SNOWFLAKE AUDIT
-
SNAPSHOT

CAROLIN ZÖBELEIN

This paper is dedicated to all the brave snowflakes who die every year during winter.

Abstract. STATUS: DRAFT

Contents

Date: Last change: December 23, 2018, Status: Draft.

The author believes in the importance of the independence of research and is funded by the public community. If you also believe in this values, you can find ways for supporting the author's work here: `https://research.carolin-zoebelein.de/crowdfunding.html`.

Preamble

The following document is for discussion purposses only. It gives no warranty for completeness and correctness.

1. Introduction

Snowflake is a pluggable transport, which uses WebRTC to proxy traffic through emporary proxies. It aims to work kind of like flash proxy [1] [2]. This document is a short snapshot audit of the current state of the existing snowflake code [3] on `https://gitweb.torproject.org/pluggable-transports/snowflake.git/`.

2. Basic package structure

At first, let's have a first look at the basic package structure of Snowflake [3].

2.1. General package information.

Snowflake is a *Go*-based Pluggable Transport using WebRTC, inspired by Flashproxy.

2.2. Package tree.

Given are the following main parts of the package:

- **appengine (d)**: Runs an Google App Engine and reflects domain-fronted requests from a client to the Snowflake broker.
- **broker (d)**: Handles the rendezvous by matching Snowflake clients with proxies. It passes the client's WebRTC session descriptions. So the clients and proxies can establish a peer connection.
- **client (d)**: Tor client component of Snowflake.
- **proxy (d)**: Browser proxy component of Snowflake.
- **proxy-go (d)**: Standalone version of the Snowflake proxy.
- **server (d)**: Server transport plugin for Snowflake. The client connects to the proxy using WebRTC and the proxy connects to the server using WebSocket.
- **server-webrtc (d)**: WebRTC server plugin which uses an HTTP server that simulates the interaction that a client would have with the broker, for direct testing.
- **CONTRIBUTING.md (f), LICENSE (f), README.md (f):** Standard package files.

d= directory, f=file.

2.3. Interaction structure.

In figure 1 you can see the interaction structure between the different parts of Snowflake.

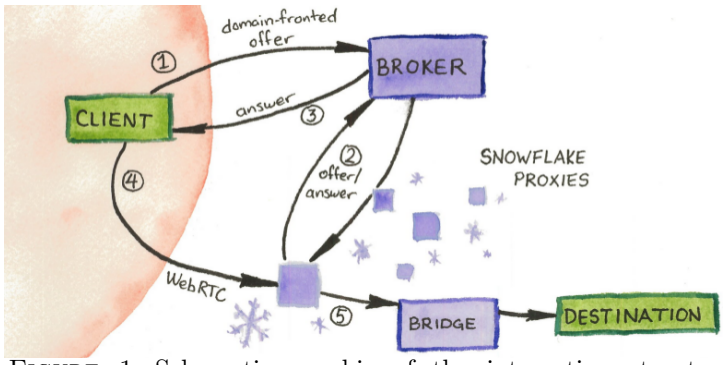

FIGURE 1. Schematic graphic of the interaction structure of Snowflake. Taken from [4].

1a. The client makes a request to the broker.
1b. The appengine reflects the domain-fronted request from the client to the broker.
2a. The broker matches the client with proxies.
2b. The broker passes the client's WebRTC session descriptions.
3. The broker gives an answer about the matching proxy to the client.
4. The client uses WebRTC for the connection to the given proxy.
5. Finally, the client connects to a bridge over WebRTC.

3. CLIENT

In the following sections we will do a raw audit of the Snowflake client.

3.1. README and torrc files.

Given are the *README.md*, *torrc*, *torrc-localhost* and *torrc-manual* files which contain similar content.

LISTING 1. torrc-Config: Standard values

```
1 UseBridges 1
2 DataDirectory datadir
3
4 Bridge snowflake 0.0.3.0:1
```

In listing 3.1 are defined the three standard values [5].

- **UseBridges 1**: Tor will fetch descriptors for bridges.
- **DataDirectory datadir**: Store working data in datadir. Can not be changed while tor is running.
- **Bridge snowflake 0.0.3.0:1**: When set along with UseBridges, instructs Tor to use the relay at "IP:ORPort" as a "bridge" relaying into the Tor network. If "transport" is provided, it must match a ClientTransportPlugin line. We then use that pluggable transports proxy to transfer data to the bridge, rather than connecting to the bridge directly.

The Snowflake client torrc default values are

LISTING 2. torrc-Config: Snowflake client default values

```
ClientTransportPlugin snowflake exec ./client \
−url https://snowflake−broker.azureedge.net/ \
−front ajax.aspnetcdn.com \
−ice stun:stun.l.google.com:19302
```

and can also be found in the client-*README.md* [6]

- **-url**: Should be the URL of a Broker instance. This is required to have automated signalling (which is desired in most use cases). When omitted, the client uses copy-paste signalling instead.
- **-front**: An optional front domain for the Broker request.
- **-ice** : A comma-separated list of ICE servers. These can be STUN or TURN servers.

. Additionally the following values can be set (see also *snowflake.go* [6]).

LISTING 3. torrc-Config: Snowflake client additional values

```
−max 3
−log snowflake.log
```

- **-max**: Capacity for number of multiplexed WebRTC peers.
- **-log**: Name of log file.

.

4. Conclusion

References

[1] Serene: *[tor-dev] Introducing Snowflake (webrtc pt)*. https://lists.torproject.org/pipermail/tor-dev/2016-January/010310.html. Version: 2016

[2] *Snowflake*. https://trac.torproject.org/projects/tor/wiki/doc/Snowflake. Version: 2018

[3] *Snowflake*. https://gitweb.torproject.org/pluggable-transports/snowflake.git/. Version: 2018/12/17

[4] *cypherpunks*. https://trac.torproject.org/projects/tor/attachment/wiki/doc/Snowflake/snowflake-schematic.png. Version: 2018

[5] Dingledine, Roger ; Mathewson, Nick: *Tor-stable Manual*. https://www.torproject.org/docs/tor-manual.html.en. Version: 2018

[6] *Snowflake - Client*. https://gitweb.torproject.org/pluggable-transports/snowflake.git/tree/client. Version: 2018/12/17

License

Carolin Zöbelein, Independent mathematical scientist, Josephsplatz 8, 90403 Nürnberg, Germany, https://research.carolin-zoebelein.de

E-mail address: contact@carolin-zoebelein.de, PGP: D4A7 35E8 D47F 801F 2CF6 2BA7 927A FD3C DE47 E13B