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. december 7.

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. december 7.

Bulla Ádám
hallgató

Kivonat

Az informatika fejlődésével és terjedésével egyre fontosabb terület lesz az informatikai biztonságtechnika ami rengeteg szerteágazó feladatot foglal magába. Jelen dolgozat ezen belül kettő területre, a biztonsági incidensek és események kezelésére, valamint a felhasználó és hozzáférés menedzsmentre fókuszál. Mindkét területre léteznek már ipari sztenderd megoldások, mint például az IBM QRadar, és Identity Manager terméke, azonban a két terület összekapcsolására nem volt még megoldás.

A dolgozat célja egy ilyen integráció elkészítése és bemutatása a fent említett két termék használatával. A cél a felhasználói, és a hozzájuk kötődő folyamat információk összegyűjtése, transzformálása, majd prezentálása a QRadar számára a szabályrendszerében való felhasználásra. Ezek segítségével az egyes események új kontextusban, új információk birtokában értékelhetők ki, ami eddig nem kezelt biztonsági incidensek észlelését teszi lehetővé.

A dolgozat az információk elérhetővé tételére két megoldást mutat be: egy JAVA EE alapú webalkalmazást, és egy IBM adatintegrációs eszköz alapú megoldást. A két rendszer közti további integrációra az említett mellett egy saját megoldást is készítettem, ami olyan, a felhasználói folyamatokkal kapcsolatos információkat továbbít a QRadar számára, amik eddig kihasználatlanok voltak.

Abstract

With the continous advancement of computers and technology, IT Security is becoming an increasingly important field of study, which envelops a multitude of tasks. This thesis focuses on two fields of IT security: security information and event management, and identity and access management. Both of these have had industry standard solutions for years, like IBM QRadar, and IBM Security Identity Manager, but there were no previous solutions for connecting the two.

The goal of this thesis project is to implement and present a solution for this problem, using the aforementioned two products. The end goal is the collection, transformation, and presentation of user identity and access information, including the processes managing these, for QRadar. With this data, the security events handled by QRadar can be evaluatied in a new context, detecting previously unhandled incidents.

This thesis presents two solutions for this integration: a Java EE based web application, and a custom solution, using an IBM data integration framework. For further integration between the two products, I've also developed a different integration solution, which generates security events for QRadar, from previously unutilized information about processes managing user data and access information.

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, valamint 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 megvalósítottam 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 5. 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 hordoznak biztonsági szempontból. A modulhoz használt keretrendszert a TDI [3] nyújtja.

Mindkét 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 IBM Security Identity Manager - ISIM
[2] Lásd 2.1.1 IBM Security QRadar SIEM
[3] Lásd 2.1.3 Tivoli Directory Integrator - TDI

1.4.1. Integráció TDI segítségével

A Tivoli Directory Integrator (TDI [4]) egy gyakran használt, általános célú integrációs eszköz. A TDI a segítségével a különböző adatforrásokat, amelyek más-más protokollon keresztül érhetők el, és más-más formátumban és struktúrában tárolják az adatokat, egy egységes ú.n. Connector interfészen keresztül érhetjük el. Az adattranszformációs feladatokat ú.n. assembly line formájában valósíthatjuk meg a TDI grafikus felületén. Egy assembly line egy műveletsort definiál, amely során felhasználhatjuk a connectorokat adat olvasásra, keresésre, kiírásra, valamint Javascript segíségével tetszőleges adattranszformációt valósíthatunk meg. A megfelelő protokoll és parser kiválasztásáról a használt connector gondoskodik, így az assembly line kompakt módon csak a valódi adattranszformációt definiálja.

A TDI gyári connectorkészletét kiegészítettem egy új, saját fejlesztésű connectorral, amely a fent említett wrapper könyvtár segítségével képes reference adatokat feltölteni és lekérni a QRadartól. A hálózati erőforrások hatékony használatának céljából a connector külön akkumulálja a feltöltendő adatokat, és a futása végén, egyben tölti azokat fel. Ez a connector általános célú TDI connector, így szabadon újra felhasználható nem csak az ISIM-mel való integrációra, hanem tetszőleges TDI assembly line formájában megvalósított integrcáiós feladatra.

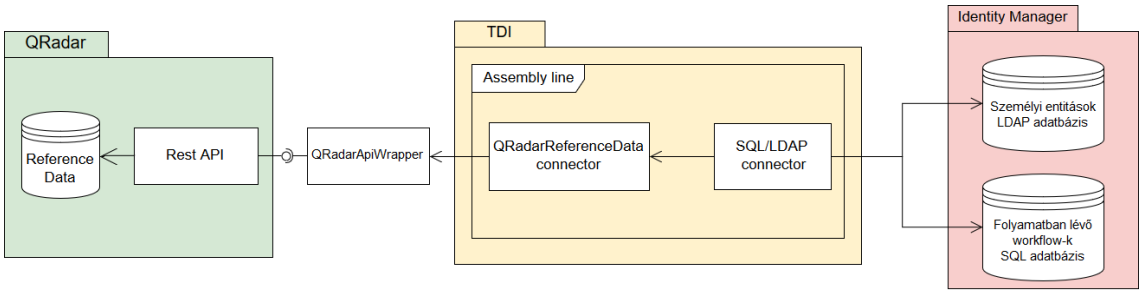
Ebben a megvalósításban a QRadar és az ISIM közti integrációt TDI assembly line-ok formájában valósítottam meg, melyeket a TDI által biztosított szerver komponens dolgoz fel és futtat. A TDI gyári funkcióit és grafikus interfészét felhasználva az egyes assembly line-ok fejlesztése időtakarékos és költséghatékony. Ennek a megvalósításnak hátránya, hogy a megoldások a TDI-hez kötöttek, valamint a lehetőségeknek határt szabnak a TDI által nyújtott lehetőségek. Egy újabb lekérdezés vagy lekérdezéstípus konfigurálása, ütemezése a TDI fejlesztői felületén történik, az üzemeltetési feladatokhoz a TDI ismerete szükséges.

1.4.2. Integráció webes (WAS) alkalmazás formájában

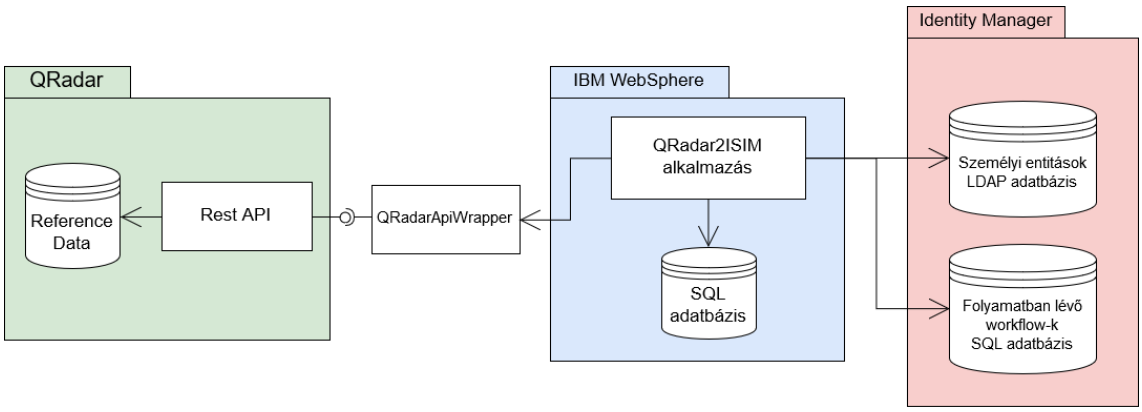
Ennél a megoldásnál egy IBM WebSphere Application Server (WAS) környeztre feljlesztett alkalmazást készítettem, amely a QRadarral való kommunikációra felhasználja az általam fejlesztett wrapper-t. Az alkalmazás rendelkezik egy felhasználóbarát webes felülettel, melyen keresztül létrehozhatunk és felkonfigurálhatunk 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 között, akkor egy teljes szinkronizációval minden adatot felküld, ezzel egy új, konzisztens állapotba álltva a rendszert.

Ebben a megvalósításban a TDI alapú megoldáshoz képest előny, hogy a WAS egy menedzselt környezetet biztosít a futtatáshoz, így jobban felügyelhetők az egyes feladatok, valamint igény szerint könnyen megvalósítható az elosztott, klaszteren futó, hibatűrő működés is. Előny továbbá, hogy saját grafikus felhasználói felülettel rendelkezik, amelyen az üzemeltetési feladatok könnyen elvégezhetők. Új lekérdezéstípus létrehozása Java fejlesztői ismeretet követel meg, konkrét technológiához kötött (pl. TDI) tudás nem szükséges. A

[4]Lásd 2.1.3 Tivoli Directory Integrator - TDI



1.1. ábra. Architektúra TDI esetén



1.2. ábra. Architektúra WAS esetén

WAS minden ISIM környezetben elérhető, ezért nem szükséges új komponensek telepítése az modul használatához.

A TDI-t használó megoldással 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.

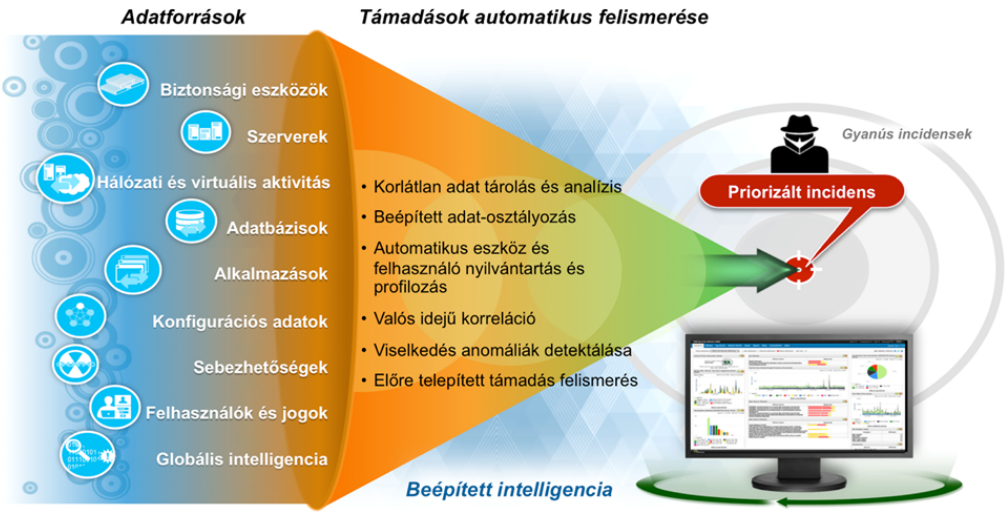
2.1.1. IBM Security QRadar SIEM

Az IBM Security QRadar SIEM az IBM security information and event manager rendszere, ami lehetővé teszi hálózatra csatlakoztatott eszközöknek a megfigyelését biztonsági szempontból.[7] 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 számos 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.

