

2022

Block DNS over HTTPS (DoH), using pfsense



<https://github.com/jpgpi250>

14-12-2022

1.	About this manual.	1
2.	DoH lists.....	1
3.	Pfsense configuration.....	2
1.	Defining the firewall aliases URLs.....	2
2.	Defining the DoH block rules.....	3
3.	Defining the exception aliases.....	4
4.	Defining the exception rules.	7
4.	List Updates.	9
5.	Sqlite3 database.	9
1.	Tables.....	10
2.	Query examples.....	11
6.	Network control, using DNS entries.	13
1.	Firefox canary domain.....	13
2.	iCloud Private Relay.....	13
7.	DoH response policy zone.	15
8.	Cloudflare addresses.	15
9.	Limiting the number of exceptions.	16
10.	CIDR (network) Exceptions.....	16
11.	DOH Suricata rules.....	18
12.	Change Log.	20

1. About this manual.

If you are reading this document, using Adobe Reader, you may click on a hyperlink to content in this document. Use the combination <Alt> <left arrow> to return to the previous location.

"Back" and "Forward" buttons can also be added to the toolbar. If you right-click on the toolbar, under "Page Navigation", they are referred to as "Previous View" and "Next View".

This document is hosted on GitHub, you can open the document (pdf), using this [link](#).

2. DoH lists.

DNS over HTTPS (DoH) is a protocol for performing remote [Domain Name System](#) (DNS) resolution via the [HTTPS](#) protocol. You can find a lot of detail on [wikipedia](#).

This document describes a method to prevent (block) clients on your network to use DoH.

In short, we will simply block all the IP's of DoH DNS servers on the firewall. Since DoH servers come and go all the time, it is hard to keep track of all DoH servers. Several

contributors dedicate time to creating lists with known DoH servers, unfortunately, most of these lists are incomplete or not maintained.

I've searched and found several lists, containing references to DoH servers:

- <https://raw.githubusercontent.com/bambenek/block-doh/master/doh-hosts.txt>
- <https://raw.githubusercontent.com/Sekhan/TheGreatWall/master/TheGreatWall.txt>
- <https://raw.githubusercontent.com/oneoffdallas/dohservers/master/list.txt>
- <https://raw.githubusercontent.com/vysecurity/DoH-Servers/master/README.md>
- <https://raw.githubusercontent.com/tjay/DoH-List/master/hosts>
- <https://raw.githubusercontent.com/flo-wer/doh-list/master/domains.txt>
- <https://raw.githubusercontent.com/wiki/curl/curl/DNS-over-HTTPS.md>
- <https://download.dnscrypt.info/dnscrypt-resolvers/json/public-resolvers.json>
- <https://dtm.uk/dns-over-https-doh-servers>
- <https://raw.githubusercontent.com/Jigsaw-Code/Intra/master/Android/app/src/main/res/values/servers.xml>
- <https://raw.githubusercontent.com/dibdot/DoH-IP-blocklists/master/doh-ipv4.txt>
- <https://raw.githubusercontent.com/dibdot/DoH-IP-blocklists/master/doh-ipv6.txt>
- <https://raw.githubusercontent.com/crypt0rr/public-doh-servers/main/dns.list>
- <https://raw.githubusercontent.com/xorguy/block-dot-doh/main/Lists/SDNS-Domains.list>
- <https://raw.githubusercontent.com/jbaggs/doh-intel/master/doh.intel>
- <https://raw.githubusercontent.com/mili-tan/ArashiDNS.Dekunua/main/DoH.list>
- <https://ahadns.com/dns-over-https/>
- <https://adguard-dns.io/kb/general/dns-providers/>

Feel free to [report](#) any other list you find.

The above lists are used to build new lists, containing IP addresses, that can be used on a firewall. This process runs daily, the resulting github files will be updated, as soon as a change is detected.

This document describes what you need to do to use the IP lists on pfSense. If your firewall doesn't support this, you might want to add the DoH [response policy zone](#) (unbound, bind, knot resolver, ...).

3. PfSense configuration.

There are different ways to use the IP block lists and IP exceptions lists. Any solution will do, as long as the result is DoH being blocked, except for the [specific devices](#) in the [exception rules](#).

I use floating rules for all LAN interfaces, except the interface I'm using in my test environment. I don't want to define additional rules on the WAN interface, to allow access to all DoH IPs on the test interface, hence targeting the LAN interface. This document describes the approach for a single LAN interface.

1. Defining the firewall aliases URLs.

The [Github repository](#) contains two relevant files, we need to create URL aliases for each list. The lists (how to use them will be explained later):

- [DOHipv4.txt](#): This list contains the IPv4 addresses of all DoH servers found in the lists.

- [DOHipv6.txt](#): This list contains the IPv6 addresses of all DoH servers found in the lists.

Goto 'Firewall / Aliases / URL's' and click 'Add'

Create all (two) the aliases (one if you're not using IPv6), using the data from the table, ensure type 'URL Table (IPs)' is selected, set the update frequency to '1'. Feel free to choose a name, description, the document will use the names from the table.

Name	URL
DoHserversIPv4	https://raw.githubusercontent.com/jpgpi250/piholemanual/master/DOHipv4.txt
DoHserversIPv6	https://raw.githubusercontent.com/jpgpi250/piholemanual/master/DOHipv6.txt

The result should look like this (example):

Properties

Name

DoHserversIPv4

The name of the alias may only consist of the characters "a-z, A-Z, 0-9 and _".

Description

DoHipv4 list, retrieved from github

A description may be entered here for administrative reference (not parsed).

Type

URL Table (IPs)

URL Table (IPs)

Hint

Enter a single URL containing a large number of IPs and/or Subnets. After saving, the URLs addresses will be created. This will work with large numbers of addresses (30,000+) or sm:
The value after the "/" is the update frequency in days.

