Network Working Group
Internet-Draft
Intended status: Informational
Expires: March 27, 2017

N. Mavrogiannopoulos
Red Hat

September 23, 2016

The OpenConnect VPN Protocol Version 1.0
draft-mavrogiannopoulos-openconnect-00

Abstract

This document specifies version 1.0 of the OpenConnect Virtual Private Network (VPN) protocol, a secure VPN protocol that provides communications privacy over the Internet. That protocol is believed to be compatible with CISCO's AnyConnect VPN protocol. The protocol allows the establishment of VPN tunnels in a way that is designed to prevent eavesdropping, tampering, or message forgery.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The purpose of this document is to specify the OpenConnect VPN protocol in a detail in order to allow for multiple interoperable implementations. This is the protocol used by the OpenConnect client and server [OPENCONNECT-CLIENT][OPENCONNECT-SERVER], and is believed to be compatible with CISCO's AnyConnect protocol.

While there are many competing VPN protocol solutions, none of them was ever described in a publicly available document. Even open source VPN solutions have their source code as the primary description of their protocol. That allowed no easy study of each protocol's properties and weaknesses, and that is the secondary goal of this document, to describe a deployed TLS based [RFC5246] VPN protocol.

1.1. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.2. Goals of This Document

The OpenConnect protocol version 1.0 specification is intended primarily for readers who will be implementing the protocol and those doing cryptographic analysis of it.

2. The OpenConnect Protocol

The OpenConnect protocol combines the TLS protocol [RFC5246], Datagram TLS protocol [RFC6347] and HTTP protocols [RFC2616] to provide an Internet-Layer VPN channel. The channel is designed to operate using UDP packets, and fallback on TCP if that's not possible.

In brief the protocol initiates an HTTP over TLS connection on a known port, where client authentication is performed. After this step, the client initiates an HTTP CONNECT command to establish a VPN channel over TCP. A secondary VPN channel over UDP will be established using information provided by the server using HTTP headers. At that point the raw IP packets flow, over the VPN channels.

2.1. VPN Session Establishment

The client and server establish a TLS connection over a known port, typically over 443, the port used for HTTPS. The client SHOULD negotiate TLS 1.1 or later, and support the following TLS protocol extensions.

Server Name Indication [RFC6066]: the client SHOULD provide the DNS name of the server in the TLS handshake.

Application-Layer Protocol Negotiation [RFC7301]: the client MAY provide this protocol name. The protocol name to be used is defined in Appendix A.

2.1.1. Server Authentication

In the OpenConnect VPN protocol, the server is always authenticated using its certificate. Once a client establishes a TCP connection to the server's well known port, it initiates the TLS protocol. In the first connection to the server, the client SHOULD verify the provided

by the server certificate, and SHOULD store its public key for verification of subsequent sessions. Thus, subsequent sessions SHOULD check whether the server's key match the initial.

The server's identity in the certificate SHOULD be placed in the certificate's SubjectAlternativeName field, and unless a special profile is assumed, it will be of type DNSName.

2.1.2. Client Authentication

The OpenConnect VPN protocol allows for the following types of client authentication, or combinations of them.

1. Password: a user can authenticate itself using a password.

2. Certificate: a user can authenticate itself using a PKIX certificate it possesses.

3. HTTP SPNEGO: a user can authenticate itself using a Kerberos ticket, or any other mechanism supported by SPNEGO (i.e., GSSAPI).

The server is authenticated to the client using a PKIX certificate presented during the TLS negotiation.

It is important to note that during the password and HTTP SPNEGO authentication methods, any headers allowed by the HTTP protocol can be present. In fact, there are legacy clients which assume that the server will keep a state using cookies, and send their username and password in different TLS and HTTP connections. This practice prevents the server from binding the TLS channel with the VPN session [RFC5056], and is discouraged. It is RECOMMENDED for clients to complete authentication in the same TLS session, and rely on TLS session resumption if reconnections to the server are needed.