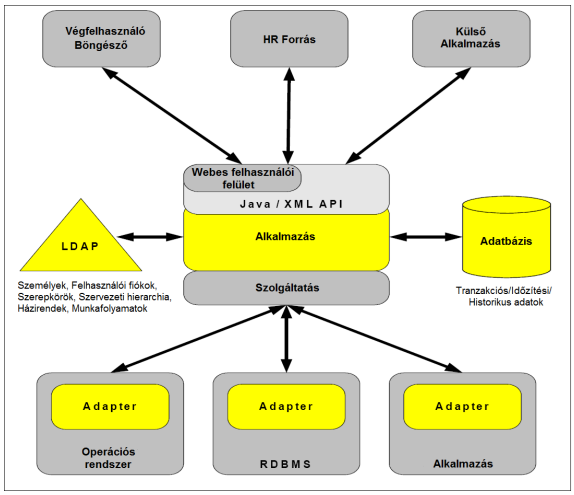
információk alapján változzon a kiértékelés, és ha valamilyen adat frissül, akkor naprakész maradjon a szabály által talált incidensek halmaza. A dinamikusan feltölthető adathalmazok (összefoglaló nevükön reference data) elérhetők egy REST API-n keresztül, így könnyen hozzájuk lehet férni és módosítani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

Emellett a reference data-ban található adatoknak lehet egy time-to-live (TTL – elévülési idő) értéke is, ami meghatározza, hogy mennyi idő elteltével törlendő az adott adat. Ennek 3 típusa lehet:



2.2. ábra. Az ISIM architektúrája és interfészei.

- UNKNOWN - A TTL érték nem kerül felhasználásra.
- LAST_SEEN - Az adat utolsó feltöltésétől számítva kalkulálódik a TTL
- FIRST_SEEN - Az adat első feltöltésétől számítva kalkulálódik a TTL

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.

2.1.2. 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.[9]

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.[8] 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. Minden connector, attól függően, hogy milyen műveletet végez 8 mód egyikében működik[1], valamint vagy aktív, vagy passzív módon dolgozik. Az előbbinél része az assembly line soros végrehajtásának, másikban nem, de külső vezérléssel működtethető.

A különböző adatforrások más-más formátumban kezelik az adatokat, így 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 számos átalakítási lépés áll rendelkezésre, mint például ciklusok vagy elágazások használata. Az assembly line-on haladó adatokat entry-k formájában kezeli a TDI.[5] Egy entry egy elemi objektum, ami nevesített attribútumok halmazát képes tárolni. Ilyen entry-t bármelyik scriptben létre lehet hozni, de egy általános TDI assembly line esetén 4 darab nevesített entry-t használunk:

- work: Az assembly line állandó entry-je. Ezen az objektumon keresztül utazik az adat az egyes komponensek között, valamint ezen hajtják végre az adatmódosító műveletket a connector-ok.
- conn: Ezt az entry objektumot használják a connecot-ok a köztes adatok tárolására a célrendszerrel való kapcsolat során. Ebből az entry-ből kerülnek át az egyes mapping-ek során az adatok a work entry-be.
- current: Bizonyos connector-ok update módjainál elérhető változó, ami a csatlakoztatott rendszeren található adatokat tartalmazza.
- error: A hibakezelést végző komponensekben elérhető entry. A hibákkal kapcsolatos információkat tartalmazza.

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.

[1] A módok pontos leírása megtalálható a 3.2.1 szekcióban.

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.

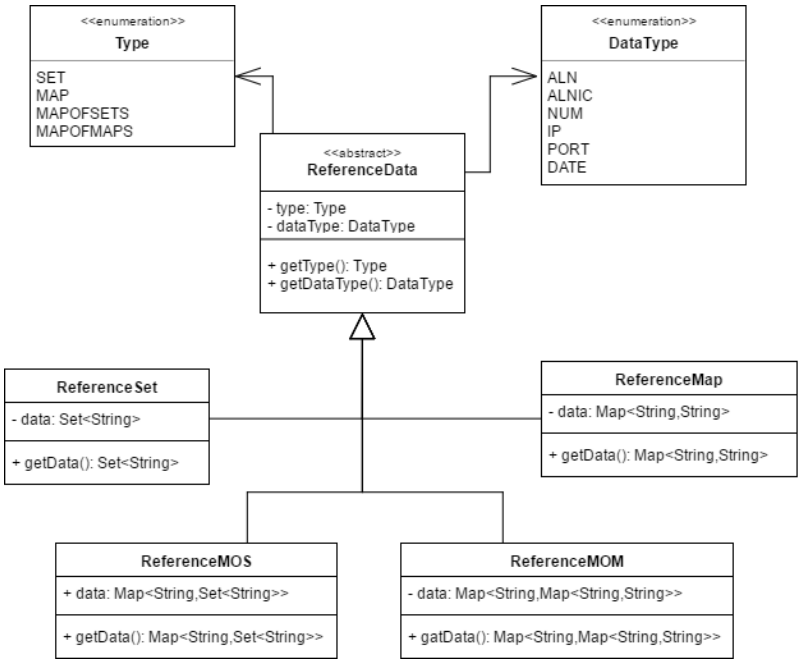
  /{name} - GET, POST, DELETE. Az URL-ben megadott paramétert a QRadar a reference set neveként értelmezi, és ezen keresztül érhető el a set lekérése (GET), teljes törlése (DELETE), valamint egy elemi adat feltöltése (POST).

  /{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[12] 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, valamint 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.

A wrapper támogatja a QRadar által használt összes adattípust (ALN, ALNIC, IP, NUM, PORT, DATE), az REST hívások be- és kimenetét az adott típus által megkövetett JSON formátumban állítja elő/dolgozza fel.

Ezen kívül a wrapper kezeli az elévülési (TTL) paramétereket is, amelyek két részből tevődnek össze: az egyik az elévülés ideje, a másik típusa, vagyis milyen szabály alapján évüljenek el az adatok. Ennek a lehetséges értékei az UKNOWN, a LAST_SEEN, és a FIRST_SEEN, amik azt határozzák meg, hogy az elévülési időt honnan számoljuk: az adott adat először bekerülésének idejétől, vagy az utolsótól. Az elévülési időt szöveges formátumban (pl.: 36 hours) várja a QRadar API-ja, amit feldolgoz, és valós idő értékekké alakít.

3.2. TDI Integráció megvalósítása

A TDI alapú megoldás célja egy olyan keret, és hozzá tartozó use case-ek kidolgozása, amely lehetőséget ad az ISIM és QRadar közti integrációs feladatok gyors, hatékony és egyszerű megvalósítására. A TDI már létező LDAP és adatbázis (JDBC) connectorral rendelkezik, ezért az ISIM adatforrásként való használata gyári komponensekkel oldható meg. A QRadar API használatához viszont új connector fejlesztésére volt szükség. 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 több lekérdezés típust, általunk, valamint az ügyfél által definiált esetekre.

3.2.1. QRadar connector fejlesztése TDI-hoz

A TDI alapú megoldás első lépése egy connector fejlesztése volt a TDI és a QRadar közti integrációhoz. Ehhez segítségemre volt a hivatalos útmutató connector fejlesztéshez. [6]

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 készítünk egy xml alapú leíró fájlt, ami meghatározza a TDI számára, hogy a connector milyen paramétereket vár, azokat milyen formában, valamint a felhasználói felületen az egyes paraméterekhez milyen GUI elemek tartozzanak. Ezt a leírót, valamint a lefordított .class fájlokat a megfelelő struktúrában becsomagoljuk egy JAR fájlba, amit a TDI által használt könyvtárak egyikébe másolunk.

Egy connector-nak nyolc működési módja lehet, amit a feljebb említett dokumentum pontosabban is 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 - Egy kritérium alapján keresést hajt végre az adatforráson és kiválasztja a kritériumra illeszkedő elemet vagy elemeket. Jellemzően több adatforrás elemeinek illesztésére használatos.
- Delete - Az assembly line-ról kapott összes elem esetén a megadott kritériumot felhasználva megkeresi az elemet, és ha megtalálta, törli azt. Lehetőség van olyan felparaméterezésre is, amely egyszerre több elemet töröl.
- Update - Már meglévő adatok módosítását végzi. Előbb megkeresi a megadott kriétérium alapján a 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. Eltárolja a kezelt elemek legutóbb használt verzióját, és csak az újonnan beolvasott elemekhez képesti különbséget hajtja végre.
- Server - Ebben a módban a connector egy szerver típusú viselkedést valósít meg, ahol vár egy bejövő kapcsolatra (TCP, LDAP, HTTP, stb.), aminek a hatására egy új szálon elindítja a saját assembly line-ja egy másolatát, ami a beérkező adatokkal végrehajtja a feladatot, választ küld a bejövő kérésre, majd terminálódik. Az eredeti példány közben újabb kapcsolatokra vár.
- CallReply - Egy speciális mód, ami olyan interakciók végrehajtására lett tervezve, amelyeknél attribútumok transzformációját (vagy új attribútum előállítását) a connector által kezelt külső komponens végzi. Működése a lookup módhoz hasonló, azzal a különbséggel, hogy keresési kritérium helyett egy attribútumhalmazt halmazt adunk át, amelyen tetszőleges műveletet végrhajthat a külső komponens. A connector ilyenkor be- és kimenetei attribútumokkal is rendelkezik.

Egy connector nem feltétlenül támogat minden módot, a támogatni kívánt módok határozzák meg, hogy a fejlesztendő connector-nak milyen metódusokat kell 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, kivéve a Server, és a CallReply, mivel ezek a konkrét kontextusban nem értelmezhetőek.

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 az assembly line lezárultakor lefut a terminate metódusa. Erre azért van szükség, hogy a connector megfelelően felszabadíthassa az általa foglalt erőforrásokat és a nyitott kapcsolatokat.

A connector életciklus ismeretében, valamint a QRadar és a REST API-jának működése és telejsítményének optimalizálása miatt úgy döntöttünk, hogy 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 hívásakor 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.

Az alábbi metódusokat implementáltam a connector által támogatott metódusokhoz:

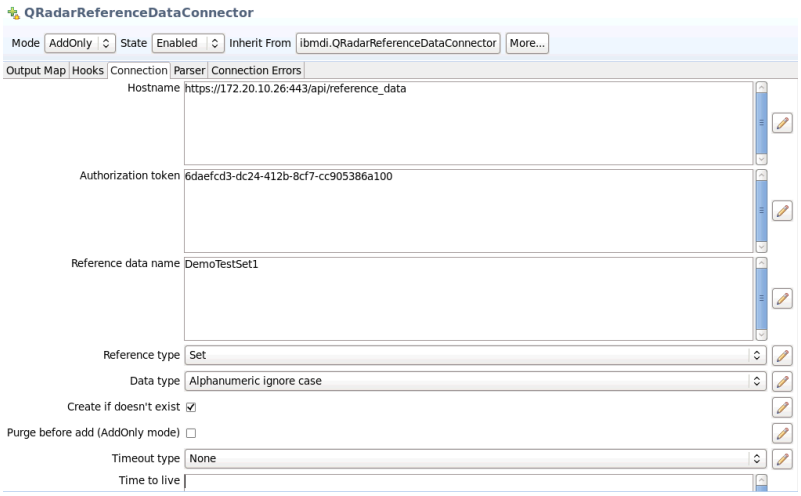
- initialize - Az inicializálást végrehajtó metódus. Alapértelmezetten az assmbly line indulásakor hívódik meg. Lekéri az ősosztály segítségével a felületen megadott paramétereket és azokat megfelelő formátumra hozza, valamint inicializálja az olyan attribútumokat, amelyek a paraméterezés alapján más értékeket vehetnek fel. Emellett ha az opció használatban van, akkor létrehozza az új reference data-t.

