Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati rendszerek és szolgáltatások Tanszék

Identitás információk kezelése biztonsági események kiértékelésében

Diplomaterv

Készítette
Bulla Ádám

Konzulens
dr. Czap László
dr. Buttyán Levente

2017. november 21.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Bulla Ádám*, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2017. november 21.

Bulla Ádám
hallgató

1. fejezet

Bevezetés

1.1. A dolgozat célja és felépítése

A modern informatika egyik fontos és feltörekvő területe az IT Security, amely a számítógép ipar fejlődésével egyre nagyobb szerepet kap. Ahogy a gépek számító kapacitása növekszik, egyre könnyebben megoldhatók olyan problémák, amelyek addig lehetetlennek, elfogadható időben kivitelezhetetlennek tűntek. Ez a fejlődés az egész szektort arra készteti, hogy folyamatosan fejlődjön, a meglévő alkalmazásokat, módszereket, algoritmusokat javítsa. Emellett a modern világban egyre nagyobb vállalatok jönnek létre, amelyeknek egyre nagyobb személyzetre van szükségük a működéshez, ami indokolttá teszi egy megfelelően stabil és jól kezelhető informatikai támogató réteg kialakítását. Több ezer, akár több tízezer alkalmazott mellett gyorsan átláthatatlanná válik, hogy kinek milyen eszközökhöz, akár hardveres, akár szoftvereshez van hozzáférése, ezek egyesével való beállítása és karbantartása pedig emberi léptékkel mérve szinte kivitelezhetetlen, és rendkívül költséges a fent említett támogató szoftverek nélkül.

Jelen dolgozat az IT Security világának számos területéből kettővel foglalkozik, ennek a kettőnek is elsősorban a kapcsolatával. Az egyik terület a Security Information and Event Management (SIEM), ami egy informatikai rendszer biztonsági felügyeletével foglalkozik. A másik terület az Identity management (IdM), ami pedig az alkalmazottak és a hozzájuk tartozó jogosultságok életciklusának menedzselésével foglalkozik. A cél a kettő terület összekapcsolása oly módon, hogy az IdM szoftverben található hasznos, felhasználókkal kapcsolatos adatok elérhetők legyenek a SIEM szoftver számára. Ezek olyan kontextust szolgáltatnak, amely az elemi eseményekből nem következik. Például a SIEM által feldolgozott események jellemzően köthetők egy felhasználónévhez, viszont nem állnak rendelkezésre azok az információk, hogy az adott felhasználónév mely valós személyhez tartozik és az illető esetleg kiléptetés alatt áll-e, a biztonsági házirenddel összhangban van-e egy adott fiók létezése és jogosultsági szintje, vagy hogy mikor és ki hagyta jóvá a felhasználói fiók létrehozását. Mindezen adatok az IdM rendszerből kinyerhetők. Az integráció célja, hogy az IdM rendszerben tárolt releváns adatokkal tudjuk támogatni a SIEM szabályrendszerét.

Egy ilyen integrációval az alábbiak, valamint ehhez hasonló use-case-ek valósíthatók meg:

- Inaktív személyekhez tartozó felhasználói fiókok - Ez az információ hasznos lehet egy QRadar szabályhoz például olyan esetben, ha arra vagyunk kíváncsiak, hogy volt-e aktivitás olyan fiókkal, amelynek a tulajdonosa már nem a cég alkalmazottja, és a fióknak meg kellett volna szűnnie.

- Valós személyhez nem köthető felhasználói fiókok - Ezek az árva fiókok biztonsági kockázatokat jelenthetnek, mert a hozzájuk tartozó műveletekért nincs kit felelős-

ségre vonni. Ezért hasznos lehet egy olyan QRadar szabály, ami kifejezetten az ilyen esetekben jelez.

- Adott erőforráshoz legitim hozzáféréssel rendelkező személyek - Ez az információ olyan esetben lehet hasznos, ha például egy támadás kiinduló pontjaként sikerül azonosítani egy eszközt. Ezzel lehetséges az olyan felhasználói fiókkal történő aktivitás észlelése, amelyet az IdM szabályrendszerét megkerülve hoztak létre.

A diplomatervem keretében fejlesztettem egy integrációs modult, amely egy általános célú adatszinkronizációs eszköz az IdM és a SIEM rendszer között, vamint ennek segítségével megvalósítottam a fent leírt use-case-ekhez szükséges lekérdezéseket és adatszinkronizációt.

A SIEM számára nemcsak a felhasználókkal kapcsolatos adatok, hanem a jogosultságkezeléssel kapcsolatos folyamatok eseményei is relevánsak, amelyeknek szintén az IdM rendszer a forrása. A diplomatervem során megvalósíottam egy olyan eszközt, amely bővíti a SIEM számára látható IdM események körét, ezzel teljesebbé téve az IdM rendszer biztonsági monitorozását.

A dolgozat felépítése a következő:

- Az 1. fejezet a dolgozat valamint a diplomaterv célját definiálja és járja körbe, rövid bemutatót adva a felhasznált technológiák főbb tulajdonságairól.
- A 2. fejezet a projektben megismert és felhasznált technológiákat mutatja be, kitérve a feladat számára fontos technológiákra.
- A 3. fejezet a konkrét implementációs kérdéseket mutatja be.
- A 4. fejezet egy rövid összefoglaló a féléves munkámról.

1.2. Security Information and Event Management

A Security Information and Event Management az informatikai rendszer részeinek monitorozásával foglalkozik biztonsági szempontból. Az infrastruktúrában található eszközök által generált eseményekhez hozzáfér a SIEM megoldás, és ez végzi az események feldolgozását és elemzését. A monitorozott rendszerek működésükkel kapcsolatos információkat biztosítanak a SIEM irányába valamilyen formában, általában log sorokként. A SIEM szerver ezeket feldolgozza, és a szabályrendszerének megfelelő eseményekből úgynevezett biztonsági incidenseket hoz létre, akár egyéb forrásokból érkező plusz informácók felhasználásával. Ilyen egyéb forrás lehet hálózati forgalom, valamilyen adatbázisból lekért adatok, vagy előre definiált, a szerverre feltöltött adatok. Nem triviális feladat a szabályrendszert úgy konfigurálni, hogy a fals pozitív riasztások száma alacsony maradjon, miközben a valós támadásokat hatékonyan detektálja.

A megvalósítandó integráció a szabályrendszer hatékonyságát kétféle képpen segíti elő. Egyrészt, az IdM-ben redelkezésre álló információk segítségével olyan támadásminták detekciója válik lehetővé, amelyeket enélkül a SIEM nem lenne képes észlelni. Másrészt, a SIEM szabályrendszerét aktuális adatokkal látja el, amely a fals pozitív riasztások számát képes csökkenteni.

Emellett az általam generált és feldolgozott IDM-ből érkező jogosultsági folyamatok eseményein keresztül a SIEM további, fontos incidenseket képes detektálni.

Jelen feladatnak nem célja a SIEM szabályrendszerének részletes kidolgozása, a dolgozat témája az IdM és a SIEM közötti adatszinkronizáció lehetőségének megteremtése.

1.3. Identity management

Az Identity management a fejlődő nagyválllatok fent említett problémáiból a nagyszámú alkalmazott hozzáféréseinek kezelését és a dolgozók, mint informatikai entitások életciklusának menedzselését oldja meg. Ez magában foglalja az entitások rendszerezését, csoportokhoz rendelését, valamint a saját és örökölt jogaik érvényre juttatását.

Minden alkalmazotthoz tartozik egy rekord, amely leírja az adott ember személyes adatait és egyéb olyan információkat, amelyek szükségesek az alkalmazott jogosultságainak meghatározásához. Ezt a létrejött entitást beosztja a megadott információk szerint a megfelelő, előre definiált szerepkörökbe, amely alapján az jogokat kap bizonyos eszközök használatára. Ezen eszközök is entitásként vannak felvéve a rendszerbe, oly módon, hogy elérhetők az eszközhöz (akár szoftveres akár hardveres) tartozó információk és menedzselhető a hozzáférés.

A dolgozat célja ezen alkalmazotti és a hozzájuk kapcsolódó életciklus információinak kinyerése és eljuttatása a SIEM rendszer számára, mivel ezek az információk értékesek lehetnek a biztonsági incidensek kiértékelése szempontjából.

1.4. A feladat specifikálása

Jelen dolgozat feladata egy olyan megoldás fejlesztése, mely lehetővé teszi a fent említett technológiák közti integrációt és az adatszinkronizációt. Az integrációt a IBM által kínált IdM (IBM Security Identity Manager - ISIM[1]) és SIEM (IBM Security QRadar SIEM - QRadar[2]) között dolgoztam ki.