After the TLS session is established the client irrespective of the supported authentication methods, should send an HTTP POST request on "/" with a config-auth XML structure of type 'init'. An example of its contents follow.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config-auth SYSTEM "config-auth.dtd">
<config-auth client="vpn" type="init">
    <version who="vpn">v5.01</version>
</config-auth>
```

The precise DTD declarations for the contents of XML messages defined in this document are listed in Appendix C. Also the HTTP Content-Type to be used for these XML structures MUST be 'text/xml'.

2.1.2.1. Authentication using certificates

During the initial TLS protocol handshake the server may require a client certificate to be presented, depending on its configuration.

Because the client certificate is sent in the clear during the handshake it SHOULD NOT contain other identifying information other than a username, or a pseudonymus identifier. It is RECOMMENDED to place the user identifier in the DN field of the certificate, using the UID object identifier (0.9.2342.19200300.100.1.1) [RFC4519].

After the TLS session is established and the the config-auth XML structure of type 'init' is sent, the server should send it reply. If the certificate sent by the client was successfully validated, it should reply using the HTTP response code 200, and the contents of the reply should be a config-auth XML structure of type 'complete', as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config-auth SYSTEM "config-auth.dtd">
<config-auth client="vpn" type="complete">
  <version who="sg">0.1(1)</version>
  <auth id="success">
    <title>SSL VPN Service</title>
  </auth>
</config-auth>
```

In that case the client should proceed to the establishment of the primary channel as in Section 2.1.4.

2.1.2.2. Authentication using passwords

After the TLS session is established and the the config-auth XML structure of type 'init' is sent, the server will reply using forms the client software should prompt the user to fill in. Its reply utilizes a config-auth XML structure of type 'auth-request'.

```
  <?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE auth SYSTEM "config-auth.dtd">
  <config-auth client="vpn" type="auth-request">
      <auth id="main">
          <message>Please enter your username</message>
          <form action="/auth" method="post">
              <input label="Username:" name="username" type="text" />
          </form>
      </auth>
  </config-auth>
```

The client may be asked to provide the information in separate forms as above, or may be asked combined as below.

```
   <?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE auth SYSTEM "config-auth.dtd">
   <config-auth client="vpn" type="auth-request">
     <auth id="main">
       <message>Please enter your username</message>
       <form action="/auth" method="post">
           <input label="Username:" name="username" type="text"/>
           <input label="Password:" name="password" type="password"/>
       </form>
     </auth>
   </config-auth>
```

The client software will then fill in the provided form and sent it back to the server using an HTTP POST on the location specified by the server (in the above examples it was "/auth"). The reply would then be of type 'auth-reply' as in the following example.

```
    <?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE config-auth SYSTEM "config-auth.dtd">
    <config-auth client="vpn" type="auth-reply">
        <version who="vpn">v5.01</version>
        <auth><username>test</username>
        </auth>
    </config-auth>
```

As mentioned above, the server may ask repeatedly for information until it believes the user is authenticated. For example, the server could present a second form asking for the password, after the username is provided, or ask for a second password if that is necessary. In these cases the server should respond with an HTTP 200 OK status code, and proceed sending its new request.

If client authentication fails, the server MUST respond with an HTTP 401 unauthorized status code. Otherwise, on successful

authentication the server should reply with a 200 HTTP code and use the 'complete' config-auth XML structure as in Section 2.1.2.1.

Note, that sending the username and password in different messages will reveal the length of them to a passive eavesdropper. For that is is RECOMMENDED for clients to use the 'X-Pad' HTTP header, which will contain arbitrary printable data to make the message length a multiple of 64 bytes.

