

2. Nadviazanie spojenia medzi klientskou aplikáciou a D2000

- 2.1. Parametre pre spustenie *D2Connector-a*
- 2.2. Základný spôsob pripojenia
- 2.3. Vytvorenie Session
- 2.4. Nadviazanie reverzného spojenia
- 2.5. Nadviazanie zabezpečeného spojenia
 - 2.5.1. Vytvorenie certifikátu pre účely zabezpečeného spojenia
- 2.6. Spojenie riadené triedou *RedundantConnectionManager*

V tejto kapitole je na príkladoch vysvetlené, ako nadviaza jednoduchos spojenie medzi klientskou aplikáciou a aplikáciou na platforme D2000. Predpokladom tejto kapitoly je aktívna D2000 aplikácia (prínajmenšom *kernel*), ku ktorej sa môžeme pripojiť. Spôsoby pripojenia sú rozdelené do troch kategórií, pričom možnosti z jednotlivých kategórií možno ubovone kombinovať.

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 aplikovateľný všade tam, kde to bezpečnostná politika v miestnej sieti dovoľuje.
- 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 počítaťovej siete, do ktorej je možné nadviazať spojenie aj z lokálneho intranetu aj z vonkajšieho internetu, ale z DMZ von nie je možné nadviazať žiadne spojenie. (Podporované od verzie 10.1.39)

Z hľadiska zabezpečenia JAPI protokolu pred odpoúvaním je možné nadviazať:

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

Z hľadiska 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 ohľadu na to, či je „hot“ alebo „stand-by server“.

V oboch prípadoch je *JConnector* stále pripojený na ten istý *D2Connector*. Po prepnutí asociovaného *kernel-a* z „hot“ na „SBS“ alebo späť 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 príručke systému D2000. Okrem toho akceptuje nasledovné parametre príkazového riadku:

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

D2Connector nadväzuje spojenie vždy len jedným spôsobom z ôsmich možných kombinácií. Tzn. bu sa aktívne pripája, alebo počúva, ale nie oboje naraz. Rovnako komunikuje bu nezabezpečeným alebo zabezpečeným spôsobom, ale nikdy neumožňuje oboje spôsoby súčasne. Bu je pripojený stále k jednému *Kernel-u* alebo sa prepína na aktuálny „hot“. V prípade, že sa ku D2000 aplikácii pripája viac rôznych klientských aplikácií, ktoré vyžadujú rôzne spôsoby pripojenia, je potrebné naštartovať pre každý spôsob samostatnú inštanciu *D2Connector-a*.

2.2. Základný spôsob pripojenia

Ide o nezabezpečené spojenie, ktoré iniciuje JAPI.

D2Connector môžeme naštartovať bez parametrov a bude počúvať na pripojenie na porte 3120

```
> d2connector.exe
```

alebo zmení počúvajúci port napríklad na 3121:

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

Pripojenie sa s použitím JAPI v prípade, že sa *D2Connector* nachádza na počítači s menom *srvapp01v* a počúva 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* umožňuje vytvoriť novú *Session* vždy rovnakým spôsobom bez ohľadu 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 používateľský účet *SystemD2000* s heslom *SystemD2000*, proces bude v systéme identifikovaný menom *MyD2Session.DCC*² a pripája sa z počítača *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 nezabezpečené spojenie medzi *D2Connector*-om a *JConnector*-om, pričom klientská aplikácia sa nachádza v DMZ, z ktorej nedokáže iniciovať TCP spojenie. Môže však použiť na prichádzajúce TCP spojenie, ktoré bude iniciovať *D2Connector*.

Klientská aplikácia je umiestnená na počítači `portal.dmz.customer.com` a počúva na porte 3125 na všetkých svojich sieťových rozhraniach⁴. *D2connector* spustíme v režime pripájania sa:

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

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

- `createEventsListener` – volá sa pri nadviazovaní TCP spojenia, ešte pred tým, ako sa pripájajúci identifikuje ako *D2Connector*. Klientská aplikácia má v tomto bode možnosť odmietnuť prichádzajúce spojenie (na základe adresy) alebo vytvoriť „listener-a“ pre zachytávanie udalostí *JConnector*-a.
- `onConnectionEstablished` – 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 nadviazať – pravdepodobne kvôli tomu, že sa o spojenie pokúsil neznámy proces, alebo *D2Connector* inej verzie.