- findEntry - Keresést végrehajtó metódus. A megadott SearchCriteria alapján végrehajt egy keresést a kiválaszott reference data elemein. Mivel a QRadar API nem definiál keresés műveletet, a keresést egy előzetesen letöltött teljes adathalmazon kell végrehajtani. Elkerülendő, hogy minden egyes findEntry híváskor újra le kelljen kérdezni a teljes adathalmazt, ez a metódus egy gyorsítótárazási megoldást használ. Az első keresésnél lekérdezi a teljes reference data-t, majd ebben a lokális példányban keres az összes többi futása során.

- putEntry - Hozzáadást, azaz adat kiírást végrehajtó metódus. A paraméterként kapott entry-t feldolgozza, és a connector számára értelmezett kulcsok (*va-*

lue,*key*,*inner_key*, *payloaf*) mentén hozzáadja őket a megfelelő akkumulátorokhoz. Ha payload érkezik, amit egy az egyben továbbítunk a QRadar felé, akkor egy Stringeket tartalmazó listához adja hozzá, ha key, value, inner_key formátumban érkezik adat, akkor pedig a megfelelő reference data akkumulátorhoz.

- modEntry - Módosítást végrehajtó metódus. Először feldolgozza a paraméterként kapott új entry-t, a régi entry-t, valamint a keresési feltételeket az alapján, hogy milyen típusú reference data-t kell módosítania. Ezután mind a négy típusra végrehajtja a szükséges lépéseket a sikeres módosításhoz. Ezek a következők:
  - Set: A régi értéket hozzáadja a törlendőket tartalmazó akkumulátorhoz, az újat pedig a hozzáadandóakhoz
  - Map: Ha az új érték kulcsa nem egyezik a régi érték kulcsával, akkor a régi értéket hozzáadja a törlendőket tartalmazó akkumulátorhoz, az újat pedig a a hozzáadandóakhoz.
  - Map of sets: A Set-hez hasonlóan, a régi értéket a törlendőhöz, az újat a hozzáadandóhoz adja.
  - Map of maps: A map-hez hasonlóan ellenőrzi, hogy a key, valamint az inner_key megegyezik e, ha nem, akkor hozzáadja a törlendőhöz a régit, az újat pedig a hozzáadandóhoz.
- deleteEntry - Törlésért felelős metódus. Feldolgozza a beérkező entry-t az alapján hogy milyen típusú reference data-t kezel, és hozzáadja a törlendő adatokat tartalmazó akkumulátorhoz.
- terminate - A connector terminálásáért, lezárásáért felelős metódus. Általában a felépített adatbázis kapcsolatokat zárja le, de jelen esetben ez a metódus végzi az akkumulátorok tartalmának feltöltését, valamint az ehhez szükséges előkészületeket (pl.: adatok előzetes kiürítése). Először feltölti az entry-ként érkező adatokat, majd a payloadokat küldi fel, és végül végrehajta a törlést a törlendő adatokkal.
- querySchema - Segéd metódus, ami TDI grafikus felülete számára szolgáltat információkat a connector által kezelt adatsémáról. Ha még nincs felkonfigurálva a connector, vagy nem érhető el a QRadar megfelelő reference data-ja, akkor a TDI ezen a metóduson keresztül szerez tudomást arról, hogy a connector milyen attribútumneveket használ a beérkező entry-kben.
- selectEntries - Iterator módnál használt metódus, ennek segítségével inicializálódik az adat, amit az Iterator mód végül entry-k formájában visszaad. Jellemzően egy adatlekérdezés, amely előállítja és cache-eli az adathalmazt, amely egyes elemein az assembly line végrehajtódik. Esetünkben a kiválasztott QRadar reference data elemiet kérdezi le és tárolja memóriában.
- getNextEntry - Iterator módnál használt metódus, egyesével visszaadja a megfelelő reference data-ban található rekordokat, entry-k formájában. Ehhez két iterátort használ, amelyek a selectEnries metódus által előállított adatokon iterál. Ezek állapota két metódushívás között állandó marad. Az első iterátor végzi az iterációt az adatokon, vagy összetett adatok esetében a külső elemeken (map of maps és map of sets esetén a kulcsokon), a második pedig a belső kulcsokon/értékeken iterál.
- getVersion - String formájában visszaadja a connector verzióját, amit a TDI használ fel.

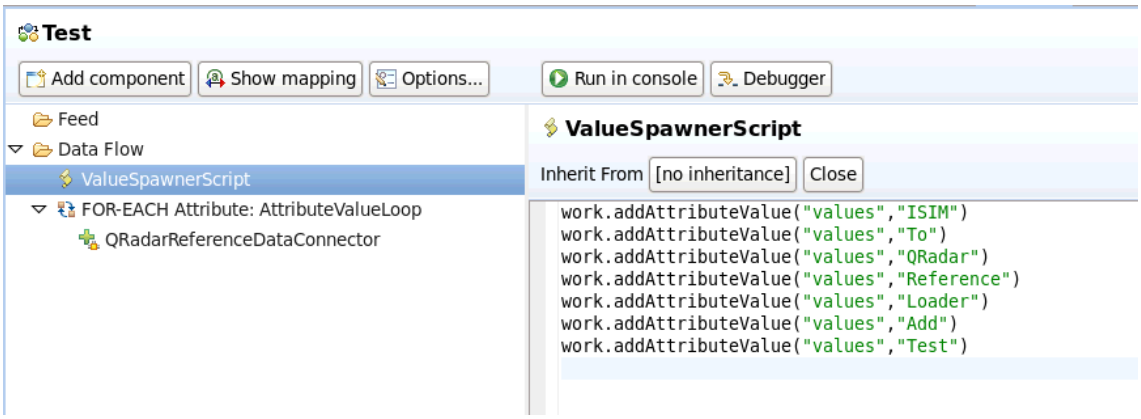


3.2. ábra. Minta a QRadar Reference Data Connector felparaméterezésére

- populateCache - Saját segédmetódus, nem része a TDI connector interfészének. A keresésnél ez a metódus hívodik meg, ami letölti a megfelelő reference data-t.

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. Az általam fejlesztett connector-hoz én is elkészítettem egy ilyen xml fájlt, ami az alábbi paramétereket képes kezelni:

- Hostname (szöveges mező): URL amin keresztül elérhető a QRadar reference data API-ja
- Authorization token (szöveges mező): Token, amit a QRadarral való kapcsolódáskor az autentikációra használunk.
- Reference data name (szöveges mező): A reference data neve, amire az adatok feltöltésre kerülnek.
- Reference type (legördülő lista): A használni kívánt reference data típusa. Lehetséges értékei: Set, Map, MOS, MOM.
- Data type (legördülő lista): A reference data-ban tárolandó adatok típusa. Lehetséges értékei: Alphanumeric, Alphanumeric ignore case, Numeric, Date, IP, Port
- Create if doesn't exist (check-box): Igaz/hamis érték, azt befolyásolja, hogy a connector létrehozza-e a reference data-t, ha az még nem létezik.
- Purge before add (Add only): Feltöltés előtt törölje-e a reference data tartalmát. Ez az opció csak AddOnly mód esetén jut érvénye.
- Timeout type (legördülő lista): A Time to live paraméterben megadott értéket milyen módon értelmezze. Lehetséges értékei: None, UNKNOWN, LAST_SEEN, FIRST_SEEN. Azt befolyásolják, hogy honnantól számítson a time to live értéke.



3.3. ábra. Adatok kézi megadása QRadar Reference data connector teszteléséhez

- Time to live (szöveges mező): Mennyi ideig legyen elérhető az adat, a timeout type alapján. Az időtartam szöveges formátumban várja az értéket. (pl *"36 hours"*)
- Comment: Milyen magyarázattal kerüljön feltöltésre az adat
- Detailed log (check-box): A connector futtatásakor a naplóinformációk extra információkat is tartalmazzanak e.

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.

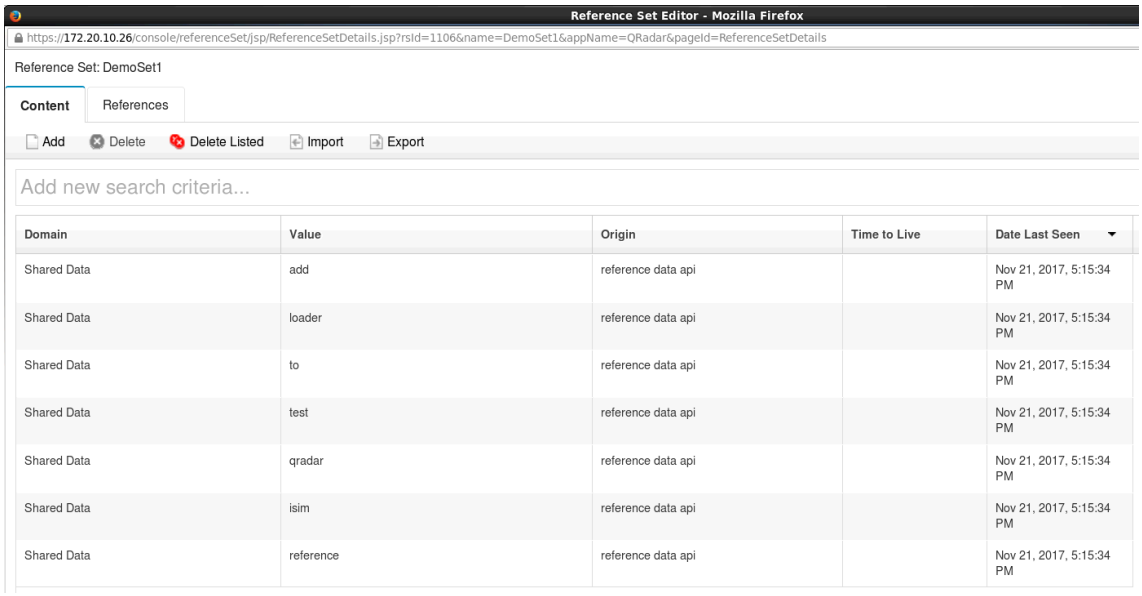
3.2.2. Connector működésének bemutatása

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 integrációs feladatok megoldását. A teszteléshez kézzel megadtam néhány inputot, melyeket megkíséreltem feltölteni a connector segítségével egy Reference Data-ba, majd ezeket ellenőriztem a QRadar belső menüjének használatával. A 3.3-es Ábrán látható, hogy egy Javascript blokk formájában feltöltöm az assembly line fő változójának értékét egyedileg definiált szövegekkel. A bal oldalon látható a teszt Assembly line, ami a feltöltő script-ből áll, a QRadar Connectorból, valamint egy ciklus vezérlőből, ami az összes string értéket egyesével továbbítja a QRadar Connector felé.

A folyamat eredményességét, vagyis hogy az összes általam definiált szöveg felkerült e, a QRadar reference set módosító felületén keresztül ellenőriztem. Az végeredmény a 3.4 Ábrán látható.

A továbbiakban bemutatom egy példán keresztül egy integrációs feladat megvalósítását, ami az ISIM-ben található árva account-okat szinkronizálja a QRadar számára. A feladat bővebb leírása, felhasználási területei a 3.4.5 fejezetben található.