An example session is shown in figure Figure 1.

```
        ,-.
        `-'
        /|\
         |                                ,------.          ,----------.
        / \                               |Server|          |ServerDTLS|
      Client                              `--+---'          `----+-----'
        |     TLS handshake Client Hello     |                   |
        | ----------------------------------->                   |
        |                                    |                   |
        |       TLS handshake Finished       |                   |
        | <-----------------------------------                   |
        |                                    |                   |
        |     HTTP POST config-auth init     |  ,--------------------!.
        | ----------------------------------->  |This is an HTTP over|_\
        |                                    |  |TLS session.          |
        |                                    |  `----------------------'
        |      config-auth auth-request      |                   |
        | <-----------------------------------                   |
        |                                    |                   |
        |  HTTP POST config-auth auth-reply  |                   |
        | ----------------------------------->                   |
        |                                    |                   |
        |        config-auth complete        |                   |
        | <-----------------------------------                   |
        |                                    |                   |
        |            HTTP CONNECT            |                   |
        | ----------------------------------->                   |
        |                                    |                   |
        |                                    |                   |
        |            ===================================         |
====================== CSTP VPN session is established =======================
        |            ===================================         |
        |                                    |                   |
        |                                    |  ,-------------------------!.
        | TLS record packet with CSTP payload|  |These packets show       |_\
        | ----------------------------------->  |that IP traffic can start  |
        |                                    |  |prior to the DTLS channel  |
```

```
        |                                    |  |establishment.             |
        |                                    |  `---------------------------'
        | TLS record packet with CSTP payload|                   |
        | <-----------------------------------                   |
        |                                    |                   |
        |               DTLS handshake Client Hello              |
        |  - - - - - - - - - - - - - - - - - - - - - - - - - - - >
        |                                    |                   |
        |                 DTLS handshake Finished                |
        | <- - - - - - - - - - - - - - - - - - - - - - - - - - - -
        |                                    |                   |
        |                                    |                   |
        |            ===================================         |
====================== DTLS VPN channel is established =======================
        |            ===================================         |
        |                                    |                   |
        |             DTLS record packet with payload            |
        |  - - - - - - - - - - - - - - - - - - - - - - - - - - - >
        |                                    |                   |
        |             DTLS record packet with payload            |
        | <- - - - - - - - - - - - - - - - - - - - - - - - - - - -
      Client                              ,--+---.          ,----+-----.
        ,-.                               |Server|          |ServerDTLS|
        `-'                               `------'          `----------'
        /|\
         |
        / \
```

Figure 1

2.1.2.3. HTTP Authentication using SPNEGO

That type of authentication is performed using the HTTP SPNEGO protocol [RFC4559], a method which is available using the Generic Security Service API [RFC2743]. The following approach is used to advertise the availability of the HTTP SPNEGO protocol by the client. A client which supports the HTTP SPNEGO protocol, SHOULD indicate it using the following header on in its initial request to the server with the config-auth 'init' XML structure.

```
    X-Support-HTTP-Auth: true
```

After that the server would report a "401 Unauthorized" status code and authentication would proceed as specified in the HTTP SPNEGO protocol. The server may utilize the following header, to indicate that alternative authentication methods are available (e.g., with plain password), if authentication fails.

```
X-Support-HTTP-Auth: fallback
```

If client authentication fails, the server MUST respond with an HTTP 401 unauthorized status code. In that case, a client which received the previous header should retry authenticating to the server without sending the "X-Support-HTTP-Auth: true" header.

Otherwise, on successful authentication the server should reply with a 200 HTTP code and use the 'complete' config-auth XML structure as in Section 2.1.2.1.

2.1.3. Exchange of Session Parameters

By the receipt of a success XML structure, the client SHOULD issue an HTTP CONNECT request. In addition it may provide the following headers.

X-CSTP-Address-Type: A comma separated list of the requested address types.

IPv4: when the client only supports IPv4 addresses.

IPv6: when the client only supports IPv6 addresses.

IPv4,IPv6: when the client supports both types of IP addresses.

X-CSTP-Base-MTU: The MTU of the link as estimated by the client.

X-CSTP-Accept-Encoding: A comma separated list of accepted compression algorithms for the CSTP channel.

User-Agent: A string identifying the client software.

For the options related to compression see Appendix B for more information.

An example CONNECT request is shown below.

```
User-Agent: Open AnyConnect VPN Agent v5.01
X-CSTP-Base-MTU: 1280
X-CSTP-Address-Type: IPv4,IPv6
CONNECT /CSCOSSLC/tunnel HTTP/1.1
```

After a successful receipt of an HTTP CONNECT request, the server should reply and provide the client with configuration parameters. The available options follow.

X-CSTP-Address: The IPv4 address of the client, if IPv4 has been requested.

X-CSTP-Netmask: An IPv4 netmask to be pushed to the client, if IPv4 has been requested. This should contain the mask on the P-t-P link and is RECOMMENDED the server address to be the first in defined network.

X-CSTP-Address-IP6: The IPv6 address of the client in CIDR notation, if IPv6 has been requested. The prefix length is RECOMMENDED to be set to 127-bits according to [RFC6164].