Pozor: Na používanie prichádzajúcich spojení bude vytvorené samostatné vlákno a všetky 3 metódy budú volané v tomto používajúcom vlákne. Je preto dôležité pamätať na potrebu synchronizácie prístupu k zdieľaným zdrojom. Taktiež by vykonanie metód malo prebehnúť podľa možnosti o najrýchlejšie, aby neblokovalo spracovanie ďalších 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 onConnectionEstablished(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 iniciovať používanie na prichádzajúce spojenia. Návrátová hodnota typu `sk.ipesoft.d2000.d2japi.ListeningHandle` uložená v premennej `handle` predstavuje používajúci socket, ktorý je možné v prípade potreby zatvoriť 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 zabezpečeného spojenia

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

Pre nadviazanie TLS spojenia je potrebné, aby bol jeden z účastníkov v roli „TLS servera“ a druhý „TLS klienta“, pričom tieto roly nie sú závislé na tom, kto inicioval TCP spojenie. „TLS server“ sa preukazuje certifikátom, ku ktorému vlastní aj súkromný kľúč. „TLS klient“ overuje platnosť certifikátu a právomokov⁵. Pre JAPI je „TLS server“ vždy *D2Connector* a „TLS klient“ vždy *JConnector*.

Predpokladom pre vytvorenie zabezpečeného spojenia je, že máme RSA kľúčový pár a k nemu X.509 certifikát. Certifikát je uložený v súbore vo formáte *.crt a musí mať k jeho kópii prístup aj *D2Connector* aj *JConnector*. Prívätý kľúč je uložený v nešifrovanej forme v súbore vo formáte *.pem a prístup k nemu musí mať iba *JConnector*. Vzhľadom k tomu, že *JConnector* považuje *D2Connector* za dôveryhodný iba na základe toho, že sa preukázal rovnakým certifikátom, ako ošetroval, použitý certifikát môže byť „self-signed“ a nie je vôbec potrebné získať certifikát od nejakej certifikačnej autority.



POZOR: Z bezpečnostného hľadiska je **veľmi** dôležité, aby bol súbor s certifikátom uložený tak, aby ho nikto bez príslušného oprávnenia nemohol zmeniť. (Na íťanie môže byť verejný.) Takisto je **veľmi** dôležité, aby súbor so súkromným kľúčom mohol prejsť iba *D2Connector* a nikto ho nemohol zmeniť. V prípade porušenia týchto podmienok hrozí kompromitácia dôveryhodnosti certifikátu a otvára sa možnosť odpočúvania zabezpečenej komunikácie.

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

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

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

Zjednodušený príklad nadviazania štandardného zabezpečeného spojenia:

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

Zjednodušený príklad nadviazania reverzného zabezpečeného spojenia:

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

2.5.1. Vytvorenie certifikátu pre účely zabezpečeného spojenia

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

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

Ku kľúčovému páru vytvoríme „Certificate Signing Request“ žiadosť 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 žiadosť vygenerovaným súkromným kľúčom, čím z neho vytvoríme „selfsigned“ certifikát, ktorý bude v súbore `certificate.crt`. Certifikát bude mať platnosť 365 dní počnúc aktuálnym dňom.

```
> 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 skupine⁶. Interným mechanizmom zabezpečí, že sa voči každému z *kernel*-ov vytvorí *JConnector* a v prípade straty spojenia sa pokúša o obnovu. Je možné ju využiť aj na pripojenie k jedinému *kernel*-u. Inštanciu je možné získať volaním

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

¹ Pre jednoduchosť nebudeme o redundancii *kernel*-a v nasledujúcich príkladoch uvažovať.

² Kernel môže k názvu procesu pridať dodatočné znaky, aby zabezpečil unikátnosť identifikátora v systéme.

³ Je to nepovinný údaj a kernel nedokáže overiť jeho správnosť, ale korektná aplikácia by túto informáciu mala poskytnúť.

⁴ V prípade, že si to bezpečnostná politika počítačovej siete vyžaduje, je možné používanie obmedziť na jedno konkrétne sieťové rozhranie.

⁵ Existuje režim, kedy sa aj klient voči serveru preukazuje svojím vlastným certifikátom, túto možnosť však JAPI nepodporuje.

⁶ O redundantnej skupine hovoríme, keď D2000 aplikácia obsahuje aspoň 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.