Minden TDI assembly line két főbb részből áll: a Feed-ből és a Data Flow-ból. A Feed rész tartalmazza az olyan connectorok-at, amik iterator módban adatokat olvasnak fel, és ezekből új entry-ket készítenek az assembly line részére. Ezeken az entry-ken haj-



3.4. ábra. Sikeresen feltöltött adatok a QRadar felületén

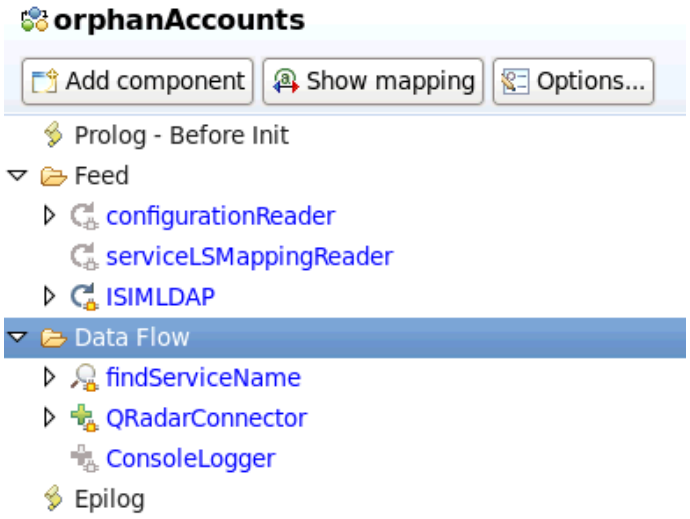
tódnak végre a Data Flow részben definiált connector-ok és script-ek által meghatározott műveletek.

Jelen esetben a fő adatokat az ISIMLDAP connector szolgáltatja. Ez kapcsolódik az ISIM által használt LDAP szerverhez, és felolvassa az árva account-okhoz tartozó információkat. A Feed-ben található másik két connector kiegészítő információk felolvasására szolgál. Ezek passzív módban vannak jelen, az Assembly Line prolog script-je ezeket felhasználva olvas be plusz információkat. A configurationReader egy property fájlból olvas be konfigurácós adatokat, a serviceLSMappingReader pedig egy másik konfigurációs fájlból olvassa fel a service DN – QRadar log forrás név párosítást, amire azért van szükség, mert a QRadar a szabályokban logforrásneveket használ, ami nem feltétlenül ugyanaz. A külső konfigurácós fájlok használta azt teszi lehetővé, hogy ugyanaz az assembly line többféle konfiguárcóval is futtatható legyen, akár egyszerre, az assemly line módosítása nélkül.

Az account információkat tartalmazó entitásokat a Data Flow rész dolgozza fel. A findServiceName egy lookup connector, amire az accountokban tárolt információk miatt van szükség. Ugyanis az ISIM LDAP-ja az egyes accountokhoz a service-t, amihez tartoznak, egy erservice attribútumban tárolja, egy DN formájában.[13] A connector ezekhez a DN-ekhez keresi meg a beszédes nevet, amit a serviceLSMappingReader connector által felolvasott párosítás indexelésére használok fel. Ezzel összeáll minden szükséges információ a feltöltéshez.

Az adat végleges formáját a QRadarConnector állítja össze, ekkor jönnek létre a **QRadar logforrás név - Account név** párosítások. Ezeket az adatokat tölti fel a megfelelő reference data-ban, jelen esetben, mivel kulcs - értékek halmaza formátumot vesznek fel az adatok, ezért egy reference map of sets-be. A futtatás eredménye a 3.6 ábrán található [1]. A felsorolásban láthatók a felhasználói fiókok neveinek egy részhalmaza. Mivel a tesztrendszeren az ISIM felügyelete alatt egy linuxos rendszer található, amely sok technikai account-tal rendelkezik, így a listában főleg ezek láthatók.

[1] A könnyebb megtekinthetőség kedvéért jelen esetben az account-ok service-hez rendelését kihagytam, és egy reference set-be töltöttem be az adatokat. Ugyanis a QRadar a reference set-ek megtekintésére egy átláthatóbb felületet ad, mint a többihez, és így az eredmény könnyebben értelmezhető.



3.5. ábra. Személyhez nem rendelt (árva) felhasználói fiókokat összegyűjtő assembly line.

A ConsoleLogger connector itt egy passzív állapotú connector, amit a assembly line többi eleme használ a sikeres műveletek vagy a hibák naplózására.

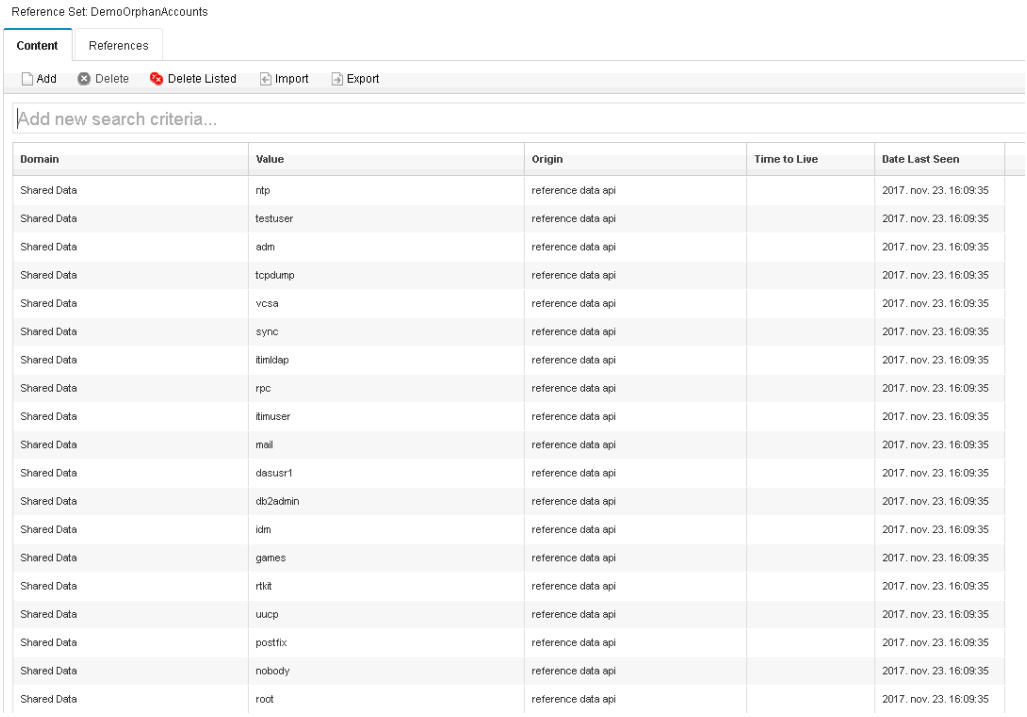
3.3. WAS integráció megvalósítása

Ennél a megoldásnál a cél egy robusztus alkalmazás megalkotása volt, ami jól illeszkedik egy nagyvállalati környezetbe, valamint működés és üzemeltetés szempontjából 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 WebSphere alapú alkalmazás fejlesztésének főbb motivációi:

- A magas rendelkezésre állás, elosztott futtatás, egységes alkalmazásmenedzsment biztosítása.
- Jogosultságkezelés és auditálhatóság megvalósítása.
- Fejlesztői és üzemeltetői feladatok szétválasztása, valamint az üzemeltetési feladatok felhasználói felülettel való támogatása.

A felsorolt funkciók TDI-vel történő megvalósítása valamilyen külső megoldást kívánna, ami növelné az alkalmazás beállítási és karbantartási komplexitását, mivel ezekre is figyelmet kell fordítanunk, ezeket is ellenőriznünk kellene 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,



3.6. ábra. Árva accountok az ISIM rendszerben, melyek nagy része az egyik felügyelt Linuxos rendszerhez tartozik.

menedzselt környezetben futtatva, az adott alkalmazásszerver által nyújtott lehetőségekkel együtt, a legtöbb felsorolt problémára megoldást nyújthat.

A Websphere alapú alkalmazás a fenti területkre 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.

3.3.1. Architektúra tervezése

Mivel az alkalmazás egy projekt részeként valósult meg, többen dolgoztunk rajta, ez 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

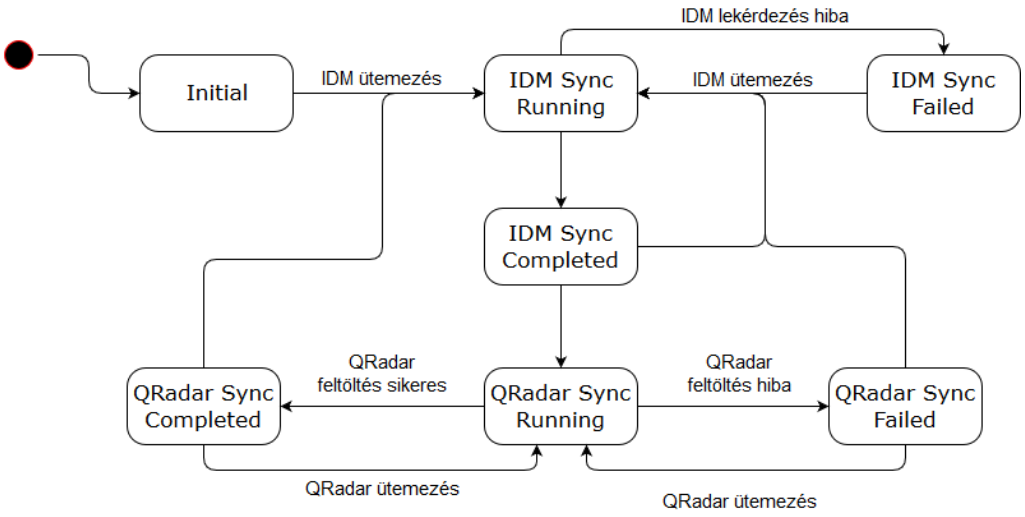
(query) típusokat készítenek a rendszerhez. Ezek a személyek mélyebb ismeretekkel rendelkeznek a rendszerrel kapcsolatban, ám fontos, hogy számukra is egy könnyen használható interfészt biztosítsunk. Emellett fontos, hogy az új lekérdezés típusok könnyen, az alkalmazás minimális – vagy optimális esetben 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 új lekérdezések konfigurálását, azok helyes felparaméterezését.

A könnyű használat mellett fontos szempont volt, hogy az alkalmazás megfelelően robusztus legyen egy vállalati környezet számára is. Fontos cél volt az adatok naprakészen tartása mellett az adatok konziszteciájának megőrzése, valamint a megfelelő performancia elérése, de a túlterhelés elkerülése. Ezen szempontok egy része egymással ellentétes követelményekkel rendelkezik az alkalmazás számára[2], így a tervezésnél kompromisszumokat kellett kötnünk.

