2. Nadviazanie spojenia medzi klientskou aplikáciou a D2000

V tejto kapitole je na príkladoch vysvetlené, ako nadviazajednoduchos spojenie medzi klientskou aplikáciou a aplikáciou na platforme D2000. Predpokladom tejto kapitoly je aktívna D2000 aplikácia (prinajmenšom *kernel*), ku ktorej sa môžeme pripoji. Spôsoby pripojenia sú rozdelené do troch kategórií, priom možnosti z jednotlivých kategórií možno ubovone kombinova.

Pripojenie je možné nadviaza dvomi základnými spôsobmi:

- Spojenie aktívne nadväzuje klientská aplikácia (základný spôsob pripojenia). Je to štandardný postup, jednoduchší na použitie a ladenie a je aplikovatený všade tam, kde to bezpenostná politika v miestnej sieti dovouje.
- Spojenie aktívne nadväzuje *D2Connector* (reverzné pripojenie). Tento postup sa používa v prípade, že sa klientská aplikácia (napríklad webový server) nachádza v tzv. „demilitarizovanej zóne" (DMZ) – segmente poítaovej siete, do ktorej je možné nadviaza spojenie aj z lokálneho intranetu aj z vonkajšieho internetu, ale z DMZ von nie je možné nadviaza žiadne spojenie. (Podporované od verzie 10.1.39)

Z hadiska zabezpeenia JAPI protokolu pred odpoúvaním je možné nadviaza:

- Nezabezpeené spojenie. Hoci je JAPI protokol binárny, všetky texty sa prenášajú v itatenej podobe. Nezabezpeené spojenie je vhodné poas ladenia alebo v prípade, že komunikácia medzi *D2Connector*-om a *JConnector* om prebieha v bezpenej sieti.
- Spojenie zabezpeené protokolom TLS v1.2. *JConnector* a *D2Connector* komunikujú šifrovaným protokolom, priom *D2Connector* preukazuje svoju identitu certifikátom a privátnym kúom, ktorý *JConnector* porovná s certifikátom, ktorý vlastní on. (Podporované od verzie 10.1.39)

Z hadiska pripojenia k „hot" *kernel*-u v redundantnej skupine:

- *D2Connector* sa vždy pripája k „hot" serveru.
- *D2Connector* je stále pripojený k tomu istému *Kernel* u bez ohadu na to, i je „hot" alebo „stand-by server".

V obidvoch prípadoch je *JConnector* stále pripojený na ten istý *D2Connector*. Po prepnutí asociovaného *kernel*-a z „hot" na „SBS" alebo spä je klientská aplikácia notifikovaná o zmene, ale JAPI túto situáciu inak nerieši [1].

2.1. Parametre pre spustenie *D2Connector*-a

D2Connector je proces systému D2000 a je distribuovaný ako konzolová aplikácia (`d2connector.exe`). Akceptuje štandardné parametre procesov D2000 pre spustenie z príkazového riadku, ktoré sú popísané v Online referennej príruke systému D2000. Okrem toho akceptuje nasledovné parametre príkazového riadku:

- `--CONNECTOR_LISTEN_PORT=<port>` - nastaví íslo TCP portu, na ktorom *D2Connector* poúva na prichádzajúce spojenie od JAPI. Ak nie je uvedené inak, poúva na porte 3120. (Parameter je ignorovaný, ak sa použije v kombinácii s `--DCC`)
- `--DCC=<hostname:port>` - prepne *D2Connector* z režimu poúvania do režimu aktívneho pripájania sa na uvedenú adresu (DNS alebo IP) a port. Pokým sa mu nepodarí nadviaza spojenie, pokúša sa o to každých 30 sekúnd. Po ukonení spojenia sa opä zane pokúša o nadviazanie spojenia.
- `--CONNECTOR_TLS_CERT=<path.crt>` - zapne TLS zabezpeenie a nastaví cestu k súboru s certifikátom vo formáte .crt.
- `--CONNECTOR_TLS_PK=<path.pem>` - zapne TLS zabezpeenie a nastaví cestu k súboru s privátnym kúom k certifikátu vo formáte `.pem`. Obidva TLS parametre je potrebné použi spolu.