A végső megoldás két részből áll, egyrészt egy teljesen új funkcionalitást biztosító integrációs modulból, valamint egy már meglévő funkcionalitást bővítő kiegészítésből. Erre a feladatra eddig nem volt automatikus megoldás, ezért ez a projekt ezt a hiányt hivatott betölteni.

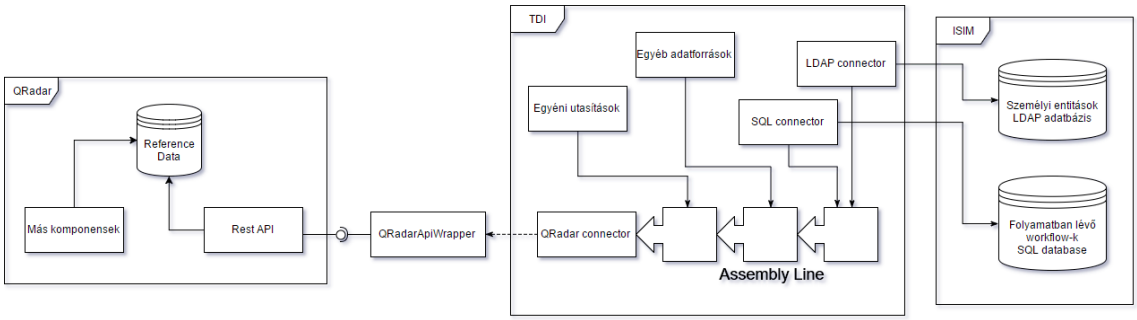
A QRadarban már megtalálható egy olyan modul, ami képes az ISIM-ben generált események egy részhalmazának feldolgozására és értelmezésére, de ez csak bizonyos audit események feldolgozására képes. Ezt kibővítendő készítettem egy megoldást, amely az előző funkcionalitást egészíti ki egyéb, eddig nem feldolgozott eseményekkel, amelyek plusz információt hordozhatnak biztonsági szempontból. A modulhoz használt keretrendszert a TDI [3] nyújtja.

Az integráció megvalósításához a Java alapú technológiát választottuk. A döntést indokolja, hogy jól illeszkedik a tipikus nagyvállalati környezethez, az IBM kiterjedt tapasztalattal és eszközkészlettel rendelkezik ezen a téren, valamint ez az ISIM technológiája és programozói interfésze is.

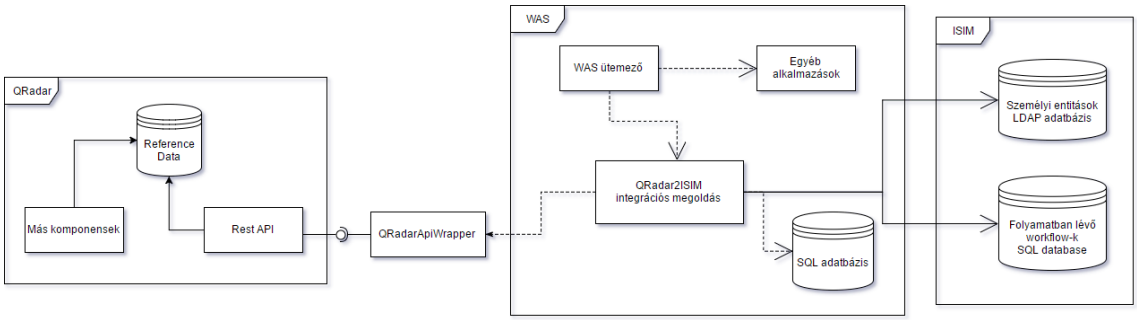
Az integráció megvalósításának első lépése egy Java API fejlesztése a QRadar-hoz. A QRadar egy technológiafüggetlen REST API-val rendelkezik. Ahhoz, hogy Java alkalmazásból az API számunkra fontos része használható legyen, egy Java Wrapper library-t fejlesztettem, amely Java metódusok formájában teszi lehetővé a QRadar API használatát. Ez a wrapper egy általános megoldást ad a QRadarba feltöltött adatok (lsd. 2.1.1) lekérésére és módosítására, így egy később felmerülő projektben is hasznos lehet.

Az integráció megvalósítására két architektúrát is kidolgoztunk, amelyeket a következő két alfejezetben ismertetek.

[1] Lásd 2.1.2
[2] Lásd 2.1.1
[3] Lásd 2.1.3 Tivoli Directory Integrator - TDI



1.1. ábra. Architektúra TDI esetén



1.2. ábra. Architektúra WAS esetén

1.4.1. Integráció TDI segítségével

A Tivoli Directory Integrator (TDI [3]) egy gyakran használt, általános célú integrációs eszköz. Ehhez készítettem egy connector-t, amely a fent említett wrapper könyvtár segítségével képes adatokat feltölteni és lekérni a QRadartól. A hálózati erőforrások takarékos használatának céljából a connector külön akkumulálja a feltöltendő adatokat, és a futása végén, egyben tölti azokat fel. Ez a connector később szabadon újra felhasználható nem csak az ISIM-mel, hanem bármely egyéb forrás integrációjára is.

Ebben a megvalósításban a QRadar és az ISIM közti integráció TDI assembly line-ok formájában jön létre, melyeket a TDI által biztosított szerver tud feldolgozni és futtatni. A TDI gyári funkcióit és grafikus interfészét felhasználva az egyes assembly line-ok fejlesztése időtakarékos és költséghatékony. Ennek a megvalósításnak hátránya, hogy a megoldások a TDI-hez kötöttek, valamint a lehetőségeknek határt szabnak a TDI által nyújtott lehetőségek.

1.4.2. WAS alkalmazás formájában

Ennél a megoldásnál egy IBM WebSphere (WAS) alkalmazást tervezünk elkészíteni, amely a QRadarral való kommunikációra felhasználja az általam fejlesztett wrapper-t. Az alkalmazás rendelkezik egy felhasználóbarát webes felülettel, melyen keresztül létrehozhatunk és felkonfigurálhatunk szinkronizációs feladatokat. Az integrációt ilyen szinkronizációs feladatok valósítják meg, melyek ütemezett futtatására támogatást biztosít az alkalmazás a WAS által nyújtott lehetőségeken keresztül. A feladatok eltárolják az aktuálisan lekérdezett adatokat egy SQL adatbázisba, valamint karbantartanak egy másik táblát ami mindig a QRadarra aktuálisan sikeresen felszinkronizált adatokat tartja számon. Ezen adatbázisok segítségével számolható egy különbség, ami az elégségesen felküldendő adatokat tartalmazza. Ezzel csökkenthető a QRadar irányába a tranzakciónkénti overhead. Emellett ha az alkalmazás inkonzisztenciát érzékel a lokális állapot és a QRadarban megtalálható adatok

[3] reffoot:TDI

között, akkor egy teljes szinkronizációval minden adatot felküld, ezzel egy új, konzisztens állapotba álltva a rendszert.

Ebben a megvalósításban TDI alapú megoldáshoz képest előny, hogy az integrációs adatokhoz tartozó szinkronizáció pontos implementációja saját fejlesztésű, így hozzáigazítható a pontos igényekhez. Emellett a WAS egy menedzselt környezetet biztosít a futtatáshoz, így jobban felügyelhetők az egyes feladatok, valamint igény szerint könnyen megvalósítható az elosztott működés is.

A TDI-t használó architektúrával szemben a hátránya, hogy a megoldás ISIM specifikus, más forrásokkal való integrációhoz szükség van a forráskód módosítására.

2. fejezet

Irodalomkutatás és a felhasznált technológiák

2.1. A felhasznált technológiák ismertetése

Mivel a feladat egy specifikus alkalmazás előállítása volt, amely már létező termékek közötti kommunikációt biztosít, ezért ennek jelentős része volt a termékekkel való alapszintű, valamint a felhasznált specifikus funkciókkal és interfészekkel való mélyebb ismerkedés.