A tervezésnél az alábbi döntéseket hoztuk:

- Egységes, moduláris lekérdezésrendszer használata
  - A különböző lekérdezés típusok egy közös interfészt implementálnak, ami miatt a típusok köre könnyen bővíthető utólag is.
  - Minden ténylegesen lefutó lekérdezés egy lekérdezés típusból, és annak a felparaméterezéséből áll. Ezeket egy táblában szerializálva tároljuk, ahonnan az ütemező felolvassa, és futtatja őket.
  - Az egyes lekérdezések felparaméterezése egy webes felületen végezhető, ami támogatja az rendszer alapján a rendelkezésre álló paraméter értékeket.
- Két ütemező, külön időzítéssel, egy az ISIM lekérésekhez, egy a QRadar szinkronizációhoz.
  - Mindkét ütemező periódusideje külön állítható, valamint a két ütemező használata miatt az összes lekérdezésnél külön állítható az IDM lekérdezés, és a QRadar feltöltés között eltelt intervallum nagysága.
- Kétfázisú lekérdezések, több lehetséges állapottal (Lsd.: 3.7 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, majd ha az IDM ütemező elindítja, átkerül IDM_SYNC_RUNNING állapotba.
  - Ha az IDM lekérdezés 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.

[2]Például: az adatok naprakészen tartása indokolná a lekérdezések folyamatosan ismétlődő futtatását, várakozási idő nélkül, a konzisztencia pedig mindig az összes adat feltöltését. Ez azonban lassan lefutó lekérdezések esetén erősen túlterhelné a rendszert, és bizonyos, fontosabb és gyorsabb lekérdezések nem tudnának érvényesülni. A REST API túlterhelése is egy probléma, ami a feleslegesen nagy mennyiségű adat folyamatos feltöltése miatt jelentkezhet.



3.7. ábra. Egy lekérdezés futásának állapotai

  - 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:

3.4.1. Felhasználói fiókok egy adott menedzselt rendszeren

A query célja az ISIM által kezelt egyes service-eken található összes felhasználói fiók összegyűjtése, és egy Map of sets típusú reference data formájában feltöltése.

Felhasználása

Bizonyos biztonsági események tartalmazhatnak információkat nem csak arról, hogy milyen műveletet hajtottak végre, hanem azt is, hogy ki hajtotta ezt végre. Ennek azonosítása általában az adott rendszeren található felhasználói névvel történik. Ennek a query-nek az eredményeként felkerült adatok használatával ellenőrizhető, hogy az eseményben jelzett fiók az ISIM által kezelt-e. Ha nem, az annak jele lehet, hogy a fiókot a biztonsági házirend megkerülésével hozták létre, ami okot adhat biztonsági riasztásra.

Megvalósítás

Szűrés az LDAP-ban található felhasználói fiókokra, aszerint, hogy a kiválasztott service-hez tartoznak-e.

A keresést egy filterrel valósítottam meg, amelynek a kiindulási pontja a felhasználókhoz tartozó fiókokat, valamint az árva fiókokat tartalmazó részfa. Szűrési paraméternek beállítottam az alábbi filter-t:

```
(&(erservice=" + serviceDN + ")(objectclass=eraccountitem))
```

Ezzel kiszűrtem az összes felhasználói fiókot (account) a rendszerben, mivel az ISIM mindegyik accountra hozzáadja objectclass-nak az eraccountitem-et. Majd ezen kívül szűrök még az erservice attribútumra, ami minden accountnál azt tárolja, hogy melyik service-en található az adott account.

3.4.2. Inaktív felhasználókhoz tartozó fiókok

A query célja azoknak a felhasználói fiókoknak az összegyűjtése, amiknek a tulajdonosa inaktív állapotban van, és egy set formájában feltölteni. Ezek olyan felhasználók, akiket az ISIM rendszerben felfüggesztettek.

Felhasználása

Egy felhasználó felfüggesztése általában valami jogvesztéssel járó procedúra során következik be, vagy ideiglenesen, amíg nem biztos például egy jogi eljárás végkimenetele, de az időtartamára szeretnénk a felhasználót kivonni a rendszer által biztosított jogkörökből, vagy véglegesen, ha például elmegy a cégtől, és már nincs szükség az adott személy rekord-ra, valamint account-okra a menedzselt rendszereken. Ha azonban valamiért nem volt sikeres a kivonás, a felhasználói fiók felfüggesztése, vagy egy régebben felfüggesztett account-ot újra aktiválnak valahogy, akkor egy inaktív emberhez egy aktív account tartozik, ami súlyos biztonsági rés lehet egy rendszerben.

Ennek detektálására a SIEM biztonsági riasztást adhat, ha olyan tevékenység történik, ami a kigyűjtött, inaktív felhasználók fiókjaihoz kapcsolódik. Emellett ezek az adatok felhasználhatóak visszamenőleg is, például egy felelősségre vonási eljárásnál, egy adott személy historikus tevékenységének ellenőrzésére.

Megvalósítás

Egy felhasználó aktív/inaktív státuszát az *erpersonstatus* attribútum tárolja, 0 értékként ha aktív, és 1-ként ha inaktív. A felhasználókat ezen attribútum alapján szűröm, majd az eredményként kapott objektumok DN-jével létrehozok egy második filtert, ami az összes DN-t vagy kapcsolatban tartalmazza. Ennek segítségével szűrök az accountokat tartalmazó organizációs egységen.

A első felhasznált filter a következő:

```
(erpersonstatus=1)
```

majd ennek az eredményét felhasználva az alábbi mintára készítek egy másik filtert:

```
(&(objectclass=eraccountitem)(|(owner=$PERSON_DN1$)(owner=$PERSON_DN2$)...))
```

ahol a $PERSON_DN$-el jelöltem a ténylegesen behelyettesítendő, az előző filterből kapott személyekhez tartozó DN-eket.

3.4.3. Felfüggesztési eljárás alatt álló felhasználók fiókjai

Az előző query-nél tárgyalthoz nagyon hasonló eset, de míg az előzőnél a már felfüggesztett, tehát a befejeződött folyamatú emberekre szűrünk, itt azokra, akiknél még zajlik az eljárás. Ez általában elég ritka, mivel a felfüggesztésnek pont az a lényege, hogy gyors, atomi művelet, ami jogvesztéssel jár, de megeshet, hogy például kommunikációs, vagy adapter hiba miatt, vagy akár egy jóváhagyás miatt egy felfüggesztési eljárás nem fejeződik be.[3]

Felhasználása

Az előző esethez hasonlóan itt is jogvesztés elmaradása áll fent, ami biztonsági réssel fenyeget. A SIEM riasztást adhat ezen felhasználói fiókok detektálásakor.

Megvalósítás

Az egyik szűrő feltétel a folyamat állapota, ami a PROCESS táblában található a STATE oszlopban. A szűréshez használt értékek az 'R', 'I', 'S', amik sorrenben a Running, azaz futó, Interrupted, azaz megszakított, és Suspended, azaz felfüggesztett. A másik szűrő feltétel a process típusa, ami meghatározza hogy egy adott process ténylegesen mit hajt végre. Ez a PROCESS táblában a TYPE oszlopban található, és jelen esetben azok a folyamatok a fontosak, amik 'US' típusúak, azaz user suspend típusúak.

Ezek alapján az alábbi két lépcsős folyamatban keresek először az alábbi utasítással:

```
Select REQUESTEE from PROCESS where type = 'US' and state in ('R','I','S')
```

Ez kiválasztja a folyamat célpontját (annak is a DN-jét), az összes olyan folyamatra, ami futó, felfüggesztett, vagy megszakított állapotban van, és a tíupsa a már említett 'US'.

Ezt követően az előző query második lekérésével megegyező formában konkatenációval készítek egy LDAP filtert, ami megadja személyekhez tartozó account-okat.

3.4.4. Törlési folyamat alatt álló felhasználói fiókok

Szintén az előző query-hez hasonló eset, csak itt a törlés alatt álló accountokat keressük. Feltöltés előtt egy transzformációt hajtok végre, amivel az account halmazokat egy QRadar logforráshoz rendelem, ezzel készítve egy map of sets-t.

Felhasználása

Ha egy felhasználói fiók törlése nem történik meg azonnal, hanem például valamilyen kommunikációs hiba miatt megakad, akkor egy ablak nyílik a felhasználó számára, hogy azokat a jogosultságokat használja, amelyek a biztonsági házirend szerint már nem jár neki. Ezeknek a detektálására készíthetünk szabályokat a QRadarban, amik figyelik az ilyen típusú potenciális támadásokat, és jeleznek.

[3] Például egy másik, hasonlóan magas posztban álló manager emberét akarná felfüggeszteni, mert szabálysértésen kapta, de mivel azonos szervezeti szinten állnak, szükség van a jóváhagyásra a másik managertől, aki épp szabadságon van. Ekkor amíg vissza nem érkezik, addig áll a folyamat.

Megvalósítás

A törlési folyamattal kapcsolatos szükséges információkat a PROCESS tábla tárolja. A szükséges folyamatok összegyűjtéséhez először lekérem a táblából azokat a rekordokat, amiknek a STATE értéke 'R','I', vagy 'S', és a TYPE értéke 'AD', azaz account delete. Ezeknek a rekordoknak a SUBJECT mezőjében megtalálható az érintett account neve, valamint a SUBJECT_SERVICE mezőben pedig annak a servicenek a neve, amelyikhez tartozik.

Utolsó lépésként egy külső fájlból betöltött QRadar logforrás - Service name hozzárendeléssel a felolvasott SUBJECT_SERVICE alapján végrehajtom a mappelést, és így előállnak a szükséges map of set-ek.

3.4.5. Árva fiókok

Az árva fiókok 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, vagy olyanok, amiket nem az ISIM-en keresztül, hanem annak megkerülésével vettek fel a menedzselt rendszeren.

Felhasználása

Az árva fiókok jelenléte komoly biztonsági rést jelenthetnek, elsősorban ha például hozzáférnek kritikus rendszerekhez, de megmaradt a alapértelmezett jelszavuk, vagy nem alkalmazták rájuk a megfelelő jelszó házirendeket, ezért fontos lehet, hogy tudjuk detektálni ezeknek a fiókoknak az összes tevékenységét, mert bármelyik fenyegetést jelenthet.

Megvalósítás

Az ISIM az LDAP adatbázisában ezeket a fiókokat egy külön LDAP részfában egységben tárolja, így elég azt lekérnem. Ezt egy LDAP filterrel végzem, az így kinyert account-okat egy előre definiált map alapján hozzárendelem a megfelelő service-hez, és annak a QRadar szerinti log forrás azonosítójához.

3.4.6. Megadott csoportokba tartozó felhasználói fiókok, menedzselt rendszer szerint

A query célja az összes felhasználói fiók összegyűjtése egy menedzselt rendszerről, és az adott rendszeren való csoporttagságuk alapján egy map of setsbe rendezése. A query megvalósításánál külön kihívást jelentett, hogy az egyes accountok más attribútum névvel tárolják a csoportot jelző attribútumot, így azt is külön ki kellett keresni.

Felhasználása

A legtöbb informatikai rendszeren létezik valamilyen felhasználói fiók csoportosítás, ami bizonyos jogköröket definiál a beléjük tartozó fiókokra,s a csoporttagságok alapján összeáll, hogy egy fiók milyen jogokkal rendelkezik az adott rendszeren.

Az egyes jogkörök általában különböző mértékben jelentenek veszélyt, ha nem megfelelő kezekbe kerülnek, így ha megvannak az adott csoportba tartozó fiókok nevei, akkor könnyen létrehozhatunk szabályokat, amiket a súlyosság szerint testreszabhatunk az alapján, hogy milyen jogokkal bírnak a fiókok, és ez milyen mértékben jelent veszélyt.

