Configuration of the D2Connector Process for SmartWeb

In this chapter, we will explain how to configure the connection on the D2000 application side for the Smartweb platform needs using examples. For communication between the SmartWeb application and D2000, the JAPI library is used which communicates with the D2Connector process on the D2000 side. To make use of this chapter, we need an active D2000 application (kernel at least) to which we can connect. Connection possibilities are divided into three categories and the possibilities from individual categories can be combined freely.

A connection can be established in two basic ways:

- Connection actively follows the SmartWeb server (basic way of connecting). It is a standard method, easier for usage and debugging and can be applied in every network in which the security policy allows it.
- Connection actively follows *D2Connector* (reverse connection). This method is used when the SmartWeb server is located in a so-called "demilitarized zone" (DMZ) - it is a computer network segment with which you can establish connection also from a local intranet and from outside internet but you cannot establish any connection from DMZ outside. (Supported from the 10.1.39 version)

From the point of view of the wiretapping protection, you can establish:

- Unsecured connection. Although the JAPI protocol is binary, all texts are transmitted in a readable way. An unsecured connection is good when debugging or in the case when commConfiguration of the D2Connector Process for SmartWebunication between *D2Connector* and *JConnector* r uns in a secure network.
- Connection secured by the TLS v1.2 protocol. *JAPI* and *D2Connector*communicate with each other by an encrypted protocol while *D2Connector* s hows its identity by a certificate and a private key which *JConnector* compares with its own certificate. (Supported from the 10.1.39 version)

From the point of view of connection to "hot" *kernel* in a redundant group:

- *D2Connector* always connects to the "hot" server
- *D2Connector* is always connected to the same *Kernel* regardless of whether it is "hot" or "standby server".

Parameters for Running *D2Connector*

D2Connector is a process of the D2000 system and it is distributed as a console application (`d2connector.exe`). It accepts standard parameters of D2000 processes for running from the command line which are described in the D2000 Online Reference Help. Besides, it accepts the following parameters of a command line:

- `--CONNECTOR_LISTEN_PORT=<port>` - it sets the number of the TCP port on which *D2Connector* listens for incoming connections from JAPI. If it is not stated differently, it will listen on the 3120 port. (The parameter will be ignored if the combination with `--DCC` is used)
- `--DCC=<hostname:port>` - it switches *D2Connector* from the listening mode to the mode of active connecting to the given address (DNS or IP) and to a port. It tries to establish a connection every 30 seconds until it succeeds. After ending the connection, it again tries to establish one.
- `--CONNECTOR_TLS_CERT=<path.crt>` - it turns on the TLS security and sets the path to the file with a certificate in the .crt format.
- `--CONNECTOR_TLS_PK=<path.pem>` - it turns on the TLS security and sets the path to the file with a private key to a certificate in the `.pem` form at. Both TLS parameters have to be used together.

D2Connector establishes connection only in one way from eight possible combinations. This means that it either actively connects or listens but not both at the same time. Similarly, it communicates either in unsecured or secured way but never in both ways at the same time. Either it is connected to one *Kernel* all the time or it switches to current "hot" one. In the case that more various client applications connect to the D2000 application and these client applications require different methods of connecting, it is necessary to run an individual instance of *D2Connector* for each method.

Basic Connection Method

It is an unsecured connection initiated by JAPI.

We can start *D2Connector* without parameters and it will listen to connection on the 3120 port

```
> d2connector.exe
```

or we can change the listening port to for example 3121:

```
> d2connector.exe --CONNECTOR_LISTEN_PORT=3121
```

Establishing a Reverse Connection

It is an unsecured connection between *D2Connector* and the SmartWeb application located in the DMZ from which it cannot initiate the TCP connection. However, it can listen to upcoming TCP connection which will be initiated by *D2Connector*.

The client application is located on the computer `portal.dmz.customer.com` and listens on the 3125 port on all of its network interfaces. We run *D2Con nector* in the mode of connecting:

```
> d2connector.exe --DCC=portal.dmz.customer.com:3125
```

Establishing a Secure Connection

It is a connection between *D2Connector* and *JConnector* secured by the TLS v1.2 protocol. The process is similar for standard and reverse connections. That is why the example encompasses both possibilities.

For establishing the TLS connection, it is necessary for one party to be in the role of the "TLS server" and the second party in the role of the "TLS client" while these roles are not dependent on who initiated the TCP connection. The "TLS server" shows a certificate to which it owns a personal key. The "TLS client" verifies the license validity and the key authenticity[5]. For JAPI, the "TLS server" is always *D2Connector* and the "TLS client" is always *JConnector*.

The requirement for creating a secure connection is that we have the RSA key pair and the X.509 certificate. The certificate is stored in the file in the `*.crt` format and both *D2Connector* and *JConnector* have to have access to its copies. The private key is stored in an unencrypted form in the file in the `*.pem` format and only *JConnector* has to have access to it. Since *JConnector* considers *D2Connector* trustworthy only based on showing the certificate it expected, the used certificate can be "self-signed" and it is not necessary to acquire the certificate from some certification authority.

> **WARNING:** From security reasons, it is **very** important for the file with a certificate to be stored in such a way that no one without given permission could change it (it can be public for reading). It is also **very** important that the file with a private key could be read only by *D2Connector* and no one could change it. In a case of breaking these conditions, there is a danger of compromising the certificate's trustworthiness and there is a possibility of wiretapping of secured communication.

It is necessary to run *D2Connector* with parameters `--CONNECTOR_TLS_CERT` and `--CONNECTOR_TLS_PK` which refer to files with a certificate and a private key. In the example, the certificate is stored in the file `certificate.crt` and the private key in the file `private.pem`. According to the needs, also parameters `--DCC` or `--CONNECTOR_LISTEN_PORT` can be used.

```
> d2connector.exe --CONNECTOR_TLS_CERT=certificate.crt --CONNECTOR_TLS_PK=private.pem
```

Creatinig a Certificate for Purposes of Secure Connection

For creating a "self-signed" certificate, it is possible to use for example the application OpenSSL from a command line. First, we have to create a key pair for the RSA code. In the example, we generate a 2048 bit key pair into the file `private.pem`.

```
> openssl.exe genrsa -out private.pem 2048
```

To the key pair, we create a "Certificate Signing Request" request for drawing certificate that would be stored in the file `request.csr`. OpenSSL queries more data which it writes into the request and which will be stated in the final certificate. However, JAPI does not examine or check this data.

```
> openssl.exe req -new -key private.pem -out request.csr
```

Then, we sign the request by a generated private key. This way, we create "selfsigned" certificate that will be located in the file `certificate.crt`. The certificate will be valid for 365 days starting with the current time.

```
> openssl.exe x509 -req -days 365 -in request.csr -signkey private.pem -out certificate.crt
```