D2Connector nadväzuje spojenie vždy len jedným spôsobom z ôsmych možných kombinácií. Tzn. bu sa aktívne pripája, alebo poúva, ale nie obidvoje naraz. Rovnako komunikuje bu nezabezpeeným alebo zabezpeeným spôsobom, ale nikdy neumožuje obidva spôsoby súasne. Bu je pripojený stále k jednému *Kernel*-u alebo sa prepína na aktuálny „hot". V prípade, že sa ku D2000 aplikácií pripája viac rôznych klientských aplikácií, ktoré vyžadujú rôzne spôsoby pripojenia, je potrebné naštartova pre každý spôsob samostatnú inštanciu *D2Connector* a.

2.2. Základný spôsob pripojenia

Ide o nezabezpeené spojenie, ktoré iniciuje JAPI.

D2Connector môžeme naštartova bez parametrov a bude poúva na pripojenie na porte 3120

```
> d2connector.exe
```

alebo zmeni poúvajúci port napríklad na 3121:

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

Pripojenie sa s použitím JAPI v prípade, že sa *D2Connector* nachádza na poítai s menom `srvapp01v` a poúva na porte 3120. Inštancia *JConnector*-a bude na konci uložená v premennej `connector`.

```
String CONNECTION_STRING = "srvapp01v:3120";

D2ConnectorEventsListener connectorEventListener = new D2ConnectorEventsListener()
{
    @Override
    public void onClose(CloseReason reason)
    {
       //implementation of connector close event handling ...
    }
};
Future<D2Connector> upcomingConnection = D2Japi.getInstance().createConnector(CONNECTION_STRING,
connectorEventListener);
D2Connector connector = upcomingConnection.get();
```

2.3. Vytvorenie Session

Pripojený *JConnector* umožuje vytvori novú *Session* vždy rovnakým spôsobom bez ohadu na to, akým spôsobom bolo spojenie s *D2Connector*-om nadviazané. Nasledujúci postup je preto univerzálny pre všetky kombinácie. V príklade je *JConnector* reprezentovaný premennou .connector

Na pripojenie sa použije používateský úet SystemD2000 s heslom SystemD2000, proces bude v systéme identifikovaný menom MyD2Session.DCC[2] a prip ája sa z poítaa nb1tbac1[3].

```
String USER = "SystemD2000";
String PASSWORD = "SystemD2000";
String SESSION_NAME = "MyD2Session";
String HOST_NAME = "nb1tbac1";

D2SessionEventsListener sessionEventListener = new D2SessionEventsListener()
{
    @Override
    public void onClose(CloseReason reason)
    {
       //implementation
    }

    @Override
    public void onRedundancyStateChanged(RedundancyStateType redundancyState)
    {
        //implementation
    }

    @Override
    public void onTerminateRequest()
    {
        //implementation
    }
};
Future<D2Session> upcomingSession = connector.createSession(
new SessionParametersBuilder()
    .userName(USER)
    .password(PASSWORD)
    .sessionName(SESSION_NAME)
    .build(),
sessionEventListener);
D2Session session = upcomingSession.get();
```

2.4. Nadviazanie reverzného spojenia

Ide o nezabezpeené spojenie medzi *D2Connector*-om a *JConnector*-om, priom klientská aplikácia sa nachádza v DMZ, z ktorej nedokáže iniciova TCP spojenie. Môže však poúva na prichádzajúce TCP spojenie, ktoré bude iniciova *D2Connector*.

Klientská aplikácia je umiestnená na poítai `portal.dmz.customer.com` a poúva na porte 3125 na všetkých svojich sieových rozhraniach[4]. *D2connector* spustíme v režime pripájania sa:

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

Klientská aplikácia musí na získanie *JConnector* pripojeného reverzným spôsobom implementova rozhranie `sk.ipesoft.d2000.d2japi.ServerSocketEventsListener`. Rozhranie má 3 metódy, ktoré je potrebné implementova.