URL Table (IPs)

https://raw.githubusercontent.com/jp

/ 1

github DOHipv4.txt

2. Defining the DoH block rules.

Goto 'Firewall / Rules / WAN' and click 'Add (rule to the top of the list)

- Action: Block

Action Block

- Select Quick

Quick ☒ Apply the action immediately on match.

- Interface: LAN

Interface LAN

- Address Family: Select the correct Address family (list IPv4 or IPv6)

Address Family

IPv4

Protocol: TCP/UDP

Protocol

TCP/UDP

Source: Any

Source

☐ Invert match

any

Destination:

Single host or alias, select the correct alias (DoHserversIPv4 or DoHserversIPv4)

Destination

☐ Invert match

Single host or alias

DoHserversIPv4

Destination Port Range: 443

Destination Port Range

HTTPS (443)

From





Custom

To

HTTPS (443)

- Optionally, enable logging and enter a comment.

You should now have 2 rules (if you're using IPv6), the result should look like this (I have logging enabled, hence the extra icon):

<input type="checkbox"/>	States	Interfaces	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description
<input type="checkbox"/>	  0 / 571 KiB	LAN	IPv4 TCP/UDP	*	*	DoHserversIPv4	443 (HTTPS)	*	none		block DoHIPv4 port 443
<input type="checkbox"/>	  0 / 659 KiB	LAN	IPv6 TCP/UDP	*	*	DoHserversIPv6	443 (HTTPS)	*	none		block DoHIPv6 port 443

If you have a firewall with multiple LAN ports, you could choose to use floating rules, allowing a specific adapter to use DoH (for test purposes).





A floating rule has an additional field.

Direction: any

Direction

any

A floating rule configuration would look like this (the WAN and OPT2 adapter have been excluded from the rule, this allows me to test DoH on a specific network segment):

<input type="checkbox"/>	States	Interfaces	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description
<input type="checkbox"/>	  0 / 571 KiB	LAN OPT1 WIFI	IPv4 TCP/UDP	*	*	DoHserversIPv4	443 (HTTPS)	*	none		block DoHIPv4 port 443
<input type="checkbox"/>	  0 / 659 KiB	LAN OPT1 WIFI	IPv6 TCP/UDP	*	*	DoHserversIPv6	443 (HTTPS)	*	none		block DoHIPv6 port 443

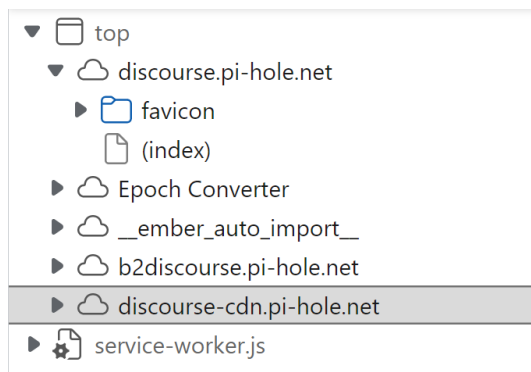
3. Defining the exception aliases.

The entries on the list are now blocked, using the above rules. Unfortunately the IP for the domain `doh1.b-cdn.net` (example) is the same as the IP for `discourse-cdn.pi-hole.net`. This will break the browsing experience. We need to ensure browsers can load pages that use content from this IP, without simply removing the entry from the list.

Examples of web pages that require an exception:

- discourse.pi-hole.net
- docs.pi-hole.net
- weboost.com

Looking at, for example `discourse.pi-hole.net`, it's clear the site uses content from `discourse-cdn.pi-hole.net`, which is behind the same IP as `doh1.b-cdn.net`.



After defining the above block rules, browsing to `discourse.pi-hole.net` (the pi-hole forum), the following firewall entries will show up in the log (if logging is enabled).

Last 100 Firewall Log Entries. (Maximum 100) Pause					
Action	Time	Interface	Source	Destination	Protocol
✗	Aug 5 10:18:46	LAN	192.168.2.228:2913	84.17.46.53:443	TCP:S
✗	Aug 5 10:18:46	LAN	192.168.2.228:2912	84.17.46.53:443	TCP:S

The IP addresses for `doh1.b-cdn.net` needs to be included (user responsibility) in the aliases `DOHserverExceptionsIPv4` and `DOHserverExceptionsIPv6`.

The idea is to use these aliases (exceptions) to allow **specific devices** to bypass the DoH block rules, defined in the previous section. This disables DoH protection, using the firewall rules, for the specific devices only.

Previous versions of this manual used predefined exception lists, hosted on GitHub. This method has been **deprecated**, due to the excessive growth of these lists, as a result of [issues](#).

Larger environments may be using URL aliases to achieve this, in small environments, the easiest way to maintain aliases with the IPs of specific devices and exceptions, is to create IP aliases:

Goto 'Firewall / Aliases / IP' and click 'Add'

Create all (four) the aliases, using the data from the table, ensure type 'Host(s)' is selected. Feel free to choose a name, description, the document will use the names from the table.

Name	Description
DoHserverExceptionsIPv4	static DoH exceptions IPv4
DoHserverExceptionsIPv6	static DoH exceptions Ipv6
DoHclientExceptionsIPv4	IPv4 hosts, allowed to bypass specific DoH IPv4 entries
DoHclientExceptionsIPv6	IPv6 hosts, allowed to bypass specific DoH IPv6 entries

Add IP addresses and descriptions as needed. **You might want to read the section that explains how to [limit the number of exceptions](#)!**

Examples of the result:

- IPv4 address of DoH server that should be unblocked for specific clients (evidently, you need to add additional IP's that need unblocking – maintain the list)

Properties

Name

DoHserverExceptionsIPv4

The name of the alias may only consist of the characters "a-z, A-Z, 0-9 and _".