X-CSTP-DNS: The IP address of a DNS server that can be used for that session.

X-CSTP-Default-Domain: The DNS domains the provided DNS servers respond for.

X-CSTP-Split-Include: The network address of a route which is provided by this server.

X-CSTP-Split-Exclude: The network address of a route that is not provided by this server.

X-CSTP-Base-MTU: The MTU of the link as estimated by this server.

X-CSTP-DynDNS: Set to "true" if the server is operating with a dynamic DNS address.

X-CSTP-Content-Encoding: if present is it set to one of the values presented by the client in 'X-CSTP-Accept-Encoding' header. It will be the compression algorithm used in the CSTP channel.

X-DTLS-Content-Encoding: if present is it set to one of the values presented by the client in 'X-DTLS-Accept-Encoding' header. It will be the compression algorithm used in the DTLS channel.

The client is expected to treat the received parameters as his networking settings. If no "X-CSTP-Split-Include" headers are present, the client is expected to assign its default route through the VPN.

2.1.4. Establishment of Primary TCP Channel (CSTP)

The previous HTTP message is the last HTTP message sent by the server. After that message, the established TCP channel is used to transport IP packets between the client and the server. The

transferred packets encoding is discussed in Section 2.2. This channel will be referred as CSTP in the rest of this document.

2.1.5. Establishment of Secondary UDP Channel (DTLS)

To establish the secondary UDP-based channel, which will be referred to as the DTLS channel, the client must advertise support for it during the issue of the HTTP CONNECT request (see Section 2.1.3). This is done by appending the following headers to the request.

X-DTLS-Accept-Encoding: A comma separated list of accepted compression algorithms for the DTLS channel.

X-DTLS-CipherSuite: Must contain the keyword PSK-NEGOTIATE.

The DTLS channel utilizes the PSK key exchange method. The key material for this session is a 256-bit value generated with an [RFC5705] exporter. The key material exporter uses the label "EXPORTER-openconnect-psk" without the quotes, and without any context value.

In its client hello message the client must copy the value received in the 'X-DTLS-App-ID' header (after hex decoding it), to a TLS application-specific ID field [I-D.mavrogiannopoulos-app-id]. That identifier, can be used by the server to associate the client initiated DTLS channel with the CSTP channel. The following headers are used by the server's response to CONNECT, and are related to the DTLS channel establishment.

X-DTLS-App-ID: A hex encoded value to be used as a DTLS application-specific identifier by the client. It serves as an identifier for the server to associate the incoming DTLS session with the TLS session.

X-DTLS-Port: The port number to which the client should send UDP packets for DTLS.

X-DTLS-CipherSuite: It must contain the value "PSK-NEGOTIATE" without any quotes.

X-DTLS-Rekey-Time: The time (in seconds) after which the DTLS session should rekey, see Section 2.4. Only considered if applicable to the negotiated DTLS protocol.

X-DTLS-Rekey-Method: The method used in DTLS rekey, see Section 2.4. Only considered if applicable to the negotiated DTLS protocol.

2.1.5.1. Legacy Establishment of Secondary UDP Channel (DTLS)

Previous versions of this protocol utilized a special DTLS protocol negotiation, based on an unpublished description of the DTLS protocol. This section attempts to summarize this negotiation, but may not be entirely accurate.

To establish the legacy UDP-based channel, the client must advertise support for it during the issue of the HTTP CONNECT request (see Section 2.1.3). This is done by appending the following headers to the request.

X-DTLS-Accept-Encoding: A comma separated list of accepted compression algorithms for the DTLS channel.

X-DTLS-Master-Secret: A hex encoded pre-master secret to be used in the legacy DTLS session negotiation.

X-DTLS-CipherSuite: A colon-separated list of ciphers (e.g., the string PSK-NEGOTIATE:AES256-SHA:AES128-SHA:DES-CBC3-SHA).

The DTLS channel utilizes session resumption as a method for preshared-key authentication. That is the value presented in X-DTLS-Master-Secret is set as a master secret to be resumed. The session ID value is sent by the server on the response to CONNECT using the 'X-DTLS-Session-ID' header. That header provides a hex-encoded value of the DTLS session ID to be used by the client. The following headers are used by the server's response to CONNECT, and are related to the DTLS channel establishment.