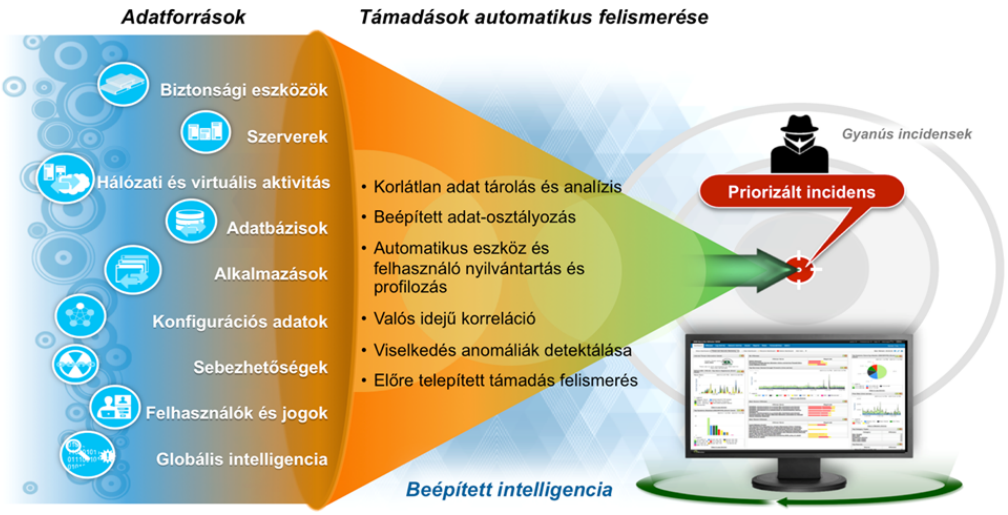
2.1.1. IBM Security QRadar SIEM

Az IBM Security QRadar SIEM az IBM security information and event manager rendszere, ami lehetővé teszi hálózatra csatlakoztatott eszközöknek a megfigyelését biztonsági szempontból. A hálózaton elosztott több ezernyi eszközvégpontból és alkalmazásból származó napló fájl eseményadatait összesíti, és a nyers adatokon azonnali normalizálási és összesítési műveleteket végez. Az eseménynaplók betöltésére számtalan automatikus módszer áll rendelkezésre, többek közt olyan közismert protokollok mint a SYSLOG, SNMP, FTP, SCP. Az IBM Security QRadar SIEM ugyancsak képes a rendszer sebezhetőségeinek és az esemény- és hálózati adatoknak az összevetésére, ezáltal segítséget nyújt a biztonsági incidensek rangsorolásában. Emellett lehetőség van egyéb adatforrások felvételére a felhasználó által is, amelyek szintén használhatók a fenyegetések és az incidensek detektálásában. Ezek jelentősége elsősorban a dinamikus szabályok létrehozásában játszik nagy szerepet, mivel ezek segítségével egy API-n keresztül karbantarthatók a létrehozott dinamikus szabályok. A szinkronizáció megvalósítására nincs egységes módszer vagy eszköz, ezt minden esetben az adatok jellege és az adatforrás által biztosított interfész határozza meg.

Felvehetők a SIEM-be bizonyos sablonok alapján összeállítható szabályok, amelyeket a rule engine kiértékel a beérkező eseményekre. A kiértékelés alapján az eseményeket besorolja a megfelelő csoportokba súlyosságuk és egyéb tulajdonságaik alapján, vagy ha szükséges létrehoz egy új, különálló eseményt. Az incidensek kezelésére külön felület szolgál, illetve különböző interfészeken keresztül értesítést tud küldeni ezekről a rendszer. Ezen túl minden feldolgozott esemény később megtekinthető keresések és szűrések segítségével.

A SIEM által kiértékelt eseményekhez egyéb információkat is rendel a rendszer, olyanokat, mint például a támadás típusa, az esemény leírása, a résztvevő felek adatai, melyeket később is meg lehet tekinteni, valamint segítségükkel és az egyéb környezeti forgalommal együtt egy egész hálózat működése visszajátszható.

Ezen dokumentum és a feladat szempontjából a legfontosabb része a QRadarnak a dinamikusan feltölthető adathalmazok és azok használata szabályokban. Ezekkel a szabályokkal érhető el, hogy más adatforrásokból (jelen esetben az ISIM-ből) frissen feltöltött



2.1. ábra. QRadar funkciói.

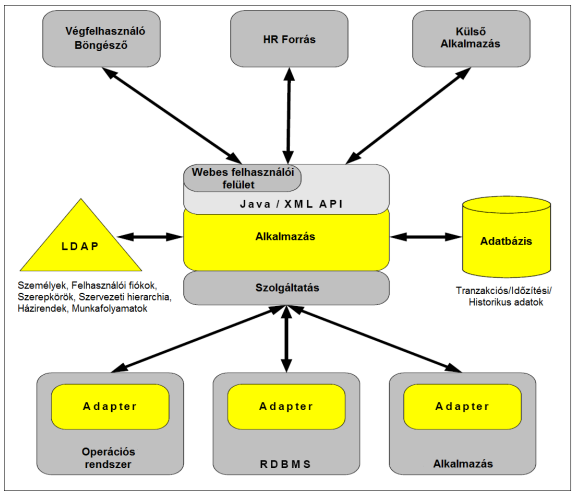
információk alapján változzon a kiértékelés, és ha valamilyen adat frissül, akkor naprakész maradjon a szabály által talált incidensek halmaza. A dinamikusan feltölthető adathalmazok (összefoglaló nevükön reference data) elérhetők egy REST API-n keresztül, így könnyen hozzájuk lehet férni és módosítani őket. Négy féle ilyen adathalmaz áll rendelkezésre:

- Reference set - Olyan adathalmaz, melyben egyedi értékek sorozata található.
- Reference map - Olyan adathalmaz, melyben kulcs-érték párok találhatók, a kulcsok egyediek, és szigorúan szöveges adatok.
- Reference map of sets - Olyan adathalmaz, melyben kulcs-halmaz párok találhatók, a kulcsok egyediek, szövegesek, és a halmazban saját csoportjukban egyedi értékek találhatók.
- Reference map of maps (tables) - Olyan adathalmaz, melyben kulcs-kulcs-érték triplet összerendelések találhatók.

Minden reference data-nak van egy típusa, ami meghatározza hogy az adott halmazban milyen típusú értékek találhatók.

- ALN - Alfanumerikus karakterek
- ALNIC - Alfanumerikus karakterek, figyelmen kívül hagyva a kis- és nagybetű közti különbséget
- IP - IP címek
- NUM - Numerikus karakterek
- PORT - Port számok
- DATE - Dátumok, miliszekundomokban 1970.01.01 óta

A feladat megvalósítása során az ISIM-ből kinyert adatokat ilyen reference data-kba töltjük fel, a típust úgy változtatva, ahogy az indokolt a kinyert adat szempontjából. A dolgozat nem foglalkozik a már feltöltött adatok további felhasználásával valamint a dinamikus szabályrendszer használatával, pusztán az integráció megvalósítására koncentrál.



2.2. ábra. Az ISIM architektúrája és interfészei.

2.1.2. IBM Security Identity Manager - ISIM

Az IBM Security Identity Manager alapú IDM megoldás elsődleges feladata érzékelni a személyügyi változásokat, és egy központi szabálymotor alapján gondoskodni arról, hogy az alkalmazottak azokkal és csak azokkal a jogosultságokkal rendelkezzenek, amelyek mindenkori munkakörük beteljesítéséhez szükségesek.

Az ISIM tartja karban a kapcsolatot a vállalatnak dolgozó személyek és e személyek IT hozzáférései, jogosultságai között, gondoskodva mind a személyekben, mind a fiókokban bekövetkezett változások az aktuális biztonsági házirend alapján történő szinkronizálásáról. Ennek megfelelően két fő folyamatot definiál a rendszer. Egyrészt a HR forrásokban, tehát a személyek adataiban bekövetkező változások hatásait kell érvénybe léptetni. Másrészt szükséges a menedzselt rendszeren (service) bekövetkezett, IDM-en kívül eszközölt módosítások detektálása, és azok átvezetése vagy korrigálása a belső szabályrendszernek megfelelően.

Az ISIM ezen információk tárolására két külső adattárolót használ: egy LDAP alapú címtárban tárolja a modell entitásokat, azaz a személyek és fiókok adatait, rendszeradatokat, munkafolyamat és házirend definíciókat. Emellett használ egy relációs adatbázist a tranzakciós adatok, azaz a workflow példányok futási kontextusa (például aktív jog igénylések), audit bejegyzések, ideiglenes szimulációs és ütemezési adatok tárolására. Az integrációs modul használata folyamán ebből a két adattárolóból nyerjük ki az adott use-case-hez szükséges információkat.

2.1.3. Tivoli Directory Integrator - TDI

A Tivoli Directory Integrator egy általános célú integrációs eszköz, ami lehetővé teszi több, különböző adatforrás koordinálását és integrációját. Mivel a legtöbb forrás más formátumot használ, és máshogy tárolja az adatot, egy ilyen integrációs lépés során szükséges bizonyos átalakításokat elvégezni az adatokon, valamint lehetséges hogy egyéb, plusz lépéseket is szükséges bevezetni, akár más adatforrások bevonásával. Ennek a procedúrának ad keretet a TDI egy grafikus fejlesztő felülettel, valamint a megfelelő Java alapú interfészekkel és kötésekkel, amelyek könnyűvé teszik új komponensek fejlesztését.