Description

static DoH exceptions IPv4

A description may be entered here for administrative reference (not parsed).

Type

Host(s)

▼

Host(s)

Hint

Enter as many hosts as desired. Hosts must be specified by their IP address or fully qualified domain name (FQDN). FQDN hostnames are periodically re-resolved and updated. If multiple IPs are returned by a DNS query, all are used. An IP range such as 192.168.1.1-192.168.1.10 or a small subnet such as 192.168.1.16/28 may also be entered and a list of individual IP addresses will be generated.

IP or FQDN

84.17.46.53

discourse-cdn.pi-hole.net // doh1

- IPv4 address of client that should be allowed to use the exception alias (add clients as required).

Properties	
Name	<input type="text" value="DoHclientExceptionsIPv4"/> <div>The name of the alias may only consist of the characters "a-z, A-Z, 0-9 and _".</div>
Description	<input type="text" value="IPv4 devices, allowed to bypass specific DoH IPv4 entries"/> <div>A description may be entered here for administrative reference (not parsed).</div>
Type	<input type="text" value="Host(s)"/>
Host(s)	
Hint	Enter as many hosts as desired. Hosts must be specified by their IP address or fully re-resolved and updated. If multiple IPs are returned by a DNS query, all are used. Aliases like 192.168.1.16/28 may also be entered and a list of individual IP addresses will be derived.
IP or FQDN	<input type="text" value="192.168.2.176"/> <input type="text" value="Yoga2Pro13"/>

- Example of an IPv4 range
- If you will be using IP host(s) and IP network(s), you'll need to create **separate aliases and rules**, pfSense aliases don't allow mixing hosts and networks in the same alias!

Properties	
Name	<input type="text" value="DoHcidrExceptionsIPv4"/> <div>The name of the alias may only consist of the characters "a-z, A-Z, 0-9 and _".</div>
Description	<input type="text" value="static DoH exceptions IPv4 (CIDR)"/> <div>A description may be entered here for administrative reference (not parsed).</div>
Type	<input type="text" value="Network(s)"/>
Network(s)	
Hint	Networks are specified in CIDR format. Select the CIDR mask that pertains to each entry. /128 specifies a single IPv6 host, /24 specifies 255.255.255.0, /64 specifies a normal IPv4 (FQDNs) may also be specified, using a /32 mask for IPv4 or /128 for IPv6. An IP range may also be entered and a list of CIDR networks will be derived to fill the range.
Network or FQDN	<input type="text" value="103.21.244.0"/> <input type="text" value="/"/> <input type="text" value="22"/> <input type="text" value="https://www.cloudflare.com/ips/"/>

[poisonsnak](#) pointed out [here](#) he has defined all the [cloudflare address ranges](#) as an exception, not recommended, but I understand the idea (peace of mind) behind the decision. More details [here](#).

4. Defining the exception rules.

Create the firewall rules to allow specific devices to connect to specific IP's, port 443 (example dns.cloudflare.com, see above).

Goto 'Firewall / Rules / WAN' and click 'Add (rule to the top of the list)

- Action: Pass

Action

- Select Quick

Quick ☒ Apply the action immediately on match.

- Interface: WAN

Interface

- Address Family: Select the correct Address family (list IPv4 or IPv6)

Address Family

- Protocol: TCP/UDP

Protocol

- Source:

Single host or alias, select the correct alias (DoHclientExceptionsIPv4 / DoHclientExceptionsIPv6)

Source			
Source	<input type="checkbox"/> Invert match	<input type="text" value="Single host or alias"/>	<input type="text" value="DoHclientExceptionsIPv4"/>

- Destination:

Single host or alias, select the correct alias (DoHserverExceptionsIPv4 / DoHserverExceptionsIPv6)









Destination			
Destination	<input type="checkbox"/> Invert match	<input type="text" value="Single host or alias"/>	<input type="text" value="DoHserverExceptionsIPv4"/>

- Destination Port Range: 443

Destination Port Range
From Custom To

- Optionally, enable logging and enter a comment.

You should now have 4 rules, 2 allow rules, 2 block rules (if you're using IPv6). You can change the order of the rules, by dragging the rule to the desired position (don't forget to save, after changing the order). The result should look like this:

Rules (Drag to Change Order)											
<input type="checkbox"/>	States	Interfaces	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description
<input type="checkbox"/>	  0 / 8.63 MiB	LAN	IPv4 TCP/UDP	DoHclientExceptionsIPv4	*	DoHserverExceptionsIPv4	443 (HTTPS)	*	none		allow DoHipv4 port 443
<input type="checkbox"/>	  0 / 571 KiB	LAN	IPv4 TCP/UDP	*	*	DoHserversIPv4	443 (HTTPS)	*	none		block DoHipv4 port 443
<input type="checkbox"/>	  0 / 0 B	LAN	IPv6 TCP/UDP	DoHclientExceptionsIPv6	*	DoHserverExceptionsIPv6	443 (HTTPS)	*	none		allow DoHipv6 port 443
<input type="checkbox"/>	  0 / 659 KiB	LAN	IPv6 TCP/UDP	*	*	DoHserversIPv6	443 (HTTPS)	*	none		block DoHipv6 port 443

Again, if you have a firewall with multiple LAN ports, you could choose to use floating rules.

4. List Updates.

The lists are generated and pushed to GitHub (if there are changes) daily at 04:30 UTC, pfsense refreshes the content of URL Table IPs aliases, using a cron job.

I have modified this cron job, this to ensure pfsense will be using the updated lists at the beginning of the workday.

```
44 7 * * * root /usr/bin/nice -n20 /etc/rc.update_urltables now forceupdate
```

Optionally, install the cron package (System / Package Manager) to change the cron job.

