by the VPN server, and the rest through your already present local DNS servers. In Access Server there is a field in the Admin UI, under VPN Settings, called **DNS Resolution Zones**. If you enter a single domain or a list of (comma-separated) domains here, then the clients will receive an instruction to only resolve those domains through the DNS server pushed by the VPN server, and resolve the rest through the client's local DNS server.

Please note that not all OpenVPN clients out there support this and there are some differences in behavior between versions of OpenVPN as well. The best results can be achieved by using OpenVPN Connect v3 client software.

When you use split-DNS, you will not see the DNS server that is being pushed in your ipconfig or ifconfig output. The DNS server will not get implemented at the network interface configuration level. Instead, it will be implemented in the DNS system in a DNS resolution policy table. On mac OS for example this can be queried using the **scutil** command line utility and on Windows this can be queried using **netsh** to query the resolution policy table in the OS. Such a table is simply a list of domains, and which DNS servers they should be resolved through. Below we will show example output of how split-DNS and normal DNS resolution looks like through a VPN tunnel. Some superfluous data has been removed from these example outputs.

Commands to see network configuration and DNS resolution policy on Windows:

```
ipconfig /all
netsh namespace show effectivepolicy
```

Commands to see network configuration and DNS resolution policy on mac OS:

```
ifconfig
scutil -dns
```

Example output on Windows when split-DNS is currently in use:

```
(OpenVPN) adapter Local Area Connection:
Description . . . . . . . . . . . : TAP-Windows Adapter V9 for OpenVPN Connect
DNS Servers . . . . . . . . . . . : fec0:0:0:ffff::1%1
                                    fec0:0:0:ffff::2%1
                                    fec0:0:0:ffff::3%1

Ethernet adapter Ethernet:
Description . . . . . . . . . . . : Intel(R) 82574L Gigabit Network Connection
DNS Servers . . . . . . . . . . . : 192.168.47.254

DNS Effective Name Resolution Policy Table Settings
Settings for .openvpn.net
----------------------------------------------------------------------
Generic (DNS Servers) : 1.2.3.4
```

*In the above output, you can see that split-DNS is now being used because the DNS server assigned to the normal network interface called Ethernet that connects to our local network has DNS server 192.168.47.254, which is our local DNS server, and the Name Resolution Policy Table has a zone for .openvpn.net that resolves through 1.2.3.4, which is the DNS server pushed by the VPN server. This means that *.openvpn.net will get resolved through the VPN DNS server, and the rest will resolve through the local DNS server 192.168.47.254. Note also that the VPN interface gets 3 IPv6 self-assigned DNS server addresses, which are not assigned by OpenVPN, but by the OS itself. This should not affect DNS resolution.*

Example output on Windows when split-DNS is **not** used:

```
(OpenVPN) adapter Local Area Connection:
  Description . . . . . . . . . . . : TAP-Windows Adapter V9 for OpenVPN Connect
  DNS Servers . . . . . . . . . . . : 1.2.3.4

DNS Effective Name Resolution Policy Table Settings
Settings for .
----------------------------------------------------------------------
Generic (DNS Servers) : 1.2.3.4
```

*In the above output, you can see that split-DNS is **not** being used because the DNS server is assigned to the network interface adapter itself, and there is only one top level zone for DNS resolution (the dot means all zones). This means that this configuration is not using split-DNS and therefore all DNS queries get redirected to the server at 1.2.3.4.*

## Troubleshooting

Below are a number of common problems you can see that we try to explain here and where to look for a solution.

**Ping request could not find domain (...). Please check the name and try again** This can happen when the DNS servers your client system is using is badly configured, cannot be reached, or if the DNS server it is using does not know the domain you are trying to resolve. For example with local DNS servers in your own network it is entirely possible that they only know local computer systems, and have no knowledge of online names like openvpn.net or such. Usually in such a case you can configure the DNS server to forward DNS queries to a public DNS server that does know the answer to those queries, so that it is able to respond to both queries for local names and also public names. A useful step in this situation may be to again run tcpdump as described in the testing DNS resolution from a client system section above, and checking to see what the output of tcpdump is. If you see a result like this:

```
18:07:10.082330 In ethertype IPv4 (0x0800), length 94: 172.27.232.2.54519 > 8.8.8.8.53: 50281+ A?
thisdomainreallydoesnotexist.com. (50)
18:07:10.082356 Out 00:0c:29:c7:60:e9 ethertype IPv4 (0x0800), length 94: 192.168.47.133.54519 >
8.8.8.8.53: 50281+ A? thisdomainreallydoesnotexist.com. (50)
18:07:10.082507 In ethertype IPv4 (0x0800), length 94: 172.27.232.2.57858 > 8.8.8.8.53: 65054+ AAAA?
thisdomainreallydoesnotexist.com. (50)
18:07:10.082521 Out 00:0c:29:c7:60:e9 ethertype IPv4 (0x0800), length 94: 192.168.47.133.57858 >
8.8.8.8.53: 65054+ AAAA? thisdomainreallydoesnotexist.com. (50)
18:07:10.103610 In 34:31:c4:8e:b5:67 ethertype IPv4 (0x0800), length 167: 8.8.8.8.53 >
192.168.47.133.54519: 50281 NXDomain 0/1/0 (123)
18:07:10.103641 Out ethertype IPv4 (0x0800), length 167: 8.8.8.8.53 > 172.27.232.2.54519: 50281
NXDomain 0/1/0 (123)
```

Specifically the item **NXDomain** here is important. It means that this DNS server does not know the name we are trying to resolve. Another DNS might still know the name. but this one doesn't. In the example above however we have purposefully selected a name that does not exist (or at least it didn't when we ran the test - it is possible of course someone may register the name in the future) to be sure we see the error. If you are encountering this problem you may want to try to use the **nslookup** program on a computer with direct access to the DNS server, and use it to query the specific DNS server directly, to confirm that it does know the domain.

If you see a result like this, repeated a few times:

```
18:19:29.935439 Out 00:0c:29:c7:60:e9 ethertype IPv4 (0x0800), length 76: 192.168.47.133.60180 >
1.2.3.4.53: 16427+ AAAA? www.google.com. (32)
18:19:29.935479 In ethertype IPv4 (0x0800), length 76: 172.27.232.3.51334 > 1.2.3.4.53: 37513+ A?
www.google.com. (32)
```

Then what you may notice here is that you do see a query arriving from the VPN client, pass through the Access Server, and go out to the Internet, but there is no reply. Usually this means that this DNS server is unreachable, or is not a DNS server at all. In the example I have chosen IP address 1.2.3.4 which I know for a fact is not a DNS server. Obviously the query will be repeated a few times but will ultimately fail. The obvious solution here is to choose a DNS server that works, or, to make sure that there is no firewall standing in the way, blocking the queries from the VPN clients to the DNS server. In some cases, when routing is used to give VPN clients access to servers on the private network behind the Access Server, it is a matter of a missing route. In such a case that packets from VPN clients make it to the target DNS server just fine, but it is not able to respond because it is receiving

packets from a subnet it does not know how to respond to. That can be solved by implementing static routes for direct VPN client communication, or switching to giving access using NAT instead. In other cases we've seen, especially on Windows Server platforms, the built-in Windows Firewall could be blocking queries coming from a subnet outside of the local network. In such a case an adjustment to the firewall is necessary to allow the DNS server to receive the query and respond to it.