A TDI alapvető struktúrája úgynevezett assembly line-okból áll. Egy assembly line jelképez egy adat transzfert, a kezdeti adatok felolvasásától az átalakításokon át, a végső kimenet feltöltéséig. A ki- és bemeneti interakció ún. connectorokon keresztül történik, amelyek egy egységes interfészt implementálnak, és valamilyen külső adatforráshoz való

kapcsolódást valósítanak meg. Mivel a legtöbb adatforrás más formátumban tárolja az adatokat, a TDI minden be- valamint kimeneti műveletnél biztosít egy hozzárendelési lépést, amellyel megadhatjuk, hogy a külső attribútumok milyen belső attribútumokra legyenek leképezve. Ilyen ún. mapping lépést az assembly line-on bármikor végrehajthatunk, és emellett még számtalan átalakítási lépés áll rendelkezésre, mint például ciklusok vagy elágazások használata. A TDI talán egyik legfontosabb képessége a Javascriptből való testreszabhatóság. Ez azt jelenti, hogy az assembly line-on az adatokat szabadon manipulálhatjuk Javascriptes kódból, létrehozhatunk szkripteket amik a futtatás bizonyos pontjain aktiválódnak, valamint számtalan egyéb funkciót érhetünk el ezekből a programokból, mint például a logolás, paraméterek módosítása, vagy arbitrális kód futtatása.

A dolgozat szempontjából az egyik legfontosabb része a TDI-nak a connectorok, mivel a feladat része volt egy ilyen fejelsztése, ami támogatja a kommunikációt egy QRadar szerverrel, azon belül is a QRadarban található reference data objektumokkal.

3. fejezet

Feladat megvalósítása

Mint már fentebb említésre került, a dolgozat témája részben egy valós, határidős projekt volt, így ennek megfelelően egy csapat dolgozott rajta. Ezen belül én is részfeladatokat kaptam és implementáltam, valamint részt vettem a tervezési procedúrában.

3.1. Wrapper fejlesztése QRadar-hoz

A projekt első kihívása egy Java alapú wrapper fejlesztése volt a QRadar reference data manipulációt kezelő webes REST apijához. Későbbiekben ezen a wrapperen keresztül bonyolítunk majd minden forgalmat az átláthatóbb kód készítése céljából, ezért fontos hogy a wrapper megvalósítson minden olyan funkciót amire szükség lehet.

A fejlesztés első lépéseként tanulmányoztam a REST Api-hoz tartozó referencia dokumentációt, ami leírja mely endpointokon milyen HTTP kérések hajthatók végre, milyen paraméterekkel, milyen választ adhat és milyen státusz üzeneteket kaphatunk. Ebből az anyagból kiderült, hogy a négy reference data típushoz 4 endpoint halmaz tartozik, amelyek hasonló felépítéssel és paraméterezéssel bírnak. Egy ilyen endpoint halmazra mutat példát az alábbi felsorolás.

- /sets - GET, és POST műveletet támogat. A POST-tal új reference set hozható létre, a GET metódussal pedig lekérhető a rendelkezésre álló setek listája.

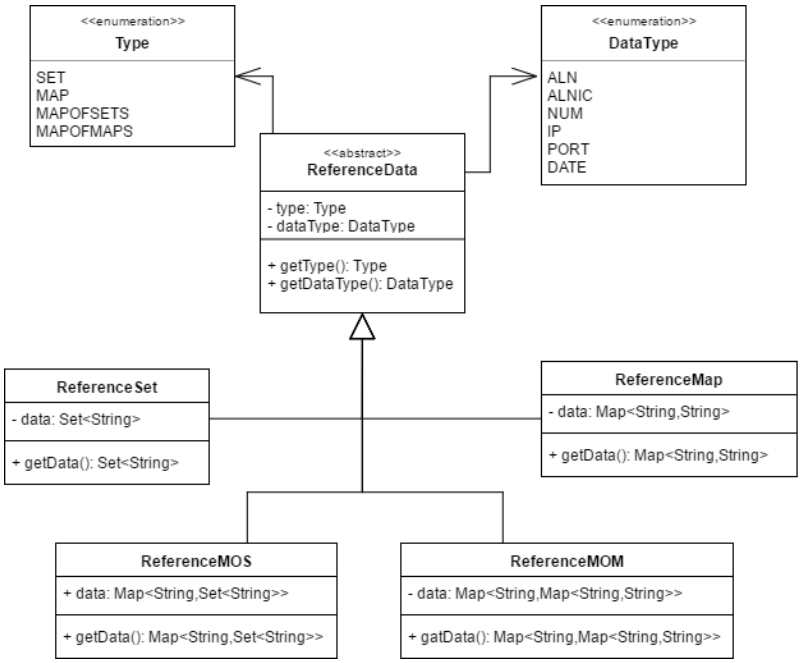
  /{name} - GET, POST, DELETE. Az URL-ben megadott paramétert a QRadar a reference set neveként értelmezi, és ezen keresztül érhető el a set lekérése (GET), teljes törlése (DELETE), valamint egy elemi adat feltöltése (POST).

  /{value} - DELETE. Ennek az endpointnak a segítségével tudunk egy bizonyos értéket törölni a reference set-ből.

  /bulk_load/{name} - POST. Az egyik legfontosabb endpoint, mivel ezen keresztül tudunk feltölteni egy olyan JSON formátumú szöveget, amellyel egyszerre több értéket is tudunk állítani egy reference set-ben (vagy más endpointok esetén más reference data-kban).

A fent felsorolt endpointok közül mindegyiket implementáltam a wrapperben, Java konvención alapuló neveket adva a függvényeknek. Egy függvény egy működést valósít meg, és ez a működés a reference data típusának szempontjából transzparens, tehát nem szükséges külön metódust hívni egy reference set és egy reference map feltöltéséhez, hanem elég egy metódust, más paraméterekkel.

A reference data-kkal való könnyebb interakció miatt definiáltunk egy saját adatszerkezetet egy Java osztály formájában, a data típusokkal megegyező néven. Mindegyik osztály egy ReferenceData navű absztrakt ősosztályból származik, ami egy egységes interfacet biztosít a leíró adatok, mint például a típus, az adatok típusa, lekéréséhez. Ennek a



3.1. ábra. A ReferenceData osztály és leszármazottainak felépítése

ReferenceData-nak a leszármazottai a konkrét reference típusokat megvalósító osztályok. Mindegyik osztály rendelkezik egy, a saját maga által reprezentált struktúrának megfelelő tárolóval, amely tárolja az adott reference data adatait. Mivel az integráció során többnyire szöveges, vagy azzá könnyen átalakítható adatokkal dolgozunk, és a QRadar irányába is JSON formátumban továbbítjuk az adatokat, így kézenfekvő mindet szövegként tárolni. A tárolókhoz használt kollekciók pontos típusa, valamint az implementációhoz használt architektúra leolvasható a mellékelt ábráról 3.1.

Lehetséges lenne más formátumban tárolni az adatokat, mint például egy Java alapú JSON reprezentációban, JSONObject-ben, vagy akár egy hosszú karakterláncként is, ám ezzel elvesztenénk a Java beépített kollekciói által nyújtott funkciókat, mint például az iterációt, vagy a tartalmazás ellenőrzését. Ezek mind nélkülözhetetlen funkciók a könnyű fejlesztés érdekében, valamint a megfelelő teljesítmény biztosítása szempontjából is fontosak, amire később látunk majd példát.

A wrapper fejlesztése közben külön kihívást jelentett a QRadar REST api-val való kommunikáció megvalósítása. A QRadar ugyanis csak HTTPS forgalmat fogad el, TLS segítségével, ezért egy, a QRadar által generált tanusítványt kellett hozzáadni minden olyan környezethez, amely a wrappert használta. Ez a két külön architektúra esetén a WebSphere és a TDI tanusítvány könyvtárát jelentette, és ez egy olyan követelmény, ami a wrapper későbbi használata esetén is szükséges. Emellett a QRadar megköveteli, hogy a REST API-jához csatlakozó kliensek használjanak egy, a QRadar által előre generált tokent, amit minden híváskor fel kell küldeniük. A wrapper osztály ezt konstruktorában kéri, és automatikusan minden kérésnél elküldi. A TLS kapcsolat biztosítja a szerver hitelességét, míg a token a szerver számára hitelsíti az API-t használó klienst, így összeségében a kommunikáció kölcsönösen hitelesített.