It's also possible to create a script to force update the lists.

Create a script /root/updatelists.sh and make it executable, content:

```
/usr/bin/nice -n20 /etc/rc.update_urltables now forceupdate
```

Selecting option 8 (Shell) on the pfsense console will allow you to execute:

```
./updatelists.sh
```

You can verify the successful execution, using the web interface (Status / System Logs / System / General), you will see something like this (using local copy of the lists).

Jun 16 10:06:47	php-cgi	rc.update_urltables: /etc/rc.update_urltables: Updated DOHipv6 content from http://192.168.2.57/DOHservers/DOHipv6.txt: no changes.
Jun 16 10:06:47	php-cgi	rc.update_urltables: /etc/rc.update_urltables: Updated DOHipv4 content from http://192.168.2.57/DOHservers/DOHipv4.txt: no changes.

5. Sqlite3 database.

The lists are generated, using dig replies in my region (Lat.Long 51.2211097,4.3997081). Unfortunately, some domains (example dns.nextdns.io) resolve into a region dependent IP address. The IP(s) for nextdns.io, retrieved in my region are thus blocked, but these

addresses may not block DOH in your region. To overcome this problem, and allow users to generate localized DOH block lists, I have created a [database](#), containing the information, used to generate the IP lists.

You might want to read section 6 ([Network control, using DNS entries](#)). It also contains information regarding localized replies for the domains *mask.apple-dns.net* and *mask-t.appledns.net*. The files in the [GitHub](#) repository ([DOHipv4.txt](#) and [DOHipv6.txt](#) only contain relay entries for my region (Lat.Long 51.2211097,4.3997081).

The database can be accessed, using *sqlite3* commands (OR *pihole-FTL sqlite3*), to make life easier, take a look at my [pi-hole manual](#), section 26/3 (Browsing the FTL database). You'll need to make some changes in */var/www/html/phpliteadmin.config.php*, this to include the new database. A script to install this on [Raspberry Pi OS Lite](#) can be found [here](#).

1. Tables

- **info:**
 - version: If you're going to write a script to build your own lists, ensure to verify the database version. The version may change, as I'm still trying to improve the quality of the resulting lists.
 - latest_timestamp: used to identify entries that are added or updated during the last run (daily).
- **urllist:** contains the url of the source lists, see [DOH lists](#). Every list has an id, used to identify the list. The timestamp indicates when the last change to the list was detected. A 'null' value indicates there has been no change since the list was added.
- **domainlist:** contains all of the domains found in all of the lists, which implies there are duplicates. Example: the domain "*dns.adguard.com*" is found in 11 lists, thus has 11 domainlists entries. See [query examples](#) to retrieve this info.

```
1 https://raw.githubusercontent.com/Sekhan/TheGreatWall/master/TheGreatWall.txt|dns.adguard.com
2 https://raw.githubusercontent.com/oneoffdallas/dohservers/master/list.txt|dns.adguard.com
3 https://raw.githubusercontent.com/flo-wer/doh-list/master/domains.txt|dns.adguard.com
4 https://raw.githubusercontent.com/wiki/curl/curl/DNS-over-HTTPS.md|dns.adguard.com
5 https://download.dnscrypt.info/dnscrypt-resolvers/ison/public-resolvers.json|dns.adguard.com
6 https://raw.githubusercontent.com/dibdot/DoH-IP-blocklists/master/doh-ipv4.txt|dns.adguard.com
7 https://raw.githubusercontent.com/dibdot/DoH-IP-blocklists/master/doh-ipv6.txt|dns.adguard.com
```

However, the unique_id will be equal for all entries.

Every entry also has a timestamp, which is updated, whenever the lists are processed (daily). If a domain is no longer on a source list, the timestamp will NOT be updated.

Several methods are used to extract the domains from the source lists (different format such as plain text, csv, html, json, ...). As a result, some domains need to be excluded from the list. Example: "*github.com*" is mentioned on several html pages, this domain will thus be excluded (not added to the database). Another example: "*meganerd.nl*" is a valid domain, the DOH server ("*chewbacca.meganerd.nl*", according to the lists) has a different IP address. Thus, "*meganerd.nl*" is excluded (not added to the database), "*chewbacca.meganerd.nl*" is included.

- **cnameinfo:** every unique domain entry is processed (dig), this to retrieve IP addresses. Some of the list entries are actually CNAMEs. Example: the lists contain an entry "*adbblock.lux1.dns.nixnet.xyz*". This is actually a cname for "*lux1.nixnet.xyz*". This information is stored in the cnameinfo table. Be aware the cnameinfo table does NOT contain the CNAME(s), only the relevant info, as a result from the DNS query.

The domainlist_id points to the entry in the domainlist table.

The timestamp is updated every time the lists are processed (daily). If a DNS query no longer returns this info, the timestamp will NOT be updated.

2. Query examples

I'm NOT a database expert, so there may be room for improvement... Some examples, I use, to get relevant information from the database. The examples are formatted to be used in a bash script (#!/bin/bash).

You need to change the location of the database to reflect the actual location.

