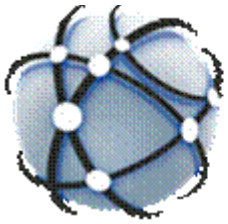

XRoot

ZTN Protocol Version 0

13-February-2021

Andrew Hanushevsky

SLAC
NATIONAL ACCELERATOR LABORATORY

1 Introduction

This document describes **XRootD**'s **ztn** security protocol. This application layer protocol allows a server to determine whether or not a client has the capability of obtaining authorization tokens from a trusted issuer. The protocol is only as successful as the authorization token is secure. Stolen tokens cannot be differentiated from ones actually obtained by the presenting client. The only guard is the expiry time in the token which should always be present and relatively short.

The general procedure is:

- The server is configured to use **ztn** as one of its authentication protocols.
- During the login sequence the client receives the list of protocols to try. Each protocol may have parameters associated with (as **ztn** does).
- The client goes down the list of protocols until it succeeds before declaring failure.

The **ztn** protocol succeeds when

- The client's connection is using **TLS**,
- the plug-in finds a token,
- sends the token to the server over the **TLS** connection, and
- the server implicitly accepts the token by not returning an error.

The **ztn** protocol should only use a **TLS** connection to send the token to the server. If the **ztn** plug-in finds that the connection is not using **TLS**, it should immediately fail and the client moves on to the next authentication protocol, if any.

2 ZTN Definitions

2.1 ZTN parameters

The authentication protocol and parameters are provided to the client during the login phase via the security token using a common format shown below.

```
ptoken: &P=protid[,protparms][ptoken]
```

Where:

protid 1- to 7-character protocol name. This name is typically used to locate a shared library that implements the security protocol.

protparms
optional protocol specific configuration parameters that should be supplied to the protocol's initialization routine when it is instantiated. The comma is required if *protparms* are present. The comma is optional otherwise. The *protparms* may not contain an ampersand (&).

The ztn specific *ptoken* entry would appear as

```
&P=ztn,flags:maxtsz:[tokloc[tokloc[…]]]
```

Where:

flags an ASCII text decimal encoding of an unsigned 64-bit integer value.

maxtsz
an ASCII text decimal encoding of a signed 32-bit integer value. The value represents the maximum length of the token, including the trailing null byte that the server will accept. Tokens longer than this value are rejected by the server.

tokloc an optional comma separated list of alternate token locations that the client should consider when doing token discovery, as directed by *flags*.

2.1.1 The *flags* definition

The *flags* should be treated as a structured 64-bit unsigned value as shown below.

```
flags: rr rr rr rr rr rr xx vv
```

Where

rr are reserved bits set to zero.

rr are flag bits set either to zero or one.

vv is an unsigned one byte version number of the protocol being used by the server.

xx bit settings

Value	Varname	meaning
0 1 2 3 4 5 6 7		
0x0000000000000100	**useFirst**	Apply *toklocs* first.
0x0000000000000200	**useLast**	Apply *toklocs* last.
0x0000000000000300	**useOnly**	Apply *toklocs* and only *toklocs*.
0x0000000000000800	**srvRTOK**	Allowed to ask for a new token.

useFirst, **useLast**, and **useOnly**

The **useFirst** and **useLast** bits indicate how the supplied *toklocs*, if any, are to be applied. Normally, the client uses the standard token discovery mechanism to find an applicable token to return to the server. The *toklocs* may be used by the server to augment that mechanism. When **useFirst** is set, the client should search as directed by *toklocs* prior to using the standard mechanism should the token not be found using the *toklocs*. When **useLast** is set, the client should use the *toklocs* if the standard mechanism failed to discover a usable token. Should **useFirst** and **useLast** be set (i.e. **useOnly**), then only the *toklocs* should be used to discover the token.

srvRTOK

The **srvRTOK** bit indicates that the server allows the client to obtain a token from an issuer in real time if the client cannot find a usable token. While the bit is defined, it is not currently supported until the token discovery mechanism defines specifies the way to obtain real-time tokens for background clients.

Notes

1) Clients embedded in a server can safely ignore the *tokloc* list as they have their own unique mechanism to locate or maintain a token.
2) Clients that do apply the supplied *tokloc* list should verify that the token being returned is an actual **JWT**. This is identified by the text being returned is in **JSON** format (i.e. minimally starts and ends with a brace (i.e. "{}") and contains a key of "**typ**" whose value is "**JWT**".

2.2 Sending a token to the server

The client should send a token to the server if it can be located using the following format:

```
  char      id[4];  ztn\0
  char      ver;    0
  char      opc;    'T'
  uint16_t  tlen;   Length of token in network byte
order
  char      token;  Actual token ending with null byte
```

Parameters

id The string "**ztn**" which is 4 characters long including the null bytes.

ver The version number of the protocol the client is using.

opc The operation code. The character '**T**' indicates that this is a token response.

tlen Is the length of the subsequent token, including the ending null byte. The length should be in network byte order.

token Is the actual token and should end with a null byte.

Notes

1) The server should reject the token if it exceeds the maximum length allowed or if it does not end with a null byte.

2.3 Requesting a list of valid issuers

To be written.

3 Document Change History

13 Apr 2021

- New Document.