Magának a HTTP forgalomnak és a REST hívásoknak a lebonyolítására az Apache Wink[1] framework-öt használtam. Ez egy egyszerű Java alapú framework, melynek része egy JAX-RS kompatiblis szerver, és egy kifejezetten REST hívások lebonyolítására kiélezett HTTP kliens. A projektben a kliens komponens RestClient osztályát használtam, va-

[1] `https://wink.apache.org`

lamint a frameworkkel együtt érkező JSON4J csomagot, a JSON inputok parse-olására és a szöveges outputok generálására. A fejlesztés közben külön kihívást jelentett a JSON4J csomag megfelelő osztályainak használata, valamint egymásba ágyazása. Ez a csomag ugyanis két osztályt bocsájt rendelkezésre a JSONObject valamint a JSONArray formájában. Az object osztály reprezentálja a map típusú, míg az array a tömb típusú struktúrákat. Ez annyiban nehezítette a fejlesztést, hogy a különböző ReferenceData leszármazottak közt nem lehetett egységes parseolást használni, hanem a többszörösen egymásba ágyazott kollekciók esetén több lépcsős iterációt kellett használni a JSON felépítéséhez. Ez túl nagy adathalmazoknál lassabb működést eredményezhet.

A wrapper a különböző reference data-k transzparens kezelésén túl egyéb funkciókat is ellát, mint például segéd funkciók biztosítása, vagy a hibák egységes kezelése. Ilyen segéd funkciók a különböző adattranszformációk a használt típusok és a JSON formátum között, vagy például különböző ellenőrzések egy reference data létezésére, vagy egy aszinkron törlés lefutására. A hibakezelés menedzselésére a wrapper egy saját kivétel osztályt definiál, ami minden, a sikeres lefutástól eltérő esetben (akár belső hiba, akár a QRadarral való kommunikáció közben fellépő hiba) eldobásra kerül. Ez az objektum tartalmaz egy szöveges üzenetet, ami a hiba okára utal, valamint egy státusz kódot, ami ha HTTP kommunikáció közbeni hiba történt, annak a kódját tartalmazza, ha belső működésbeli hiba (például parse-olási hiba) akkor egy 0-nál kisebb számot tartalmaz. Ez egységesen és könnyen használhatóvá teszi a felsőbb rétegek számára, ahol például logolást kezeljük, mert egyértelmű, hogy a hiba milyen forrásból adódott. A saját kivétel típus pedig tovább könnyíti a hibák elválasztását, főleg ha a wrapper-t egy nagyobb framework-ben használjuk.

3.2. TDI Integráció megvalósítása

A TDI alapú megoldás célja egy olyan keret, és hozzá tartozó proof of concept use case-ek kidolgozása, amely lehetőséget ad az ISIM és QRadar közti integrációs feladatok gyors, hatékony és egyszerű megvalósítására. Mivel a TDI által nyújtott keretrendszerbe illeszkedik a megoldás, így képes használni az általa nyújtott számos lehetőséget, például a már létező connector-okat sokféle rendszerhez, valamint a use case implementációk alapján később könnyedén készíthetők új megoldások olyanok által, akik járatosak a TDI használatában.

A megvalósításhoz szükség volt elsősorban egy connector létrehozására, majd ezt felhasználva implementáltam néhány tényleges megoldást, általunk, valamint az ügyfél által definiált esetekre.

3.2.1. QRadar connector fejlesztése TD-Ihoz

A projekt következő lépése a connector fejlesztése volt a TDI és a QRadar közti integrációhoz. Ehhez segítségemre volt a hivatalos útmutató connector fejlesztéshez, ami az IBM oldalán megtalálható. Egy saját connector fejlesztése TDI alatt abból áll, hogy létrehozunk egy új Java osztályt ami implementálja a megfelelő, TDI által specifikált interfészt, ennek metódusain belül elkészítjük a kívánt üzleti logikát, majd egy xml leíró fájllal együtt, a megfelelő struktúrában becsomagoljuk azt egy JAR fájlba, amit a TDI által használt könyvtárak egyikébe másolunk.

Alapvetően hat működési módja lehet egy connector-nak, amit a feljebb említett dokumentum specifikál. Ezek a következők:

- Iterator - Végigiterál az adatforrás elemein, azokat felolvassa, és az assembly line rendelkezésére bocsájtja.
- AddOnly - Az assembly line-on érkező adatokat hozzáadja az adatforráshoz.

- Lookup - Lehetőséget ad több adatforrás elemeinek illesztésére. Úgynevezett link criteria megadásával az erre illeszkedő elemeket választja ki.
- Delete - Az assembly line-ról kapott összes elem esetén a megadott link criteria-t felhasználva megpróbálja megkeresni az elemet, és ha megtalálta, törli azt. Lehetőség van olyan felparaméterezésre is, amely egyszerre több elemet töröl.
- Update - Már meglévő adatok módosítását végzi. Előbb megpróbálja megkeresni a link criteria-val megadott rekordokat, ha talált ilyen elemet, akkor összehasonlítja az assembly line-on érkező elemmel, és elvégzi a szükséges módosításokat. Ha nem talált megfelelő elemet, akkor hozzáadja újként.
- Delta - Egy különleges mód, melyhez szükség van további, ilyen módot támogató elemekre az assembly line-on. A felolvasott adatokat összehasonlítja egy külső tárolóban (ún. delta store) tárolt elemekkel, és a két elem differenciáiból delta műveleteket képez, amiket végrehajt a célrendszeren. Lényegében eltárolja a kezelt elemek legutóbb használt verzióját, és csak az újonnan beolvasott elemekhez képesti különbséget hajtja végre.

Ezek a módok meghatározzák, hogy a fejlesztendő connector-nak milyen metódusokat muszáj implementálnia a megfelelő működéshez. Például az AddOnly mód csak az Initialize, putEntry, és a terminate metódusokat használja, így ha a connector csak ezt akarja használni, elég ezeket implementálni. Ezzel szemben például az Update mód ezeken felül használja a findEntry, és a modEntry metódust is. Minden connector a saját logikáját definiálja, amivel megvalósítja az adott célrendszeren az absztrakt módon megfogalmazott műveletet. Egy JDBC connector például adatbázis parancsokkal valósítja meg a fent definiált műveleteket egy kapcsolaton keresztül. Jelen esetben egy REST API-n keresztül elérhető a kommunikáció a célrendszerrel, így a connector szabványos HTTP kéréseket használ. A fejlesztés során megvalósításra került az összes fent említett mód.

Első lépésként tehát elkészítettem egy QRadarReferenceDataConnector osztályt, ami örököl egy generikus ősosztályból, ami már előre megvalósít olyan metódusokat a TDI által használt ConnectorInterface-en, amelyek nem kifejezetten connector specifikusak, hanem generikus feladatokat látnak el, például konfigurációs fájlok beolvasása vagy paraméterek beállítása.

Az implementációs döntések megértéséhez fontos ismerni egy connector életciklusát. A connector létrejöttekor meghívódik a konstruktora, de ez a dokumentációban leírtaknak megfelelően nem szabad hogy paraméter és egyéb beállításokat tartalmazzon, mert a példány már létrejöhet az előtt, hogy a szükséges paraméterek beolvasásra kerültek. Ez azért történhet meg, mert ezt a konstruktort használja a TDI grafikus felülete a kezdeti beállítási és paraméterezési felület elkészítéséhez. A konfiguráló, valamint az erőforrás foglaló műveletek ezért az Initialize metódusban kaptak helyet, ami az assembly line futás elején hívódik meg. A connector objektum egészen az assembly line végéig életben marad, majd annak végén mielőtt a destruktor meghívódna, lefut az objektum terminate metódusa. Erre azért van szükség, hogy a connector megfelelően felszabadíthassa az általa foglalt erőforrásokat.

A connector életciklus ismeretében, valamint a QRadar és a REST API-jának működése és telejsítménye miatt az alábbi megoldás mellett döntöttünk: az assembly line futása közben nem azonnal kerülnek fel az adatok a QRadar megfelelő reference datajába, hanem azok először a connector egy változójában akkumulálódnak, és az életciklus végén, a terminate metódus segítségével kommitálódnak. Két ilyen változót használtam, egyet a törlendő, egyet az újonnan hozzáadandó elemek tárolására. A megvalósítás valamint a wrapper használata miatt adott volt, hogy ezeknek az akkumulátor változóknak

a megvalósítását a QRadarApiWrapper mellé fejlesztett ReferenceData osztály, valamint leszármazottjai szolgáltassák.