- Get last detected list change date ('null' implies no change since the list was first added)

```
sqlite3 /home/pi/DOH/sqlite3/DOH.db \  
"SELECT address, date(timestamp, 'unixepoch') FROM urllist;"
```

- Get list(s) that contain specific domain:

```
sqlite3 /home/pi/DOH/sqlite3/DOH.db \  
"SELECT address, domain FROM 'domainlist' \  
INNER JOIN 'urllist' ON urllist.id = domainlist.urllist_id \  
WHERE domainlist.domain = 'dns.adguard.com';"
```

- Get domains for a specific CNAMEinfo entry

```
sqlite3 /home/pi/DOH/sqlite3/DOH.db \  
"SELECT domain, urllist_id, cname_domain, unique_id FROM 'cnameinfo' \  
JOIN 'domainlist' ON domainlist_id = domainlist.unique_id \  
WHERE cnameinfo.cname_domain = 'lux1.nixnet.xyz.';"
```

- Get unique domains

```
sqlite3 /home/pi/DOH/sqlite3/DOH.db \  
"SELECT DISTINCT domain FROM 'domainlist';"
```

OR, using the 'latest_timestamp' value (only list the domains detected in the last version of the source lists):

```
sudo sqlite3 /home/pi/DOH/sqlite3/DOH.db \  
"SELECT DISTINCT domain FROM 'domainlist' \  
WHERE domainlist.timestamp = \  
(SELECT value FROM info \  
WHERE property = 'latest_timestamp');"
```

This query result can be used to generate a custom pi-hole list. Redirect the output to a file, a new local pi-hole blocklist, thus:

```
sudo sqlite3 /home/pi/DOH/sqlite3/DOH.db \
    "SELECT DISTINCT domain FROM 'domainlist' \
    WHERE domainlist.timestamp = \
    (SELECT value FROM info \
    WHERE property = 'latest_timestamp');" \
    | sudo tee /var/www/html/DOHdomains.txt
```

Add a local pi-hole list (Group Management / Adlists), do NOT include the quotes:

- `"http://localhost/DOHdomains.txt"`
- OR
- `"file:///var/www/html/DOHdomains.txt"`

Pi-hole users can check if any of their clients has submitted a DNS query to pi-hole by executing the following query, **you don't have to use the proposed blocklist to run this query**:

```
sqlite3 "/home/pi/DOH/sqlite3/DOH.db" \
    "ATTACH database '/etc/pihole/pihole-FTL.db' as 'FTLdb'; \
    SELECT DISTINCT domainlist.domain, queries.client \
    FROM domainlist \
    JOIN FTLdb.queries ON domainlist.domain = queries.domain;"
```

Ideally, this would return an empty result. I've been testing a lot on both my pi and workstation, my (partial) result looks like this:

```
adblock.lux1.dns.nixnet.xyz|127.0.0.1
adblock.lux1.dns.nixnet.xyz|192.168.2.228
chewbacca.meganerd.nl|127.0.0.1
chewbacca.meganerd.nl|192.168.2.227
doh.captnemo.in|127.0.0.1
doh.captnemo.in|192.168.2.228
lux1.nixnet.xyz|127.0.0.1
```

- Get unique domains, display all fields

```
sqlite3 /home/pi/DOH/sqlite3/DOH.db \
    "SELECT * FROM 'domainlist' group by domain;"
```

- Get unique domains, display all fields, order by specific field

```
sqlite3 /home/pi/DOH/sqlite3/DOH.db \
    "SELECT * FROM 'domainlist' \
    GROUP BY domain \
    ORDER BY unique_id;"
```

6. Network control, using DNS entries.

Unfortunately, some lists already contain entries for domains that should NOT be blocked. These domains are used for controlling the use of DoH on a network level. If the DNS queries are not answered correctly (NXDOMAIN), blocking them will cause delays.

1. Firefox canary domain.

Firefox has provided a [method](#) to prevent the browser from using DoH.

In order to ensure the browser doesn't use DoH, you should ensure the DNS query for the domain *use-application-dns.net* is answered with NXDOMAIN.

In the latest pi-hole version, the reply for this domain is hardcoded, so no additional action is needed.

If you don't use pi-hole and the DNS reply for this domain (`dig use-application-dns.net`) isn't NXDOMAIN, you can:

- Use [dnsmasq](#) to ensure the query is answered correctly, by adding a dnsmasq configuration line:

```
server=/use-application-dns.net/
```

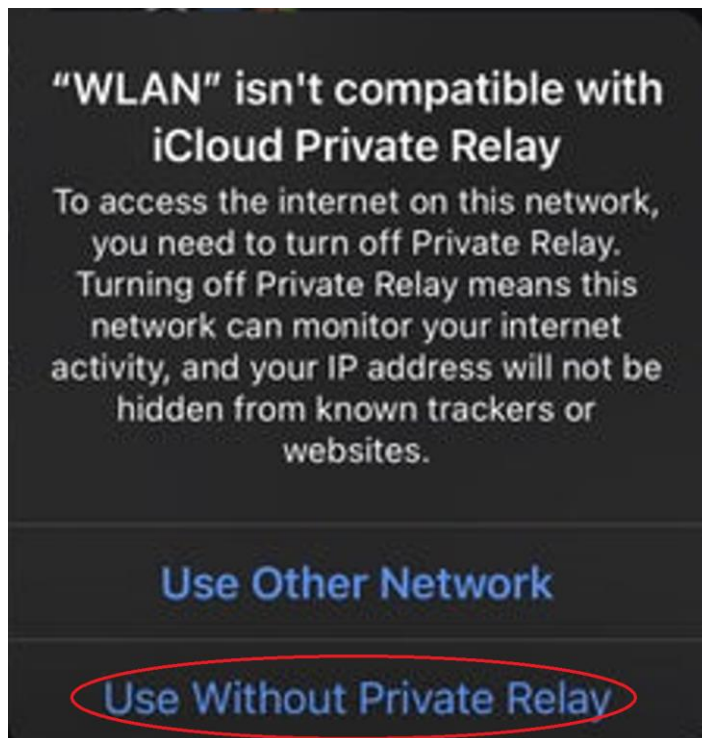
- Use [unbound](#) to ensure the query is answered correctly, by adding unbound configuration lines:

```
server:  
    local-zone: " use-application-dns.net." always_nxdomain
```

2. iCloud Private Relay.