Ilyen szabály lehet például ha a privilegizált felhasználók munkaidőn kívül lépnek be, ami jelentheti akár a fiókjuk kompromittálódását is, ami széles jogkörökkel lényegesen nagyobb veszélyt jelenthet, mintha például egy végfelhasználó fiókja sérülne.

Megvalósítás

Az ISIM az alábbi architektúrát használja a service-ek és az accountok kezelésére: Minden service rendelkezik egy service profile-al, ami leírja, hogy egy service típus milyen attribútumokal rendelkezik, plusz egyéb technikai információkat tárol. Ez egy attribútum formájában adja meg, hogy a rajta található account-ok milyen típusúak, azaz milyen account profile definiálja őket. Az egyes service-ekhez tartozó accountokon definiálva van egy csoportosítás, amit group-oknak nevezünk, és egy, az accounton található, többértékű attribútum értéke tartalmazza. Ennek az attribútumnak a neve azonban minden account típusra más, és ezt a nevet az egyes accountokhoz tartozó service profile-ok tartalmazzák.

Ezért a megvalósításhoz egy LDAP szűréssel lekérdezem a kijelölt service-hez tartozó account-okat és az összes attribútumukat. Majd a service típusán keresztül, a service profileból lekérem a rajta található account-ok csoport attribútumának nevét. Az accountokat ezután összerendelem a csoportok szerint, és elkészítem a map of sets-et.

3.5. QRadar esemény küldő és feldolgozó 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 nem megengedett karakterek (például pontosvesszők) backslash-el történő prefixálása az attribútumok értékeiben.

A generálási folyamat utolsó lépése a sorok felküldése 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 formátumot használjon a logger az üzenet előállításához.

A Log4J által biztosított syslog logger azonban implementációjából fakadóan nem volt megfelelő a QRadar irányába syslog üzenetek küldésére, mivel az 1019 byte-nál hosszabb üzeneteket több üzenetté tördelte. Emiatt a QRadar a töredékeket külön eseményekként kezelte. A problémára több megoldás is kínálkozott, például egy máshogyan implementált logger használata, vagy egy saját fejlesztése, de végül a dependenciák minimalizálása, és felesleges munka elkerülése végett a QRadar egy speciális működési módját használtuk, az UDP multiline syslog-ot.[**?**] Ennél a módnál a QRadar az ehhez definiált forrásoktól olyan üzeneteket vár, amik tartalmaznak egy egyedi azonosítót. Az azonosító felismerését egy reguláris kifejezéssel végzi a QRadar, és az értéke bármilyen karakterlánc lehet. Ennek az azonosítónak a lényege, hogy ezen keresztül azonosítja a QRadar az összetartozó tördelékeket, és a több ilyen azonosítóval rendelkező üzeneteket összefűzi egy eseménnyé.

Az UDP multiline syslog mód támogatására átalakítottam a TDI assembly line-t úgy, hogy a tördelési műveletet saját magam végzem, megadott szabályok szerint. A tényleges payload-ot maximum 800 karakter méretű szeletekre tördelem, figyelve az attribútum határokat jelölő pontosvesszőkre. Így, a generált 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

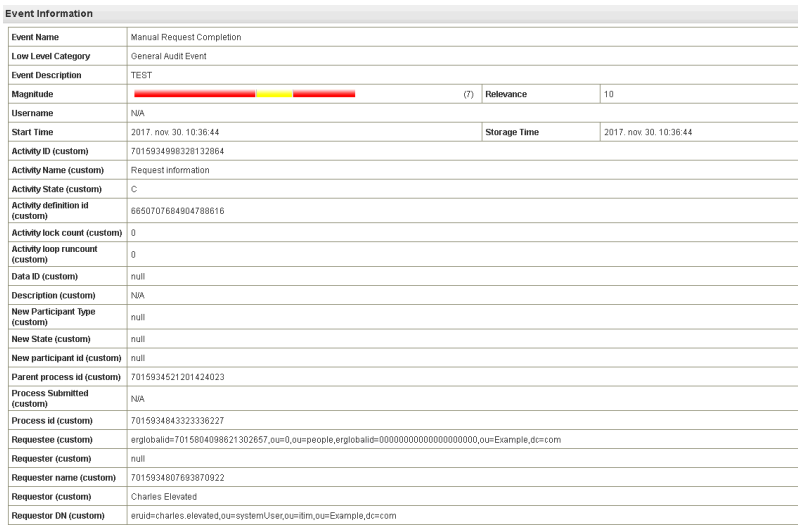
A QRadar oldali fogadást a már említett UDP multiline syslog segítségével biztosítottam. Ezt egy eseményforrás felkonfigurálásánál kell megadni, és annyiban különbözik az átlagos syslog protokollon érkező üzenetektől, hogy az 514-es port helyett az 517-est használja, valamint a beérkező üzeneteknek rendelkeznie kell a már tárgyalt egyedi 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. Az egyedi azonosítók feldolgozásához egy *msg_uuid* mezőt keres a property feldolgozó, az alábbi reguláris kifejezés segítségével:

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

Az összeállított üzenetek eseménnyé alakításához definiáltam egy saját esemény típust, amelyet egy egyedi 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 reguláris kifejezést. Mivel az adatok normalizálásánál egy egységes szisztémát követtem, ezért az összes feldolgozó reguláris kifejezés az alábbi sémára épül:

```
  ATTRIBÚTUM_NÉV=(.*?[^\\]);
```



Event Information

Event Name	Manual Request Completion		
Low Level Category	General Audit Event		
Event Description	TEST		
Magnitude	(7)	Relevance	10
Username	N/A		
Start Time	2017. nov. 30. 10:36:44	Storage Time	2017. nov. 30. 10:36:44
Activity ID (custom)	7015934998328132864		
Activity Name (custom)	Request information		
Activity State (custom)	C		
Activity definition id (custom)	6650707684904788616		
Activity lock count (custom)	0		
Activity loop runcount (custom)	0		
Data ID (custom)	null		
Description (custom)	N/A		
New Participant Type (custom)	null		
New State (custom)	null		
New participant id (custom)	null		
Parent process id (custom)	7015934521201424023		
Process Submitted (custom)	N/A		
Process id (custom)	7015934843323336227		
Requestee (custom)	erglobalid=7015804098621302657,ou=0,ou=people,erglobalid=00000000000000000000,ou=Example,dc=com		
Requester (custom)	null		
Requester name (custom)	7015934807693870922		
Requestor (custom)	Charles Elevated		
Requestor DN (custom)	eruid=charles.elevated,ou=systemUser,ou=itim,ou=Example,dc=com		

3.8. ábra

Ennél a reguláris kifejezésnél az attribútum értéke pontosan az első találati csoportban érhető el, amibe minden karakter beletartozik, ami az attribútum nevet követő egyenlőség jel, és az első olyan pontosvessző között található, ami előtt nem backslash áll. Ezen kifejezésekkel sikerült feldolgoznom az összes lehetséges beérkező property-t, amiből a fontosabbak listája megtekinthető a 3.1 táblázatban, valamint az összes a függelék **??** táblázatában. Az attribútum nevek felépítése a következő: az első karakter, ami a _ előtt áll a táblát jelöli, amelyből származik (A: ACTIVITY, P: PROCESS, PL: PROCESSLOG), a _ utáni rész pedig az adott táblában található oszlop neve. A feldolgozott esemény a felolvasott property-kkel a QRadar felületén a 3.8 ábrán látható.

3.1. táblázat. Szemelvény a feldolgozott esemény property-kből.

Attribútum név	Attribútum funkciója
A_COMPLETED	Az activity befejezésének időpontja
A_DESCRIPTION	Maximum 300 karakteres, beszédes leírása az activity-nek
A_NAME	Acvitiy beszédes neve. Segíti az activity azonosítását
A_RESULT_SUMMARY	Két karakteres leírása a végeredménynek. Ezzel a mezővel szűrhetünk az activity-k végeredményére.
A_TYPE	Az activity típusa, egy karakterként tárolva. Pl: kézi (M), alkalmazás (A), subprocess (S)
P_NAME	A process neve.
P_REQUESTEE_NAME	Annak a neve, aki számára a process végrehajtásra kerül.
P_REQUESTER_NAME	A process kérelmezőjének a neve.
P_RESULT_SUMMARY	A process végeredménye, két karakterként reprezentálva. Pl: jóváhagyva (AA), elutasítva (AR), sikertelen(SF).
P_TYPE	2 karakteres kódja a process típusának. Pl: új felhasználó (UA), jelszóváltoztatás (AP).
PL_REQUESTOR	Igénylő neve felhasználókkal kapcsolatos eseményeknél

Az új esemény típus létrehozásának utolsó lépése az összerendelés definiálása, ami megadja, hogy egy adott, feldolgozott eseményt milyen attribútumok alapján, milyen típusba sorolunk. Ehhez kettő értéket használunk, amiket a QRadar minden eseménynél megpróbál feldolgozni. A két érték az Event ID, és az Event Category. Az Event ID az elsődleges klasszifikációt biztosítja. Amennyiben a feldolgozott érték megegyezik az egyik

összerendelésben megadott értékkel, az egyértelműen beazonosítja az adott esemény típusát az összerendelés szerint. Az Event Category extra metadatokkal való kategorizálást tesz lehetővé az események között, olyan információkkal kibővítve az esemény feldolgozását mint az olvasható név, leírás, súlyosság, vagy alacsony- és magas szintű kategória.

4. fejezet

Megoldások telepítése, használata

A telepítés és a használat leírásának során feltételezem, hogy már fel van telepítve, és rendelkezésre áll a felhasználni kívánt QRadar, az ISIM, a TDI, valamint a megfelelő WebSphere. A fejlesztés és a tesztelés során felhasznált applikációk verziói a következők:

- QRadar 7.2.8
- ISIM 6.0
- TDI 7.1.1
- WebSphere liberty 17.0.0.2

4.1. TDI alapú integrációs modul

A megoldás általam készített része egy wrapper, a hozzá tartozó segédosztályokkal, valamint egy TDI connector implementáció, ami a wrapperre épül. A wrapper önmagában is felhasználható más projektekben. A connector egy .jar formájában használható fel, amit a megfelelő TDI installáció $INSTALL_DIR$/jars/connectors mappájába bemásolva használhatunk fel.

4.1.1. Dependenciák

A wrapper a HTTP hívások bonyolítására az Apache Wink[12] framework-öt használja, így ezen a library-n kívül szükség van ennek a dependenciáira is, mint például a J2EE bővítményekre.

A wrapper ezen kívül logolásra az SLF4J könyvtárat használja.[11]

Mivel a QRadar irányú kapcsolat SSL titkosított és ehhez 2048 bites kulcsot használ, valamint DHE kulcscserét, ezért bizonyos java verziók esetén hibák léphetnek fel az SSL Handshake folyamán [1] [2]. Ajánlott a Java 1.7-es verzióját használni, amiben ezek már javítva vannak.