A connector használatához szükséges annak megfelelő felparaméterezése. Erre a TDI a grafikus fejlesztői felületén ad lehetőséget, ahol a connector által definiált paraméterek megadására beviteli mezők állnak rendelkezésre. Minden connectornak más paraméterekre van szüksége, ezt egy tdi.xml fájl megadásával írhatjuk le, amit a connector osztályt tartalmazó JAR fájlba csomagolunk be. Ezeket aztán a GUI segítségével megadhatjuk kézzel, paraméter fájl használatával, Javascript kóddal vagy helyettesítéssel, ami akár az aktuális entitás értékeit is felhasználhatja. A QRadar connector működéséhez legalább a QRadar példány elérhetőségét, a hozzá tartozó token-t, a reference data nevét, típusát, valamint az általa használt adat típusát meg kell adni. Végeredményképp létrehoztam a felvázolt működésnek megfelelő connector-t, ami képes a paraméterezés alapján megadott QRadar példánnyal felvenni a kapcsolatot, számára adatot feltölteni, módosítani, törölni. Ez bármilyen assembly line-ban használható, későbbi projektek során is.

3.2.2. Connector tesztelése

A fejlesztés következő lépése az elkészített connector tesztelése volt, valamint a helyes működés ellenőrzése. Az ennek során szerzett tapasztalatokra építve készítettem el a 3.4. Query-k implementációja fejezetben leírt query-ket.

A teszteléshez kézzel megadtam néhány inputot, melyeket megkíséreltem feltölteni a connector segítségével egy Reference Data-ba, majd ezeket ellenőriztem a QRadar belső menüjének, valamint az API-jának használatával. A hozzáadáson kívül ellenőriztem hogy a további műveletek, a törlés valamint a módosítás is működik e.

3.3. WAS integráció megvalósítása

Ennél a megoldásnál a cél egy robusztus alkalmazás megalkotása volt, ami jól illeszkedik egy nagyvállalati környezetbe, valamint működésileg és üzemeltetésileg is összhangban van a hozzá kapcsolódó IBM-es termékekkel. Az előző szekcióban ismeretett TDI alapú megoldás ugyan ellátja a szükséges feladatokat, a fejlesztés egyszerű és gyors, de a használt technológiából adódik, hogy más területeken hátrányban van egy különálló, vagy akár egy menedszelt környezetben futtatott alkalmazással szemben. Ezeket a hátrányos tulajdonságokat hivatott áthidalni a Websphere alapú megoldás.

A TDI alapú megoldás fő limitációi:

- A magas rendelkezésre állóság, elosztott futtatás, és az adatok naprakészségének biztosítása nehezen megoldható.
- Technikai és hozzáférési információk védelme és kezelése.
- Fejlesztői és operátori / üzemeltetői feladatok nehezen választhatók szét.

A felsorolt problémák egy része valamilyen külső megoldást kíván, ami növeli az alkalmazás beállítási és karbantartási komplexitását, mivel ezekre is figyelmet kell fordítanunk, ezeket is ellenőriznünk kell esetleges hibák felmerülésekor. További probléma, hogy a TDI alapú megoldás testre szabhatósága bizonyos kereteken túl nem, vagy nagyon nehezen megvalósítható. Ilyen például a felhasználói felület kérdése. A TDI alapú megoldásnál adottak az opciók: a Configuration Editor, azaz a fejlesztői alkalmazás használata, vagy a megfelelő parancssoros lehetőségek használata. Ha saját felületet akarunk készíteni, akkor nem csak annak a fejlesztését kell megvalósítanunk, de a TDI által biztosított interfészekkel

való kommunikációt is. Ezzel szemben egy saját alkalmazás, menedzselt környezetben futtatva, az adott alkalmazásszerver által nyújtott lehetőségekkel együtt, a legtöbb felsorolt problémára megoldást nyújthat.

A Websphere alapú alkalmazás a felsorolt problémákra az alábbi módon kínál megoldást

- Futtatásra használhatjuk a WebSphere által kínált beépített ütemezőt, ami az általunk megadott szabályok szerint végrehajtja a feladatokat.
- Az paraméterek, valamint a feladatok elosztott végrehajtását képes a szerver kezelni, így a magas rendelkezésre állóságot elég ha a szerver szintjén biztosítjuk. Az adatok naprakészen tartásához emellett készíthetünk egyedi logikát, ami szükség esetén változtat a futtatandó feladatok során.
- Mivel az alkalmazás nem közvetlenül, hanem a szerver által biztosított kapcsolatokon keresztül csatlakozik a célrendszerhez, a költséges erőforrások poolozása könnyen támogatható, valamint biztonsági szempontból is elég a szervert biztosítani.
- Egy saját fejlesztésű, egyedi felülettel könnyen szétválaszthatók a fejlesztői, és a felhasználói feladatok a rendszer használatánál. A fejlesztő megvalósított néhány általános use case-t, amelyet a felhasználó később elér, hogy saját igényei szerint felparaméterezze, és használja őket.

TODO !UML diagramok?

3.3.1. Architektúra tervezése

Mivel az alkalmazás egy projekt részeként valósult meg, így többen dolgoztunk rajta, ami különösen fontossá tette az architektúra átgondolását, és alapos megtervezését.

A cél egy olyan alkalmazás fejlesztése volt, ami mind a jövőbeni fejlesztők, mind az üzemeltetéssel foglalkozó személyzet számára könnyen használható. Jelen esetben a fejlesztők alatt elsősorban azokat a személyeket értem, akik majd későbbiekben új lekérdezés (query) típusokat készítenek a rendszerhez. Ezek a személyek mélyebb ismeretekkel rendelkeznek a rendszerrel kapcsolatban, ám fontos hogy számukra is egy könnyen használható interfészt biztosítsunk. Emellett fontos hogy az új lekérdezés típusok könnyen, az alkalmazás minimális, optimális esetben semmilyen, módosításával bevezethetők legyenek. Ezzel jól szétválasztható a fejlesztők és az üzemeltetők számára szükséges tudás, mivel egy már kész, új típust egy, a rendszer belső működéséhez kevésbé értő ember is gyorsan be tud vezetni. Üzemeltetői részről egy könnyen használható felület biztosítása volt a cél, ami egyértelmű tudósítást ad a rendszer állapotáról, és megkönnyíti az egyes lekérdezések helyes felparaméterezését.

A tervezésnél az alábbi döntéseket hoztuk:

- Egységes, moduláris lekérdezésrendszer
  - A különböző lekérdezés típusok egy közös interfészt implementálnak.
  - Minden ténylegesen lefutó lekérdezés egy lekérdezés típusból, és annak a felparaméterezéséből áll. Ezeket egy táblában szerializálva tároljuk, ahonnan az ütemező felolvassa, és futtatja őket.
- Két ütemező, külön időzítéssel, egy az ISIM lekérésekhez, egy a QRadar szinkronizációhoz.
- Kétfázisú lekérdezések, több lehetséges állapottal (Lsd.: 3.2 Ábra)

  - Minden lekérdezés két fázisból áll: egy ISIM irányú lekérési fázisból, és egy feltöltési fázisból a QRadar felé
  - A lekérdezés létrejöttekor egy kezdeti állapotban van.
  - Az ütemező elindítja a lekérdezést, ekkor IDM_SYNC_RUNNING állapotba kerül, és elkezd futni a definiált ISIM lekérdezés.
  - Ha a folyamat sikeres volt, IDM_SYNC_COMPLETED állapotba kerül, ha nem, akkor IDM_FAILED-be
  - Az IDM_SYNC_COMPLETED állapot után a lekérdezés vár, amíg az ütemező el nem indítja a QRadar irányú szinkronizációt, vagy újra sorra nem kerül az ISIM ütemező számára.
  - A QRadar szinkronizáció futása ha sikeres, akkor COMPLETED állapotba kerül a lekérdezés. Ezt az állapotot nevezzük a sikeres végállapotnak.
  - Ha a QRadar szinkronizáció nem volt sikeres, akkor QRADAR_SYNC_FAILED állapotba kerül. Ekkor a lekérdezés vár, hogy a két ütemező közül valamelyik elindítsa az egyik feladatát.