Apple uses an implementation of oDoH ([oblivious DoH](#)), this to increase privacy. When enabled (default), the system no longer uses the configured DNS server(s), provided by DHCP. Apple has provided a [method](#) to ensure users get a warning. The user then needs to disable the use of iCloud Private Network for the network he is connected to.

Example, warning for WLAN (NXDOMAIN reply for the domain(s) received), the user needs to select "Use without Private Relay", this to avoid using oDoH (iCloud Private Relay).



In order to ensure the system does warn the user about iCloud Private Relay(s) (oDoH), you should ensure the DNS queries for the domains *mask.apple-dns.net* and *mask-t.appledns.net* are answered with NXDOMAIN.

In the latest pi-hole version, the reply for these domains is hardcoded, so no additional action is needed.

If you don't use pi-hole and the DNS reply for these domains isn't NXDOMAIN, you can:

- Use [dnsmasq](#) to ensure the queries are answered correctly, by adding dnsmasq configuration lines:

```
server=/mask.icloud.com/  
server=/mask-h2.icloud.com/
```

- Use [unbound](#) to ensure the queries are answered correctly, by adding unbound configuration lines:

```
server:  
  local-zone: " mask.icloud.com." always_nxdomain  
  local-zone: " mask-h2.icloud.com." always_nxdomain
```

The normal response (not yet changed to NXDOMAIN) for the above domains is localized, the response contains the addresses of the local available relays, the system will use, different for each region. If you want to ensure the relays in your region are blocked,

using the firewall rules, you need to generate the IP lists yourself, using the [database](#), see section 5 ([Sqlite3 database](#)). A list of all relays can be found [here](#).

7. DoH response policy zone.

Not all firewalls are capable of implementing this. To provide protection for (o)DoH, the domains are added to the [rpz file](#), can be used to ensure unbound (bind and knot resolver also support this – not tested) returns an appropriate response for these domains. More on this in this [document](#). As an alternative, if possible, you can use suricata to reject queries for (o)DoH domains, see [here](#).

It may be wise to implement the RPZ feature, even if your firewall is configured as described in this document. You probably already have created the exception aliases and rules, this to ensure the websites, using resources from the same IP as a known DoH server, are accessible. A client / app may be attempting to use DoH, due to the exceptions this may work. The domain entry in the RPZ provides protection, if the client uses your DNS server to get the IP for the DoH server.

The unbound documentation for response policy zones can be found [here](#).

8. Cloudflare addresses.

Unfortunately, some DoH servers are hosted on cloudflare, the IP address can change over time. This makes it harder to maintain exception lists. You can use [grepcidr](#) to determine if an IP address is part of a CIDR network.

The cloudflare CIDR ranges can be found [here](#), [IPv4](#) and [IPv6](#).

A simple bash script can be used to determine if an IP address is a cloudflare address, provide the IP to test as script parameter (reference [here](#))

- Result is IP -> part of cloudflare CIDR.
- No Result -> NOT part of cloudflare CIDR.

Example IPv4:

```
#!/bin/bash
IPv4cidr=""
while read -r network; do
    IPv4cidr="${IPv4cidr} ${network}"
done <<(wget -qO - https://www.cloudflare.com/ips-v4 2>&1 && echo "")
grepcidr "${IPv4cidr}" <(echo "$1")
```

Assuming you have a local copy of the IP bloclist(s) (from github) and the cloudflare CIDR lists, you can use the following command to list all IP addresses in the cloudflare CIDR networks. Example ipv4 (add -c to get only the count.):

```
grepcidr -f /home/pi/ips-v4 /var/www/html/DOHservers/DOHipv4.txt
```


9. Limiting the number of exceptions.

Some DoH servers are hosted by external providers who use cidr addresses. The address, returned, changes regularly. When using the online [DNSchecker](#) on the domain docs.pi-hole.net, you'll notice different addresses. When using [DNSQuery](#) (enter the IP address in the IP Whois Query box), you can see the (google) CIDR range. Without additional configuration, this would imply you'll need to add exceptions for all the addresses, returned (The address even changes within a region).

You can overcome this by configuring unbound (specific syntax for other resolvers not listed) with a redirect assignment for docs.pi-hole.net

```
server:

    local-zone: "docs.pi-hole.net." redirect

    local-data: "docs.pi-hole.net. A 34.159.132.250"

    local-data: "docs.pi-hole.net. AAAA 2a05:d014:275:cb01:8909:43f0:2069:7b77"
```

Domains, using CNAME entries might require additional configuration entries, example, the domain discourse-cdn.pi-hole.net requires:

```
server:

    local-zone: "discourse-cdn.pi-hole.net." redirect

    local-data: "discourse-cdn.pi-hole.net. A 185.93.2.248"

    local-data: "discourse-cdn.pi-hole.net. AAAA 2400:52e0:1e02::932:1"

    local-zone: "b2discourse.pi-hole.net." redirect

    local-data: "b2discourse.pi-hole.net. A 185.93.2.248"

    local-data: "b2discourse.pi-hole.net. AAAA 2400:52e0:1e02::932:1"

    local-zone: "b2discourse.b-cdn.net." redirect

    local-data: "b2discourse.b-cdn.net. A 185.93.2.248"

    local-data: "b2discourse.b-cdn.net. AAAA 2400:52e0:1e02::932:1"
```

The above unbound configuration implies you only need to create an exception for a single IPv4 and IPv6 address per domain, eliminating the need to add exceptions, due to ever changing addresses.

The down side of this method is, of course, the website may migrate to another provider, which will require reconfiguration.

10. CIDR (network) Exceptions.

If the above solution isn't an option, you'll need to create exceptions for an address range.