- `createEventsListener` – volá sa pri nadväzovaní TCP spojenia, ešte pred tým, ako sa pripájajúci identifikuje ako *D2Connector*. Klientská aplikácia má v tomto bode možnos odmietnu prichádzajúce spojenie (na základe adresy) alebo vytvori „listener-a" pre zachytávanie udalostí *JConnector*-a.
- `onConnectionEstabilished` – volá sa po úspešnom nadviazaní spojenia medzi *JConnector*om a *D2Connector*-om, parametrom je *JConnector* pripravený na použitie.
- `onConnectionFailed` – volá sa, ak sa spojenie nepodarilo nadviaza – pravdepodobne kvôli tomu, že sa o spojenie pokúsil neznámy proces, alebo *D2Connector* inej verzie.

> **Pozor:** Na poúvanie prichádzajúcich spojení bude vytvorené samostatné vlákno a všetky 3 metódy budú volané v tomto poúvajúcom vlákne. Je preto dôležité pamäta na potrebu synchronizácie prístupu k zdieaným zdrojom. Taktiež by vykonanie metód malo prebehnú poda možností o najrýchlejšie, aby neblokovalo spracovanie alších prichádzajúcich spojení.

```
ServerSocketEventsListener listener = new ServerSocketEventsListener()
{

    @Override
    public D2ConnectorEventsListener createEventsListener(InetAddress address, int port, ListeningHandle
handle) throws ConnectionRejectedException {
        if (acceptConnection(address, port)) {
            if (needConnectorListener(address, port))
                return new D2ConnectorEventsListener() { /* implementation */ };
            else
                return null; // simplest -> accept connection without any listener
        }
        else
            throw new ConnectionRejectedException(); // reject connection
    }

    @Override
    public void onConnectionEstabilished(D2Connector connector, InetAddress inetAddress, int port,
ListeningHandle handle)
    {
        // use connector to create sessions
    }

    @Override
    public void onConnectionFailed(ConnectorException ex, InetAddress inetAddress, int port, ListeningHandle
handle)
    {
        // handle false attempt - log it and dispose of created listener
    }
};
```

Následne je možné iniciova poúvanie na prichádzajúce spojenia. Návratová hodnota typu `sk.ipesoft.d2000.d2japi.ListeningHandle` uložená v premennej `handle` predstavuje poúvajúci socket, ktorý je možné v prípade potreby zatvori volaním metódy `close`.

```
String INTERFACE_NAME = ""; // all interfaces
int PORT_NUMBER = 3125;
ListeningHandle handle = D2Japi.getInstance().startListeningForConnection(INTERFACE_NAME, PORT_NUMBER,
listener);
```

2.5. Nadviazanie zabezpeeného spojenia

Ide o spojenie medzi *D2Connector*-om a *JConnector*-om zabezpeené protokolom TLS v1.2. Postup je podobný pre štandardné aj reverzné spojenie, príklad preto zaha obidve možnosti.

Pre nadviazanie TLS spojenia je potrebné, aby bol jeden z úastníkov v roli „TLS servera" a druhý „TLS klienta", priom tieto roly nie sú závislé na tom, kto inicioval TCP spojenie. „TLS server" sa preukazuje certifikátom, ku ktorému vlastní aj súkromný kú. „TLS klient" overuje platnos certifikátu a pravos kúov[5]. Pre JAPI je „TLS server" vždy *D2Connector* a „TLS klient" vždy *JConnector*.

Predpokladom pre vytvorenie zabezpeeného spojenia je, že máme RSA kúový pár a k nemu X.509 certifikát. Certifikát je uložený v súbore vo formáte `*.crt` a musí ma k jeho kópii prístup aj *D2Connector* aj *JConnector*. Privátny kú je uložený v nešifrovanej forme v súbore vo formáte `*.pem` a prístup k nemu musí ma iba *JConnector*. Vzhadom k tomu, že *JConnector* považuje *D2Connector* za dôveryhodný iba na základe toho, že sa preukázal rovnakým certifikátom, ako oakával, použitý certifikát môže by „self-signed" a nie je vôbec potrebné získa certifikát od nejakej certifikanej autority.

> **POZOR:** Z bezpenostného hadiska je **vemi** dôležité, aby bol súbor s certifikátom uložený tak, aby ho nikto bez príslušného oprávnenia nemohol zmeni. (Na ítanie môže by verejný.) Takisto je **vemi** dôležité, aby súbor so súkromným kúom mohol preíta iba *D2Connector* a nikto ho nemohol zmeni. V prípade porušenia týchto podmienok hrozí kompromitácia dôveryhodnosti certifikátu a otvára sa možnos odpoúvania zabezpeenej komunikácie.