X-DTLS-Session-ID: A hex encoded value to be used as a DTLS session ID by the client. It also serves as an identifier for the server to associate the incoming DTLS session with the TLS session.

X-DTLS-Port: The port number to which the client should send UDP packets for DTLS.

X-DTLS-CipherSuite: The ciphersuite selected by the server. It should be one of the options present in the client's X-DTLS-CipherSuite header.

X-DTLS-Rekey-Time: The time (in seconds) after which the DTLS session should rekey, see Section 2.4.

X-DTLS-Rekey-Method: The method used in DTLS rekey, see Section 2.4.

The following table lists the ciphers negotiated via the X-DTLS-CipherSuite header, and the corresponding DTLS ciphersuite.

OpenConnect cipher	DTLS ciphersuite	DTLS version
DES-CBC3-SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA1	DTLS 0.9 (pre-draft version)
AES128-SHA	TLS_RSA_WITH_AES_128_CBC_SHA1	DTLS 0.9 (pre-draft version)
AES256-SHA	TLS_RSA_WITH_AES_256_CBC_SHA1	DTLS 0.9 (pre-draft version)
OC-DTLS1_2-AES128-GCM	TLS_RSA_WITH_AES_128_GCM_SHA256	DTLS 1.2
OC-DTLS1_2-AES256-GCM	TLS_RSA_WITH_AES_256_GCM_SHA256	DTLS 1.2

Table 1

The legacy DTLS protocol negotiation described in this section, is similar to DTLS 1.0 except for the following deviations:

The negotiated protocol version for the handshake and record headers is 1.0 instead of 254.255.

The Hello Verify and Hello verify request messages are included in the handshake hashes.

The handshake header is not included as part of the handshake hashes.

The ChangeCipherSpec message is 3 byte long instead of 1, and contains the handshake sequence number (2-bytes long) appended to the message id.

2.2. The CSTP Channel Protocol

The format of the packets sent over the primary channel consists of an 8-bytes header followed by data. The whole packet in encapsulated in a TLS record (see [RFC5246]). The bytes of the header indicate the type of data that follow, and their contents are explained in Table 2.

byte	value
0	fixed to 0x53 (S)
1	fixed to 0x54 (T)
2	fixed to 0x46 (F)
3	fixed to 0x01
4-5	The length of the packet that follows this header in big endian order
6	The type of the payload that follows (see Table 3 for available types)
7	fixed to 0x00

Table 2

The available payload types are listed in Table 3.

Value	Description
0x00	DATA: the TLS record packet contains an IPv4 or IPv6 packet
0x03	DPD-REQ: used for dead peer detection. Once sent the peer should reply with a DPD-RESP packet, that has the same contents as the original request.
0x04	DPD-RESP: used as a response to a previously received DPD-REQ.
0x05	DISCONNECT: sent by the client (or server) to terminate the session. No data is associated with this request. The session will be invalidated after such request.
0x07	KEEPALIVE: sent by any peer. No data is associated with this request.
0x08	COMPRESSED DATA: a Data packet which is compressed prior to encryption.
0x09	TERMINATE: sent by the server to indicate that the server is shutting down. No data is associated with this request.

Table 3

2.3. The DTLS Channel Protocol

The format of the packets sent over the UDP channel consists of an 1-byte header followed by data. The header byte indicates the type of data that follow as in Table 3. The header and the data are encapsulated in a DTLS record packet (see [RFC6347]).

2.4. The Channel Re-Key Protocol

During the exchange of session parameters (Section 2.1.3), the server advertizes the methods available for session rekey using the "X-CSTP-Rekey-Method" and "X-DTLS-Rekey-Method" HTTP headers. The available options for both the server and client are listed below.

1. none: no rekey; the session will go on until 2^48 DTLS records have been exchanged, or 2^64 TLS records.

2. ssl: a TLS or DTLS rehandshake will be performed periodically.

3. new-tunnel: the session will tear down and the client will reconnect periodically.

When the value is other than "none" the rekey period is determinated by the "X-CSTP-Rekey-Time" and "X-DTLS-Rekey-Time" headers. These headers contain the time in seconds after which a session should rekey.