Example, creating a cidr (network) exception for discourse.pi-hole.net:

'dig +short discourse.pi-hole.net' will return a single address (52.14.183.198). The address will probably be different when you try it, the address changes regularly, thus creating an IP address exception for this domain may work for a while, once the address changes, your exception won't work anymore.

To determine the required cidr (network) exception you need to prevent this from happening (example), use a browser to retrieve the info (IP address from the dig result:

```
https://api.bgpview.io/ip/52.14.183.198
```

The result:

```
{ "status": "ok", "status_message": "Query was successful", "data": { "ip": "52.14.183.198", "ptr_record": "discourse.pi-hole.net", "prefixes": [ { "prefix": "52.14.0.0/16", "ip": "52.14.0.0", "cidr": 16, "asn": { "asn": 16509, "name": "AMAZON-02", "description": "Amazon.com, Inc.", "country_code": "US" }, "name": "AT-88-Z", "description": "Amazon Technologies Inc.", "country_code": "US" } ], "rir_allocation": { "rir_name": "ARIN", "country_code": null, "ip": "52.0.0.0", "cidr": 10, "prefix": "52.0.0.0/10", "date_allocated": "1991-12-19 00:00:00", "allocation_status": "allocated" }, "iana_assignment": { "assignment_status": "legacy", "description": "Administered by ARIN", "whois_server": "whois.arin.net", "date_assigned": null }, "maxmind": { "country_code": null, "city": null } }, "@meta": { "time_zone": "UTC", "api_version": 1, "execution_time": "242.03 ms" } }
```

In order to ensure discourse.pi-hole.net will always work, you'll need to create an exception alias:

Firewall / Aliases / Edit

Properties

Name

DoHserverExceptionsIPv4cidr

The name of the alias may only consist of the characters "a-z, A-Z, 0-9 and _".

Description

static DoH exceptions IPv4 (cidr)

A description may be entered here for administrative reference (not parsed).

Type

Network(s)

Network(s)

Hint

Networks are specified in CIDR format. Select the CIDR mask that pertains to each entry. /32 specifies a single host, /24 specifies a normal IPv4 network, etc. Hostnames (FQDNs) may also be entered and a list of IP ranges may also be entered.

Network or FQDN

52.14.0.0

/

16

discourse.pi-hole.net

And create a firewall rule to allow these cidr (network) exceptions. Make sure the rule is above the block rule (order). Screenshot is an implementation of floating rules.

<input type="checkbox"/>		0 / 0 B	LAN, OPT1, OPT2, WIFI	IPv4 TCP/UDP	DoHclientExceptionsIPv4 *	DoHserverExceptionsIPv4	443 (HTTPS)	*	none	allow DoHIPv4 port 443
<input type="checkbox"/>		0 / 153.42 MiB	LAN, OPT1, OPT2, WIFI	IPv4 TCP/UDP	DoHclientExceptionsIPv4 *	DoHserverExceptionsIPv4cidr	443 (HTTPS)	*	none	allow DoHIPv4 (cidr) port 443
<input type="checkbox"/>		0 / 1 KiB	LAN, OPT1, OPT2, WIFI	IPv4 TCP/UDP	*	DoHserversIPv4	443 (HTTPS)	*	none	block DoHIPv4 port 443

11. DOH Suricata rules.

Not all firewalls are capable of implementing this. To provide protection for (o)DoH, the domains are added to the suricata rules file, can be used to ensure suricata rejects DNS queries for these domains. These rules should be used only when using **Inline IPS Mode**. Only Inline IPS Mode can selectively drop packets. **Legacy Mode will block ALL traffic to / from the target IP address**, this may likely not what you want. As an alternative, if possible, you can use response policy zones (RPZ) to block queries for (o)DoH domains, see [here](#).

It may be wise to implement the suricata rules, even if your firewall is configured as described in this document. You probably already have created the exception aliases and rules, this to ensure the websites, using resources from the same IP as a known DoH server, are accessible. A client / app may be attempting to use DoH, due to the exceptions this may work. The suricata rules provides protection, if the client uses standard (port 53) DNS queries to get the IP for the DoH server.

In order to add these rules to suricata (pfsense screenshots):

- Services / Suricata / Global Settings:
 - o Check "Download Extra Rules"
 - o Add the Extra Rules "Name" and "URL", use (check) "MD5", save (bottom of page)

Name: DOH

URL: <https://raw.githubusercontent.com/jpgpi250/piholemanual/master/DOH/DOH.rules>

Download Extra Rules

☒ Download Extra Rules
 Download extra rules file or tar.gz archive with rules. If "Check MD5" is set, the code will assume a matching filename additional extension of ".md5".

Extra rules

Rule

DOH

https://raw.githubusercontent.com/jpgpi250/piholemanual/master/DOH/DOH.rules

☒ Check MD5

Name

URL

- Services / Suricata / Updates:
 - o "Update your rule set", click the "Update" button.
 - o The result, after the download (MD5 checksum changes every day):

EXTRA RULE SET MD5 SIGNATURES		
Rule Set Name	MD5 Signature Hash	MD5 Signature Date
DOH	d7e3ef5e5861a567159ba3184f4c7066	Sunday, 13-Nov-22 11:17:11 CET

- Services / Suricata / Interface

The new rules need to be applied to the WAN interface, due to the “source” (\$HOME_NET), “target” (\$EXTERNAL_NET) and “flow” (to_server) definitions in the rules.

- Click on the “pencil” icon (Edit this Suricata interface mapping)
- Interface Tab “WAN Categories”, scroll to the bottom of this page and check “extrarule-DOH.rules”

Enabled

Extra Ruleset: DOH

☒
[extrarule-DOH.rules](#)

- Save, A message will appear (top of screen)

Suricata is 'live-loading' the new rule set on this interface.