- Lekérdezések eredményeinek tárolása, delta számolás
  - Egy lekérdezés futtatása közben, minden sikeresen lefuttatott fázis után szerializáljuk az eredményt egy SQL adatbázisba. Emiatt ha az alkalmazás hibára futna, vagy újraindulna két fázis között, a lekérdezés eredménye megmarad.
  - A QRadar-ra legutóbb feltöltött állapotról tárolunk egy lokális verziót, amit összehasonlítunk az ISIM lekérdezés eredményével. Ha változás történt, akkor csak a két állapot különbségét töltjük fel.
  - Ha a különbség feltöltése közben hibára futunk, feltételezhetjük hogy a rendszeren kívülről valaki módosítást hajtott végre, és a lokális változat már nincs szinkronban. Ekkor egy teljes feltöltést végzünk az adatokkal.

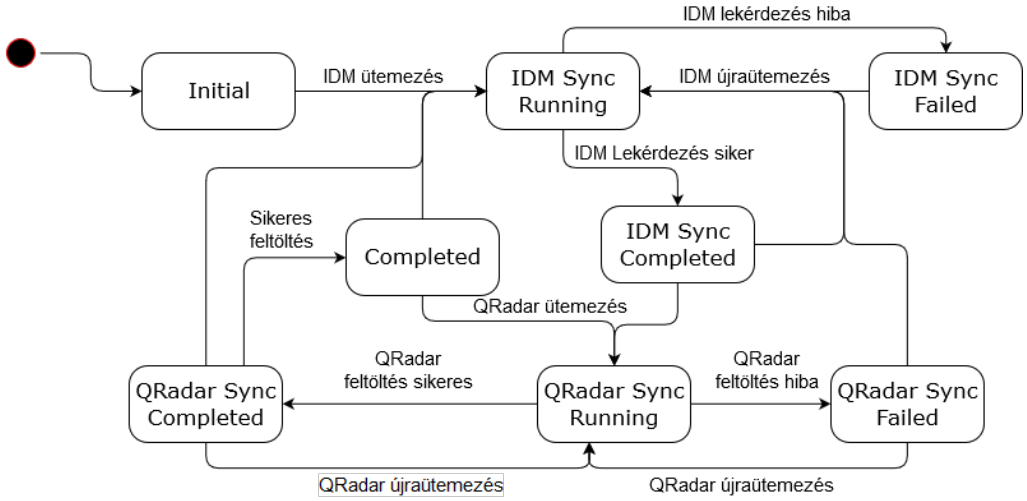
- Egyedi felhasználói felület fejlesztése
  - Olyan felület, amelyen egyszerűen láthatók a rendszerben megtalálható, már felkonfigurált lekérdezések, valamit könnyen és gyorsan új lekérdezéseket konfigurálhatunk fel.
  - Lekérdezések létrehozásának könnyítése sémafelderítéssel, aminek a segítségével a rendszerben található service-ek, organizációs egységek, szerepkörök, stb... listákból kiválaszthatók.

3.4. Query-k implementációja

A koncepció, valamint a fejlesztett megoldások tesztelésére az alábbi use-case-eket definiáltuk:

- Felhasználói fiókok egy adott menedzselt rendszeren
  - Felhasználása: Ellenőrizhető, hogy az eseményben jelzett fiók az ISIM által kezelt e, és ha nem, az okot adhat biztonsági riasztásra.
  - Megvalósítás: Szűrés az LDAP-ban található felhasználói fiókokra, aszerint, hogy melyik service-hez tartoznak.

- Inaktív felhasználókhoz tartozó fiókok



3.2. ábra. Egy lekérdezés futásának állapotai

  - Felhasználása: A SIEM riasztást adhat ha olyan tevékenység történik, ami ezekhez a felhasználói fiókokhoz kapcsolódik. Ezek elsősorban akkor következhetnek be, vagy akkor jelenthetnek kockázatot, ha a felfüggesztés a felhasználó fiókjaira valamiért nem érvényesült, és képes használni. De felhasználható visszamenőleg is, irreguláris tevékenységek keresésére.
  - Megvalósítás: LDAP szűrés az összes inaktív felhasználóra, majd szűrés az összes fiókra, az alapján, hogy tulajdonosa e az egyik megtalált felhasználó e.

- Felfüggesztési eljárás alatt álló felhasználók fiókjai
  - Felhasználása: A SIEM riasztást adhat ezen felhasználói fiókok detektálásakor. Biztonsági szempontból első sorban azért érdekes, mert a felfüggesztett személyeket általában okkal, és jogvesztéssel együtt függesztik fel (bírósági vizsgálat, kirúgás, stb), emiatt lehetséges hogy a felfüggesztés célpontja bosszúból valami ártalmas tevékenységet követ el.
  - Megvalósítás: Adatbázis keresés a jelen pillanatban futó felfüggesztési folyamatokra, majd az ebből kinyert felhasználó azonosítója alapján LDAP szűrés az összes fiók között.

- Törlési folyamat alatt álló felhasználói fiókok
  - Felhasználása: Az előző esethez hasonlóan a törlés is egy jogfosztó művelet, ami ha nem atomi módon fut le, hanem például valamilyen kommunikációs hiba miatt megakad, akkor egy ablak nyílik a rosszindulatú felhasználók számára, hogy kárt tegyenek a rendszerben.
  - Megvalósítás: Adatbázis keresés a jelen pillanatban futó felhasználói fiók törlési folyamatokra.

- Árva fiókok
  - Felhasználása: Ezek olyan fiókok, melyek nem köthetők valós, a rendszerben kezelt személyhez. Ilyenek lehetnek például a rendszer által menedszelt technikai fiókok, vagy olyanok, amik korábban valós felhasználókhoz tartoztak, de valamiért megmaradtak a szétválás után is. Ezek komoly biztonsági rést jelenthetnek, elsősorban ha például hozzáférnek kritikus rendszerekhez, de megmaradt a alapértelmezett jelszavuk, vagy nem alkalmazták rájuk a jelszó házirendeket.
  - Megvalósítás: Mivel az ISIM az LDAP adatbázisában ezeket a fiókokat külön tárolja, elég ezt lekérnünk. Majd a kinyert adatok alapján hozzárendeljük őket a megfelelő service-hez, és annak a megfelelő azonosítójához.

- Egy menedzselt rendszeren a megadott csoportokba tartozó felhasználói fiókok
  - Felhasználása: A legtöbb informatikai rendszeren létezik valamilyen felhasználói fiók csoportosítás, ami többnyire az azonos jogkörrel rendelkező fiókokat definiálja. Ha megvannak egy adott csoportba tartozó fiókok nevei, akkor könnyen létrehozhatunk irregularitás detektáló szabályokat, mint például ha a privilegizált felhasználók munkaidőn kívül lépnek be, ami jelentheti akár a fiókjuk kompromittálódását is.
  - Megvalósítás: LDAP szűrés az előre definiált csoport attribútum alapján, az adott service-hez tartozó felhasználói fiókokon.

3.5. QRadar esemény küldő fejlesztése

A feladat motivációja, az eddig felsorolt problémákhoz hasonlóan, a QRadar monitorozási és detektálási hatékonyságának bővítése az ISIM segítségével. Az előző két implementált megoldásban az ISIM-ben tárolt felhasználói adatok egy részhalmazát tettem elérhetővé a QRadar szabályrendszere számára, mert ezek a plusz információk hasznosak lehetnek az események értelmezésében. Ezzel szemben ennél a megoldásnál az ISIM mint log forrást illesztettem a QRadarhoz. Egy ilyen megoldás már rendelkezésre állt, de az a QRadar JDBC csatlakozóját használja, ami limitált képességekkel bír (például nem join-olhatók vele táblák), és csak az ISIM-ben található audit információkat kezelte.

A fejlesztett megoldás célja más, nem audittal kapcsolatos információk küldése a QRadar számára log formájában. Ezek is fontosak lehetnek biztonsági eseményeknél, hiszen például az ISIM-ben futó/futott folyamatok információit felhasználva új incidenseket vehetünk észre. Ilyenek lehetnek jelszó változtatások (például egy széles jogkörrel rendelkező felhasználó jelszó cseréje nem várt időpontban), az ISIM szabályrendszerének változása, vagy például kézi jóváhagyás műveletek.

A folyamatokkal kapcsolatos információkat az ISIM a saját DB2 adatbázisában tárolja. Minden folyamathoz tartozik egy rekord a PROCESS nevű táblában. Attól függően hogy az adott folyamat pontosan hogy van definiálva, lehet hogy más folyamatok is meghívódnak egy futás közben. Az egyes folyamatok konkrét implementációjával kapcsolatos információk az ACTIVITY táblában, míg a folyamat lefutásával kapcsolatos audit események a PROCESSLOG táblába kerülnek. Az események generálásakor elsősorban ezzel a három táblával dolgoztam.