4.1.2. Telepítés

Fordítás

Ha csak forráskód áll rendelkezésre, akkor szükséges a projekt lefordítása, és egy .jar fájlba csomagolása. A fordításhoz szükségesek a fent felsorolt dependenciák elérhetővé tétele,

[1] Megoldások 1.6-os Java esetén: https://stackoverflow.com/questions/6851461/java-why-does-ssl-handshake-give-could-not-generate-dh-keypair-exception

[2] Megoldás 1.6-os Java és IBM WebSphere használata esetén: https://developer.ibm.com/answers/questions/209245/ssl-exception-error-in-wesphere-application-server.html

valamint kettő, a TDI által biztosított .jar fájl: *miserver.jar* és *miconfig.jar*. Ezek megtalálhatók a *$TDI_INSTALL$/jars/common* mappában. Az elkészült jar fájl tartalmazza a fordított .class fájlokat a megfelelő mappa struktúrában, valamint a gyökérkönyvtárban a 3.2.1. fejezetben leírtaknak megfelelően elkészített tdi.xml-t.

TDI oldali konfiguráció

Ahhoz, hogy a TDI használni tudja a connector-t, az elkészített .jar fájlt be kell másolni a *$TDI_INSTALL$/jars/connectors* mappába, valamint a szükséges dependenciákat a *$TDI_INSTALL$/jars/3rdparty/others* mappába.

4.1.3. Kommunikációs beállítások

SSL titkosítás beállítása

Mivel a QRadar SSL titkosítást használ, amihez egy self-signed certificate-el rendelkezik, ezt a certificate-et hozzá kell adni a TDI által használt SSL keystore-okhoz. Erre a feladatra ajánlott a TDI-al együtt érkező Java installáció által biztosított Ikeyman[3] grafikus alkalmazás használata. A két keystore, amihez a kulcsokat hozzá kell adni a *$TDI_install_dir$/testserver.jks* valamint a *$TDI_install_dir$/serverapi/testadmin.jks*.[10]

Engedélyezett alkalmazás felvétele a QRadar oldalán

A TDI connector - QRadar irányú authentikáció biztosításához szükséges egy QRadar API kulcs. Ezt a QRadar webes admin felületén, az *Admin -> User Management -> Authorized Services* menüpont alatt található. Itt egy új rekord felvétele és felkonfigurálása után, az Authentication Token mezőben található token-t használva a connector felkonfigurálásához létrehozható a kapcsolat a REST API-val.

Amennyiben biztonsági szempontból szükséges, beállítható, hogy a kulcsot használó applikációk milyen csoport tagsággal rendelkezzenek, valamint a hozzáférés elévülési ideje. A csoport tagság definiálja azt, hogy pontosan milyen típusú operációkhoz fog hozzáférni az alkalmazás, ezért fontos, hogy a connector által használt tokenhez tartozó csoport tudja írni és olvasni is a reference data-kat.

4.1.4. Használat

Az előzetes konfigurációs lépések elvégzése után a connector elérhető a TDI grafikus szerkesztőjén keresztül, és szabadon felhasználható assembly line-okban a többi, beépített connector-nak megfelelő módon. Egy ilyen felhasználásra mutat példát a 3.2.2 fejezetben leírt példa.

4.2. Websphere alapú alkalmazás

4.2.1. Dependenciák

Az alkalmazás egy Java EE webalkalmazás, amely WebSphere alkalmazásszerverre lett tervezve, azon belül pedig a WebSphere Liberty változatára. Mivel csak ezen lett tesztelve, ezért más alkalmazásszerver nem támogatott, de apróbb változtatásokkal valószínűleg más szerveren is futtatható, de a leírásnak nem célja egy mindenre kiterjedő útmutatót adni.

[3] Elérési út: *$TDI_install_dir$/jvm/jre/bin*

A teszteléshez 17.0.0.2-es Liberty installációt használtam, valamint fontos, hogy a Liberty szerver támogassa, és legyenek telepítve az alábbi beépülők: webProfile-7.0, localConnector-1.0, jsp-2.3, jca-1.7, concurrent-1.0, appSecurity-2.0, servlet-3.1, passwordUtilities-1.0, distributedMap-1.0.

Az ütemezéshez az alkalmazás a commonj [**?**] könyvtárat használja, így ezt is elérhetővé kell tenni a szerver számára.

Mivel ez az alkalmazás is az általam készített wrappert használja a QRadarral való kommunikációra, ezért a 4.1.1-ben leírtaknak megfelelően annak a dependenciáira is szükség van, nevezetesen az SLF4J könyvtárra, és az Apache Wink könyvtár megfelelő részeire.

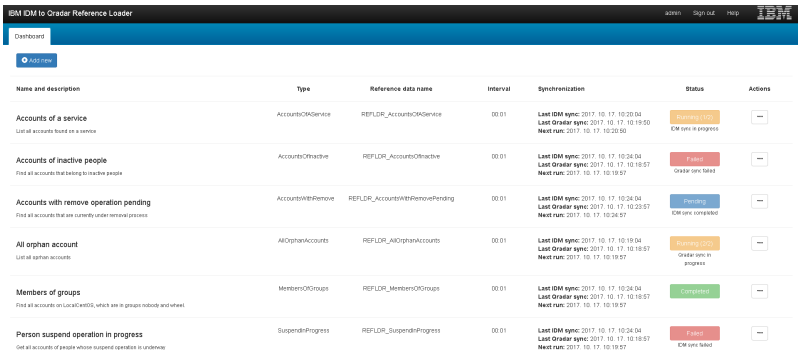
Az alkalmazás működéséhez továbbá szükség van a megfelelő verziójú DB2-es Java driverre[4] és licenszre, amiket az alkalmazás az ISIM adatbázisaihoz, valamint a saját adatbázisaihoz való kapcsolódásra használ. Ezeket egy library formájában fel kell venni a Liberty szerver definíciós XML-jébe, továbbá a megfelelő adatbázisokat a hozzájuk tartozó authentikációs adatokkal dataSource-ként.[2]

A fent említetteken kívül, valamint az általános konfigurációs értékeken kívül szükséges a szerverhez tartozó server.xml állományába felvenni / módosítani a következő konfigurációs értékeket:

- distributedMap[4]: Az alkalmazás által használt DynaCache definíciója
- managedScheduledExecutorService[1]: Az alkalmazás által használt ütemező definíciója

4.2.2. Telepítés

A dependenciák elérhetővé tétele, és a szerver megfelelő felkonfigurációja után az alkalmazás telepítésére két mód is rendelkezésre áll. Lehetőség van a lefordított alkalmazás csomagolt formában való elhelyezésére a Liberty által rendszeresen ellenőrzött *dropins* mappában, vagy az alkalmazás konfigurációjában elhelyezni egy rekordot, ami az telepítendő applikáció elérési útját tartalmazza. [3]



4.1. ábra. A WebSphere alapú integrációs megoldás főképernyője, az összes lehetséges query állapottal.

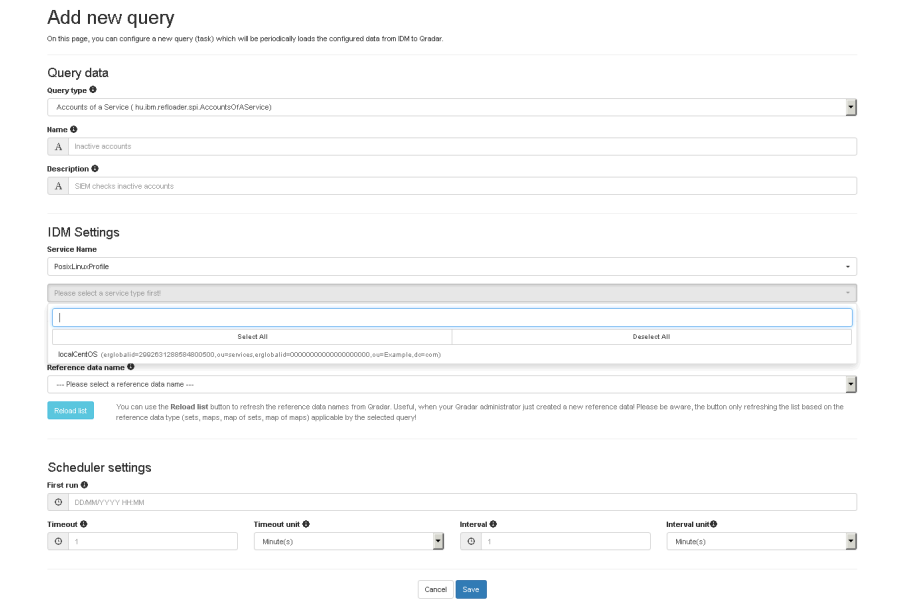
4.2.3. Használat

Az alkalmazás egy Bootstrap alapú felhasználói felületről érhető el. A konfigurációban megadott bejelentkezési adatok segítségével belépve egy központi oldal fogad, amin lát-

[4]A megfelelő verzió jelen esetben azt jelenti, hogy a felhasznált driverek az adott, felhasznált DB2 instanciának megfelelő driverek kell hogy legyenek

hatók a felkonfigurált lekérdezések, azok állapotai, valamint az ütemezési információik. A felületen megjelenített mezők jelentése a következő:

- Name and description: A szinkronizációs feladat neve és leírása, amit annak felvételekor adhatunk meg.
- Type: A lekérdezés típusa
- Reference data name: A lekérdezés által kezelt reference data neve.
- Interval: A lekérdezés futtatási gyakorisága.
- Synchronization: Információk a lekérdezés ütemezésével kapcsolatban.
- Status: Információ a lekérdezés aktuális állapotáról.
- Actions: Lenyíló menü a lekérdezéssel kapcsolatos máveletek indítására.



4.2. ábra. Az új query-k hozzáadását támogató felület

Egy felvett lekérdezés típuson 5 művelet típus hajtható végre, amik a következők:

- Synchronize now: Az adott lekérdezést azonnal hozzáadja az ütemezési sorhoz, attól függetlenül, hogy milyen ütemezési paraméterekkel rendelkezik, és milyen állapotban van.
- Reload: Frissíti a lekérdezés állapotát és a hozzá kapcoslódó információkat a felületen.
- View content: Egy felugró ablakban megmutatja a feltöltött reference data állapotát, az alkalmazás által tárolt lokális állapot alapján.
- Preview: Az ISIM-ből lekérdezett adatokat mutatja olyan lekérdezéseknél, amelyek még nem töltötték fel az eredményeket a QRadarnak.
- Delete: Törli a lekérdezést.

Új query felvételéhez a bal felül található "Add query" gomb használható, ami egy új oldalra, a 4.2. ábrán láthatóra irányít át. Ezen az oldalon megadhatjuk a query típusát, ami alapján az oldal kezelőfelülete megváltozik, és a query által használható paraméterek konfigurlását lehetővé tevő mezők lesznek elérhetők rajta. A mezők, ahol releváns, segítik a kitöltést, és felkínáljak a lehetséges inputokat. A felkonfiruált lekérdezést elmentve az bekerül az ütemezési sorba, és végrehajtódik, ha rá kerül a sor.

5. fejezet

Összefoglalás

Jelen dolgozat kiindulási problémáját az adta, hogy az IT biztonság területén használt SIEM, az IBM Security QRadar felhasználási lehetőségeit kibővítsük egy Idm, az IBM Security Identity Manager segítségével. A konkrét cél az volt, hogy a két termék közti integrációt egyedi megoldásokkal támogassuk, és az ISIM-ben elérhető, felhasználókkal kapcsolatos adatokat és folyamat információkat felkínáljuk a QRadar számára. Ezek segítségével elsősorban potenciálisan veszélyes tevékenységek detektálhatók az ISIM által kezelt menedzselt rendszereken, valamint az ISIM folyamatain belül.