It should be noted that when the "ssl" rekey option is used, care must be taken by both the client and the server to ensure that either safe renegotiation is used ([RFC5746]), or that the identity of the peer remained the same.

2.5. The Keepalive and Dead Peer Detection Protocols

In OpenConnect there are two packet types that can be used for keep-alive or dead peer detection, as shown in Table 3. These are the DPD-REQ and KeepAlive packets.

The timings of the transmission of these packets are set by the server, and they for the DPD are advisory to a client. However, any peer receiving these packets MUST response with the appropriate packet. For DPD-REQ packets, the response MUST be DPD-RESP, and for KeepAlive packets the response must be another KeepAlive packet. The main difference between these two types of packets, is that the DPD packets similarly to [RFC3706] are sent when there is no traffic or when the other party requests them, and allow for arbitrary data to be attached, making them suitable for Path MTU detection.

The server advertizes the suggested periods during the exchange of session parameters (Section 2.1.3). The available headers are listed below.

X-CSTP-DPD: applicable to CSTP channel; contains a relative time in seconds.

X-CSTP-Keepalive: applicable to CSTP channel; contains a relative time in seconds.

X-DTLS-DPD: applicable to DTLS channel; contains a relative time in seconds.

X-DTLS-Keepalive: applicable to DTLS channel; contains a relative time in seconds.

3. Security Considerations

This document provides a description of a protocol to establish a VPN over a TLS channel. All security considerations of the referenced documents in particular [RFC5246] and [RFC6347] are applicable, in addition the following considerations.

The protocol is designed to be as compatible as possible with a legacy VPN protocol and as such it carries cruft, such as partial dependence on a non-standard DTLS version, and utilization of an awkward method to establish a DTLS session which relies on session resumption. Nevertheless, these particularities are not believed to cause a degradation of the overall protocol security, and could be addressed with a backwards compatible protocol upgrade.

The protocol provides a VPN channel which carries payload hidden from eavesdroppers. However, the payload's length remain visible and in certain scenarios that may be sufficient to determine the transferred payload. Furthermore, there are scenarios where compressed payload lengths may reveal more information than the uncompressed data [COMP-ISSUES][COMP-ISSUES2]. For that we RECOMMEND that implementations don't enable compression by default, and only allow it after notifying the users and administrators about the consequences.

This protocol could sometimes be used because of the fact that it ressembles the TLS protocol and thus is not detected by the available VPN blockers. While an implementation could intentionally masquerade its packets to ressemble a typical HTTPS session, a fully compliant implementation will be distinct from an average HTTP session due to the DTLS session establishment, and the transferred packet sizes.

For certificate authentication OpenConnect relies on the TLS protocol. However, as mentioned in the text, TLS version 1.2 and earlier do not protect the client's (or the server's) certificate from eavesdroppers. For that it is RECOMMENDED that certificates to be used with this protocol contain the minimum possible identifying information.

This document defines a protocol name for Application-Layer Protocol Negotiation. That, if used by a client would indicate to any eavesdropping parties that the client wishes to use VPN, thus compromising its intention privacy. On the other hand, providing that information would help a server that re-uses the same port for different protocols under TLS, to forward to the appropriate handler of the connection. That is, it would allow hosting a plain HTTPS server serving content, and a VPN server using openconnect at the

same port. It is left to the client to decide the balance between privacy and usability with such servers.

4. Acknowledgements

None yet.

5. Normative References

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5746] Rescorla, E., Ray, M., Dispensa, S., and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, DOI 10.17487/RFC5746, February 2010, <http://www.rfc-editor.org/info/rfc5746>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <http://www.rfc-editor.org/info/rfc2616>.

[RFC4559] Jaganathan, K., Zhu, L., and J. Brezak, "SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows", RFC 4559, DOI 10.17487/RFC4559, June 2006, <http://www.rfc-editor.org/info/rfc4559>.

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, DOI 10.17487/RFC2743, January 2000, <http://www.rfc-editor.org/info/rfc2743>.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, DOI 10.17487/RFC5056, November 2007, <http://www.rfc-editor.org/info/rfc5056>.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <http://www.rfc-editor.org/info/rfc5705>.