- Verify the rules: Interface Tab / “WAN Rules”, select “extrarule-DOH.rules”, you should see something like this (screenshot).
- There are no “default disabled” rules, therefore, the rules should be active immediately. Be aware that “Enable All” can be wasteful of config.xml space, read [here](#) (explained by the experts).

WAN Settings
WAN Categories
WAN Rules
WAN Flow/Stream
WAN App Parsers
WAN Variables
WAN IP Rep

Available Rule Categories

Category
extrarule-DOH.rules
View All

Select the rule category to view and manage.

Rule Signature ID (SID) Enable/Disable Overrides

SID Actions
Apply
Reset All
Reset Current
Disable All
Enable All

When finished, click APPLY to save and send any SID state/action changes made on this tab to Suricata.

Rules View Filter

Rule Signature ID (SID) Enable/Disable Overrides

Legend:
✔ Default Enabled
✔ Enabled by user
✔ Auto-enabled by SID Mgmt
⚡ Action/content modified by SID Mgmt
⚠ Rule action is alert
⚠ Rule contains noalert option
✖ Default Disabled
✖ Disabled by user
✖ Auto-disabled by SID Mgmt
🔴 Rule action is drop
🚫 Rule action is reject











State	Action	GID	SID	Proto	Source	SPort	Destination	DPort	Message
✔	👤	1	27995001	dns	any	any	any	53	(o)DoH Query for dns.google
✔	👤	1	27995002	dns	any	any	any	53	(o)DoH Query for cloudflare-dns.com

You can also check “Active Rules” (WARNING: this takes a long time, press the “SID” header to change the order).

- Verify everything works, run (example) “dig dns9.quad9.net”

```
> <<>> DiG 9.16.33-Raspbian <<>> dns9.quad9.net
;; global options: +cmd
;; connection timed out; no servers could be reached
```

- Services / Suricata / Alerts

Last 250 Alert Entries. (Most recent entries are listed first)										
Note: Alerts triggered by DROP rules that resulted in dropped (blocked) packets are shown with highlighted rows below.										
Date	Action	Pri	Proto	Class	Src	SPort	Dst	DPort	GID:SID	Description
11/13/2022 16:05:56		2	UDP	Potentially Bad Traffic	  	45298	206.220.231.3   	53	1:27995003   	(o)DoH Query for dns9.quad9.net

12. Change Log.

16-06-2020

- Initial release.

11-07-2020

- Added list from Alphabet's [Intra app](#), ref [GitHub issue 4](#).

17-01-2021

- Entries from the existing (previous version) lists are only kept if the IP can be resolved, using the OpenDNS resolvers.

09-02-2021

- Removed list from heuristicsecurity.com (<https://heuristicsecurity.com/dohservers.txt>), this URL is redirected to a non-DoH related page.
- Added Lists from <https://github.com/dibdot/DoH-IP-blocklists>, ref [GitHub issue 8](#).

28-02-2021

- Added the database to the GitHub repository.

01-03-2021

- Added the 'latest_timestamp' property (info table).
- Added query example to use the 'latest_timestamp' value.
- Additional info for pi-hole users.

29-03-2021

- Added 'timestamp' field to 'urllist' table, this to identify unmaintained lists. A 'null' value indicates the list hasn't changed, since the list was added.
- Added query example to show human readable list change date.
- Increased database version (new field).

10-04-2021

- Increased database version (3).

20-09-2021

- Added List:
<https://raw.githubusercontent.com/crypt0rr/public-doh-servers/main/dns.list>.

01-10-2021

- Added section [Network control, using DNS entries](#).

21-12-2021

- Removed **0.0.0.0** and **::** entries (return value for dns.dnsoverhttps.net).
- Added automatic cleanup of database cnameinfo entries.

02-01-2022

- Added section [DoH response policy zone](#).
- Daily updated [DOH.rpz](#) on [Github](#).

02-08-2022

- Added new lists:
<https://raw.githubusercontent.com/xorguy/block-dot-doh/main/Lists/SDNS-Domains.list>
<https://raw.githubusercontent.com/jbaggs/doh-intel/master/doh.intel>
<https://raw.githubusercontent.com/mili-tan/ArashiDNS.Dekunua/main/DoH.list>

05-08-2022

- Database version 4, the exceptions table has been removed.
- The exceptions files will no longer be updated, it is the users responsibility to maintain the exception aliases. This is due to the ever growing list of exceptions, as a result of issues. The exception files are considered **deprecated**, but will remain on GitHub, to prevent problems with older implementations.
- Added instructions and examples to add and maintain user exception aliases.

07-08-2022

- Added new list (thanks, [poisonsnak](#)):
<https://ahadns.com/dns-over-https/>

This list can only be retrieved (if DoH protection is already in place) using:

```
# retrieve regional addresses below (A & AAAA) from https://dns-lookup.com/ahadns.com
curl --resolve *:443:104.26.2.137,2606:4700:20::681a:289 https://ahadns.com/dns-over-https/"
```

The list is added to ensure the ahadns.com entries will remain blocked, even if removed from other lists.

13-08-2022

- Added [Cloudflare CIDR](#) information.

24-09-2022

- Added section, explaining how to [limit the number of exeptions](#).

13-11-2022

- Added [DOH Suricata rules](#).
- Added warning Inline IPS Mode / Legacy Mode (ref. [comment\(s\)](#) from [bmeeks](#) – thanks for this, highly appreciated).

28-11-2022

- Added new list:
<https://adguard-dns.io/kb/general/dns-providers/>

This list can only be retrieved using:

```
lynx -dump "https://adguard-dns.io/kb/general/dns-providers/"
```

30-11-2022

- Added section, explaining how to add [CIDR \(network\) Exceptions](#).

14-12-2022

- Increased suricata sid allocation (27990000-27999999), due to the large amount of new entries, see the sid allocation table [here](#).