A felsorolt táblákból kinyerhetők a folyamattal kapcsolatos technikai információk, mint a folyamat típusa, kezdeti ideje, általa hivatkozott egyéb folyamatok és activity-k. Emellett viszont olyan adatokat is tárolnak a táblák, amik a folyamatban résztvevő személyeket azonosítják. Ha ezeket az adatokat képes feldolgozni a QRadar oldali esemény fogadó, akkor az előzőekben ismertetett megoldás által feltöltött adatok segítségével kialakíthatók olyan szabályok, amik fontos incidenseket detektálhatnak.

3.5.1. TDI alapú syslog küldő fejlesztése

Az ISIM-ben található adatok feldolgozására, és ezekből syslog események generálására a már az előzőekben bemutatott TDI keretrendszert használtam. Ez kézenfekvőnek tűnt, mivel a TDI biztosít connectorokat mind az ISIM DB2 adatbázisa irányába, mind a syslog események generálásához és küldéséhez. Előbbire a Java JDBC protokollt használó JDBC connector-t, utóbbira a beépített Log connector-t, ami sok különböző standard logoló motor mellett a syslog-ot is támogatja.

A fejlesztés első lépése a táblák és a bennük található adatok felmérése volt. Ez alapján az alábbi következtetésekre jutottam:

- A PROCESS tábla az egyes folyamatok operatív információit tartalmazza.

  - A legfontosabb információk itt találhatók, többek közt: folyamat indítója; folyamat alanya; organizációs egység, amelyben fut; folyamat típusa és eredménye; indulási és befejezési időpont

- Az ACTIVITY tábla sorai az egyes folyamatokhoz tartozó elemi akciók információit tartalmazza.
- A PROCESSLOG tábla a folyamat egyes lépéseinek a kiegészítő és audit információit tartalmazza.
- A három tábla közül a PROCESSLOG tábla tartalmazza az legkisebb felbontásban az folyamattal kapcsolatos információkat.
- A folyamatban résztvevő felhasználókkal kapcsolatos legfontosabb adatok a PROCESS táblában találhatók.

Ezek alapján megvalósítottam az adatok kigyűjtését a TDI segítségével. A lekérést egy 3 táblából álló join művelettel végeztem el, melyben a PROCESSLOG táblát a PROCESS táblával a PROCESS_ID oszlopon keresztül, az ACTIVITY táblát pedig az ACTIVITY_ID oszlopon keresztül kötöttem össze. A teljesség kedvéért mindegyik tábla mindegyik mezőjét kigyűjtöttem, további felhasználási célra. Mivel ez a TDI-ban ütközést okozott az azonos nevű mezőkön, ezért minden oszlopot a tábla nevével prefixáltam.

A következő lépés az adatok átalakítása volt a QRadar számára könnyen kezelhető formára. Mivel a feldolgozás egyik módja a reguláris kifejezések használata, így kézenfekvő volt egy olyan struktúra kialakítása, amire jól illeszthetők ilyen kifejezések. Emiatt végül az alábbiak mellett döntöttem:

- Az összes attribútum összefűzése egy folytonos karakterlánccá.
- Az összes attribútum értéke elé az adott attribútum nevének hozzáfűzése. Például a PROCESSLOG tábla EVENTTYPE mezőjénél ez az alábbi formátumot eredményezte: PL_EVENTTYPE=érték
- Az egyes attribútumok elválasztása pontosvesszővel. Azért erre a karakterre esett a választásom, mert ezt gyakran használják ilyen célra, például az LDAP konvenciók szerint is, és pont emiatt az LDAP attribútumok értékében egy tiltott karakter, és bár közvetlenül relációs adatbázisból szelektálunk, ezek a táblák többnyire az LDAP adatbázisból kinyert, vagy ott is tárolt információkat tartalmaznak.
- Az új sor karakterek eltávolítása, valamint a pontosvesszők TODO !kieszképelése az attribútumok értékeiből.

A generálás utolsó lépése a sorok felküldése volt syslog protokollon keresztül a QRadar megfelelő fogadó interfészére. Ehhez a beépített Log connector-t használtam, egy egyedileg konfigurált Log4J logger segítségével. Az egyedi konfiguráció definiálja, hogy a beépített syslog logger legyen a használt logolási mód, milyen IP-re és portra küldjük az üzeneteket, valamint milyen formátumban.

A Log4J által biztosított syslog logger azonban alapértelmezetten nem volt megfelelő erre a célra, mivel úgy implementálták azt, hogy az 1019 byte-nál hosszabb üzeneteket tördelje megfelelő méretűvé, és több részben küldje tovább. Ez a QRadar oldalán azt okozta, hogy a logsor tördelékek külön eseményenként kerültek rögzítésre, ami nem volt megfelelő a kívánt működés szempontjából. A QRadar azonban biztosít egy megoldást a

töredékek kezelésére: az UDP multiline syslog[2] kezelést, amihez a töredék üzeneteket ki kellett egészítenem azonosítókkal.

A probléma megoldására átalakítottam a TDI assembly line-t úgy, hogy a tördelési műveletet saját magam végzem, a megadott szabályok szerint. A tényleges payload-ot maximum 800 karakter méretű szeletekre tördelem, figyelve az attribútum határokat jelölő pontosvesszőkre. Így, a képzett syslog header-rel együtt, az üzenetek nem haladják meg a limitet, és egyben kerülnek elküldésre.

A szükséges egyedi azonosítókat szintén TDI-ban generálom. Mivel nem kriptográfiailag biztonságos véletlenek generálásáról van szó, hanem csak egy megfelelően nagy spektrumban egyedi azonosítókról, ezért ehhez egy egyszerű, Javascript alapú pszeudo random uuid generálást használok. Ez 8 darab 0000 - ffff értékig terjedő stringből áll, ami összesen 16^{4*8} darab egyedi kombinációt ad, ami a feladat szempontjából elégséges méretű tartományt. Ezzel a lépéssel az adatok előálltak, és a QRadar számára olvasható formátumba kerültek.

3.5.2. QRadar oldali esemény fogadó fejlesztése

Ezek után a QRadar oldali fogadást kellett biztosítanom. Ezt a már említett UDP multiline syslog segítségével tettem. Ezt egy eseményforrás felkonfigurálásánál kell megadni, és annyiban különbözik az átlagos syslog protokollon érkező üzenetektől, hogy az 514-es port helyett az 517-est használja, valamint a beérkező üzeneteknek rendelkeznie kell egy azonosítóval. Ha az azonosító két vagy több üzenetben megegyezik, akkor azokat a QRadar összefűzi egy eseménnyé.

Ezek alapján felkonfiguráltam egy új eseményforrást IBM Identity Manager néven, ami UDP multiline syslog-okat fogad. Ezek azonosításához egy *msg_uuid* mezőt keres, az alábbi regex segítségével:

```
msg_uuid=(.*?[^\\]);
```

Ezután a QRadar-t kellett felkészítenem az események kezelésére. Ehhez létrehoztam egy saját esemény típust, amelyet egy saját syslog header alapján azonosítok. A típus határozza meg, hogy a QRadar milyen szabályok szerint, milyen attribútumokat próbál meg az azonosított eseményből feldolgozni, valamint azokat hogyan használja fel a továbbiakban. Ehhez elkészítettem az összes lehetséges felküldött attribútumhoz a megfelelő regex kifejezést. Mivel az adatok normalizálásánál egy egységes szisztémát követtem, ezért az összes parse-oló regex az alábbi sémára épül:

```
  ATTRIBÚTUM_NÉV=(.*?[^\\]);
```

Ennél a reguláris kifejezésnél az attribútum értéke pontosan az első találati csoportban érhető el, valamint a felépítése miatt megfelelően kezeli az TODO !eszképelt pontosvesszőket is.

TODO !Screenshot előtte, utána

[2] UDP multiline syslog dokumentáció

4. fejezet

Összefoglalás

A féléves munkám során a következő feladatokat végeztem el:

- Megismerkedtem az IdM és a SIEM területekkel.
- Tanulmányoztam és tapasztalatot szereztem az IBM IdM és SIEM megoldásaival, továbbá a TDI és WAS szoftverekkel.
- Kidolgoztam a projekt többi résztvevőjével két rendszer integrációjának lehetséges architektúráját és technikai részleteit.
- Megvalósítottam egy általánosan használható QRadar Java API-t, amely a QRadar REST API-jának az integráció szempontjából releváns részét Java programozói környezetben teszi elérhetővé.
- Fejlesztettem egy TDI connector-t, amely a TDI segítségével megvalósuló integráció alapja, és egyben egy általánosan használható eszköz QRadar reference adatok kezelésére más, TDI-vel megvalósított integrációs feladatokra is.