[RFC4519] Sciberras, A., Ed., "Lightweight Directory Access Protocol (LDAP): Schema for User Applications", RFC 4519, DOI 10.17487/RFC4519, June 2006, <http://www.rfc-editor.org/info/rfc4519>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <http://www.rfc-editor.org/info/rfc6066>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <http://www.rfc-editor.org/info/rfc7301>.

[RFC6164] Kohno, M., Nitzan, B., Bush, R., Matsuzaki, Y., Colitti, L., and T. Narten, "Using 127-Bit IPv6 Prefixes on Inter-Router Links", RFC 6164, DOI 10.17487/RFC6164, April 2011, <http://www.rfc-editor.org/info/rfc6164>.

[RFC3706] Huang, G., Beaulieu, S., and D. Rochefort, "A Traffic-Based Method of Detecting Dead Internet Key Exchange (IKE) Peers", RFC 3706, DOI 10.17487/RFC3706, February 2004, <http://www.rfc-editor.org/info/rfc3706>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[I-D.mavrogiannopoulos-app-id]
Mavrogiannopoulos, N. and D. Woodhouse, "A TLS application-specific identifier", draft-mavrogiannopoulos-app-id-00 (work in progress), September 2016.

[COMP-ISSUES]
Bhargavan, K., Fournet, C., Kohlweiss, M., Pironti, A., and P-Y. Strub, "TLS Compression Fingerprinting and a Privacy-aware API for TLS", 2012.

[COMP-ISSUES2]
Kelsey, J., "Compression and information leakage of plaintex", International Workshop on Fast Software Encryption , 2002.

[OPENCONNECT-CLIENT]
Woodhouse, D., "http://www.infradead.org/openconnect/", 2016.

[OPENCONNECT-SERVER]
Mavrogiannopoulos, N., "http://www.infradead.org/ocserv/", 2016.

Appendix A. Name for Application-Layer Protocol Negotiation

```
      Protocol:  openconnect-vpn/1.0
      Identification Sequence:
              0x6f 0x70 0x65 0x6e 0x63 0x6f 0x6e 0x6e 0x65 0x63
              0x74 0x2d 0x76 0x70 0x6e 0x2f 0x31 0x2e 0x30
```

Appendix B. Compression

The available compression algorithms for the CSTP and DTLS channels are shown in Table 4. Note, that all algorithms are intentionally stateless to prevent the influence of independent packets (e.g., from different sources) on each others compression. That does not eliminate all known attacks on compression before encryption, and for that reason an implentation MUST NOT enable compression by default.

After compression is negotiated each side may choose to compress the payload and use the 'COMPRESSED DATA' header from Table 3, or may send uncompressed data with the 'DATA' payload. Each side MUST be able to process both payloads.

Algorithm	Description
oc-lz4	The stateless LZ4 compression algorithm.
lzs	The stateless LZS (stacker) compression algorithm.

Table 4

Appendix C. DTD declarations

C.1. config-auth.dtd

```
<!ELEMENT config-auth (version*,auth*)>
  <!ATTLIST config-auth client CDATA #FIXED "vpn">
  <!ATTLIST config-auth type (init|auth-reply|auth-request|complete) "init">
<!ELEMENT version (#PCDATA)>
  <!ATTLIST version who (sg|vpn) "sg">
<!ELEMENT auth (title*,username*,password*,message*,form*)>
  <!ATTLIST auth id (success|main|failure) "failure">
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT username (#PCDATA)>
  <!ELEMENT password (#PCDATA)>
  <!ELEMENT message (#PCDATA)>
  <!ELEMENT form (input)>
    <!ATTLIST form action CDATA #FIXED "/auth">
    <!ATTLIST form method CDATA #FIXED "post">
    <!ELEMENT input (EMPTY)>
       <!ATTLIST input label CDATA "">
       <!ATTLIST input name (username|password) "username">
       <!ATTLIST input type (text|password) "text">
    <!ELEMENT select (option)>
       <!ATTLIST select label CDATA "">
       <!ATTLIST select name (group_list) "group_list">
    <!ELEMENT option (#PCDATA)>
```

Author's Address

Nikos Mavrogiannopoulos
Red Hat

EMail: nmav@redhat.com