D2Connector je potrebné spusti s parametrami `--CONNECTOR_TLS_CERT` a `--CONNECTOR_TLS_PK`, ktoré odkazujú na súbory s certifikátom a súkromným kúom. V príklade je certifikát uložený v súbore `certificate.crt` a súkromný kú v súbore `private.pem`. Poda potreby sa môže použi aj parameter `--DCC` alebo `--CONNECTOR_LISTEN_PORT`.

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

JConnector je možné získa obdobne ako bolo uvedené v kapitolách 3.2. a 3.4. avšak je potrebné použi preažené varianty metód `createConnector` a `startListeningForConnection` rozšírené o parameter `certificatePath`. Hodnota parametra musí obsahova cestu (relatívnu alebo absolútnu) k súboru `certificate.crt`.

Zjednodušený príklad nadviazania štandardného zabezpeeného spojenia:

```
String CERTIFICATE_PATH = "certificate.crt";
Future<D2Connector> upcomingConnection = D2Japi.getInstance().createConnector(CONNECTION_STRING,
CERTIFICATE_PATH, connectorEventListener);
```

Zjednodušený príklad nadviazania reverzného zabezpeeného spojenia:

```
ListeningHandle handle = D2Japi.getInstance().startListeningForConnection(INTERFACE_NAME, PORT_NUMBER,
CERTIFICATE_PATH, listener);
```

2.5.1. Vytvorenie certifikátu pre úely zabezpeeného spojenia

Pre vytvorenie „self-signed" certifikátu je možné použi napríklad aplikáciu OpenSSL z príkazového riadku. Najskôr musíme vytvori kúový pár pre RSA šifru. V príklade generujeme 2048 bitový kúový pár do súboru `private.pem`.

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

Ku kúovému páru vytvoríme „Certificate Signing Request" žiados o vystavenie certifikátu, ktorá bude uložená v súbore `request.csr`. OpenSSL sa opýta na viaceré údaje, ktoré zapíše do žiadosti a ktoré budú vo výslednom certifikáte uvedené. JAPI však tieto údaje neskúma a nekontroluje.

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

Následne podpíšeme žiados vygenerovaným súkromným kúom, ím z neho vytvoríme „selfsigned" certifikát, ktorý bude v súbore `certificate.crt`. Certifikát bude ma platnos 365 dní poínajúc aktuálnym asom.

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

2.6. Spojenie riadené triedou *RedundantConnectionManager*

Trieda *RedundantConnectionManager* zjednodušuje pripojenie a udržiavanie aktívneho pripojenia k redundantnej skupine[6]. Interným mechanizmom zabezpeí, že sa voi každému z *kernel*-ov vytvorí *JConnector* a v prípade straty spojenia sa pokúša o obnovu. Je možné ju využi aj na pripojenie k jedinému *kernel*-u. Inštanciu je možné získa volaním

```
connectionManager = D2Japi.getInstance().createRedundantConnectionManager(...)
```

---

[1] Pre jednoduchos nebudeme o redundancii *kernel*-a v nasledujúcich príkladoch uvažova.

[2] Kernel môže k názvu procesu prida dodatoné znaky, aby zabezpeil unikátnos identifikátora v systéme.

[3] Je to nepovinný údaj a kernel nedokáže overi jeho správnos, ale korektná aplikácia by túto informáciu mala poskytnú.

[4] V prípade, že si to bezpenostná politika poítaovej siete vyžaduje, je možné poúvanie obmedzi na jedno konkrétne sieové rozhranie.

[5] Existuje režim, kedy sa aj klient voi serveru preukazuje svojim vlastným certifikátom, túto možnos však JAPI nepodoruje.

[6] O redundantnej skupine hovoríme, ke D2000 aplikácia obsahuje aspo 2 *kernel*-y, z ktorých je vždy práve jeden riadiaci (*HOT*) a ostatné sú záložné (*SBS*) . Ak má riadiaci *kernel* poruchu, prevezme riadiacu funkciu automaticky jeden zo záložných.