Az integráció megvalósításához két módszert definiáltunk:

- Releváns felhasználói adatok összegyűjtése és feltöltése a QRadar szabályrendszere számára.
- ISIM folyamat információiból események generálása, és ezek elküldése a QRadar számára.

A munkám előtt rendelkezésre állt már egy QRadar bővítmény, ami képes kezelni az ISIM-ből érkező audit események egy részhalmazát, ám ez nem volt felkészítve azokra a folyamat információkat tartalmazó események feldolgozására, amiket az általam készített megoldás kezel. Az általam megvalósított rendszer az IBM Tivoli Directory Integrator-t használja a betöltés, a generálás és a felküldés végrehajtására, és a felhasznált adatokat a konkrét ISIM folyamatokból gyűjti össze. Ezekből egy syslog sort generálnak, amit a QRadar egy általam definiált esemény típussá dolgoz fel, ami miatt ezek további feldolgozása már a QRadarban elérhető módszerekkel elvégezhető.

A munkám előtt ISIM felhasználói adatainak integrációjára nem volt még kidolgozott megoldás, így ez egy teljesen új funkcionalitást biztosít. A felhasználói adatok feltöltéséhez a QRadar reference data adattárolóit használom, szintén a QRadar által biztosított REST API-on keresztül. Ez a feltöltési folyamat egy biztonságos, TLS titkosított csatornán keresztültörténik, aminél az applikációt egy API token hitelesíti a QRadar felé. A tényleges feltöltéshez készítettem egy Java wrapper osztályt, ami egy egyszerű, és könnyen használható interfészt biztosít a Qradar összes reference data típusának használatához, valamint könnyen kezelhetővé teszi az egyes plusz funkciók használatát is.

A feltöltő alkalmazás konkrét megvalósításához két architektúrát definiáltunk és valósítottunk meg, az egyik az IBM Tivoli Directory Integrator nevű adatintegrációs eszközt és framework-öt használja, a másik pedig egy Java webalkalmazás, ami IBM WebSphere alkalmazásszerveren fut. A TDI alapú megoldásnál én készítettem el az integrációhoz szükséges connectort a QRadar felé, valamint a minta integrációs feladatok egy részét. Mivel a fejlesztés egy projekt keretében zajlott, és a TDI alapú és a WebSphere alapú megoldás párhuzamosan készült, ezért a Websphere alapú fejlesztésében csak a tervezési fázisban, a tesztelési fázisban, valamint az integrációs feladatok fejlesztésében vettem részt.

A fejlesztési folyamat eredményeként létrejött három megoldás, amik támogatják a leírt adatintegrációs feladatokat mind alkalmankénti futtatás, mint ütemezett, rendszeres futtatás esetén.

Irodalomjegyzék

[1] IBM Knowledgebase: Configuring managed scheduled executors. https://www.ibm.com/support/knowledgecenter/en/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ae/twlp_config_scheduledexecutor.html.

[2] IBM Knowledgebase: Configuring relational database connectivity in liberty. https://www.ibm.com/support/knowledgecenter/en/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ae/twlp_dep_configuring_ds.html.

[3] IBM Knowledgebase: Deploying applications in liberty. https://www.ibm.com/support/knowledgecenter/en/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/twlp_dep.html.

[4] IBM Knowledgebase: Distributed map interface for dynamic caching. https://www.ibm.com/support/knowledgecenter/en/SSEQTP_8.5.5/com.ibm.websphere.wlp.doc/ae/rwlp_feature_distributedMap-1.0.html.

[5] IBM Knowledgebase: The entry object. https://www.ibm.com/support/knowledgecenter/en/SSCQGF_7.1.0/com.ibm.IBMDI.doc_7.1/entryobject.htm.

[6] IBM Knowledgebase: Hivatalos útmutó tdi connector fejlesztéshez. https://www.ibm.com/support/knowledgecenter/en/SSCQGF_7.1.0/com.ibm.IBMDI.doc_7.1/referenceguide155.htm.

[7] IBM Knowledgebase: Ibm qradar technikai prezentációja. https://www.ibm.com/us-en/marketplace/ibm-qradar-siem.

[8] IBM Knowledgebase: Ibm qradar technikai prezentációja. https://www.ibm.com/support/knowledgecenter/en/SSCQGF_7.1.1/com.ibm.IBMDI.doc_7.1.1/pdguide09.htm.

[9] IBM Knowledgebase: Ibm security identity manager technikai dokumentációja. https://www.ibm.com/support/knowledgecenter/en/SSRMWJ_6.0.0/com.ibm.isim.doc_6.0/ic-homepage.htm.

[10] IBM Knowledgebase: Qradar ssl konfigurációja. https://www.ibm.com/support/knowledgecenter/en/SSCQG_7.1.1/com.ibm.IBMDI.do_7.1.1/adminguide36.htm.

[11] Quality Open Software: Slf4j dokumentáció. https://www.ibm.com/support/knowledgecenter/en/SSCQG_7.1.1/com.ibm.IBMDI.do_7.1.1/adminguide36.htm.

[12] Apache software foundation: Apache wink. https://cwiki.apache.org/confluence/display/WINK/Index.

[13] Unknown: Ldap dn, and rdn definition. https://www.ldap.com/ldap-dns-and-rdns.

Függelék

F.1. Feldolgozott attribútumok

F.1. táblázat. Az ACTIVITY táblában található oszlopokból készített attribútumok

Attribútum név	Attribútum funkciója
A_ACTIVITY_INDEX	Activity számláló azonosítója, ha egy ciklusban szerepel
A_COMPLETED	Az activity befejezésének időpontja
A_DESCRIPTION	Maximum 300 karakteres, beszédes leírása az activity-nek
A_ID	Activity egyedi azonosítója
A_LASTMODIFIED	Az activity utolsó módosításának dátuma
A_LOCK_COUNT	A függőben lévő feladatok az activity-n
A_LOOP_COUNT	Iteráció specifikus érték. Az eltelt iterációk száma.
A_LOOP_RUNCOUNT	Aszinkron ciklusos activity-k értéke. A hátralévő iterációk száma
A_NAME	Acvitiy beszédes neve. Segíti az activity azonosítását
A_PRIORITY	Activity prioritása.
A_PROCESS_ID	Az adott activity-hez tartozó process egyedi azonosítója. Ezen keresztül köthető össze a process táblával
A_RESULT_DETAIL	Végeredmény beszédes leírása
A_RESULT_SUMMARY	Két karakteres leírása a végeredménynek. Ezzel a mezővel szűrhetünk az activity-k végeredményére.
A_RETRY_COUNT	Próbálkozások száma az activity végrehajtására
A_SHORT_DETAIL	Rövid, szöveges leírása az activity végeredményének
A_STARTED	Az activity indításának időpontja
A_STATE	Az activity aktuális állapota. Ezzel szűrhetünk pl a futó, a megállított, a befejezett, stb állapotokra.
A_SUBPROCESS_ID	Az activity-hez kapcsolódó subprocess azonosítója.
A_SUBTYPE	Kézi activity-knél altípus, pl ezzel különböztethető meg a jóváhagyás/elutasítási, és az információ adási kérés.
A_TYPE	Az activity típusa, egy karakterként tárolva. Pl: kézi (M), alkalmazás (A), subprocess (S)

F.2. táblázat. A PROCESS táblában található oszlopokból készített attribútumok

Attribútum név	Attribútum funkciója
P_COMMENTS	Megjegyzések a process-hez
P_COMPLETED	A process befejezésének ideje.
P_DEFINITION_ID	A process definíciójának azonosítója.
P_DESCRIPTION	Process szöveges leírása.
P_ID	Adott process azonosítója.
P_LASTMODIFIED	A process utolsó változásának időpontja.
P_NAME	A process neve.
P_NOTIFY	Megadja, hogy ki értesüljön a process befejeztekor.
P_PARENT_ACTIVITY_ID	Az adott processhez tartozó szülő activity azonosítója, ha van neki.
P_PARENT_ID	A szülő process azonosítója, ha van neki.
P_PRIORITY	A process prioritása.
P_REQUESTEE	Annak a DN-je, aki számára a process végrehajtásra kerül.
P_REQUESTEE_NAME	Annak a neve, aki számára a process végrehajtásra kerül.
P_REQUESTER	A process kérelmezőjének a DN-je.
P_REQUESTER_NAME	A process kérelmezőjének a neve.
P_REQUESTER_TYPE	A process kérelmezőjének a típusa. Pl végfelhasználó (U), ISIM System (P), Workflow (S).
P_RESULT_DETAIL	Részletes információk a process eredményéről.
P_RESULT_SUMMARY	A process végeredménye, két karakterként reprezentálva. Pl: jóváhagyva (AA), elutasítva (AR), sikertelen(SF).
P_ROOT_PROCESS_ID	A legmagasabb szinten álló szülő process azonosítója.
P_SCHEDULED	A process ütemezett indítási időpontja.
P_SHORT_DETAIL	Rövid összefoglaló a process eredményéről.
P_STARTED	A process indításának ideje.
P_STATE	A process aktuális állapota. Pl: futó (R), kész (C), megállított (A)
P_SUBJECT	A process alanya.
P_SUBJECT_ACCESS_ID	A kérelmezett hozzáférés DN-je. (Ha a process hozzáférési kérelem volt)
P_SUBJECT_ACCESS_NAME	A kérelmezett hozzáférés neve. (Ha a process hozzáférési kérelem volt)
P_SUBJECT_PROFILE	Az alany LDAP szintű típusa.
P_SUBJECT_SERVICE	Az a service, amihez az alany tartozik.
P_SUBMITTED	A process létrehozásának időpontja.
P_TENANT	A kérelmezőhöz tartozó tartomány DN-je.
P_TYPE	2 karakteres kódja a process típusának. Pl: új felhasználó (UA), jelszóváltoztatás (AP).

F.3. táblázat. Az összes feldolgozott property-t tartalmazó táblázat.

Attribútum név	Attribútum funkciója
PL_ACTIVITY_ID	A processlog eseményhez tartozó activity azonosítója.
PL_CREATED	A processlog esemény létrehozásának ideje.
PL_DATA_ID	Az adat azonosítója adat változás esetén.
PL_EVENTTYPE	Az adott log esemény típus kódja. Pl: activity létrehozva (AC), activity állapot változott (AS)
PL_ID	Az adott process log esemény azonosítója.
PL_NEW_DATA	Új adatok az adatváltozási esemény során, ha azok mérete nagyobb mint egy limit.
PL_NEW_PARTICIPANT_ID	Az új résztvevő azonosítója feladat delegáció esetén.
PL_NEW_PARTICIPANT_TYPE	Az új résztvevő típusa feladat delegáció esetén.
PL_NEW_STATE	Új típus, egy típus változási esemény során.
PL_OLD_PARTICIPANT_ID	Feladat delegációs esemény esetén a régi résztvevő azonosítója.
PL_OLD_PARTICIPANT_TYPE	Feladat delegációs esemény esetén a régi résztvevő típusa.
PL_OLD_STATE	Régi típus egy típus változási esemény során.
PL_PROCESS_ID	A process azonosítója, amihez a process log esemény tartozik.
PL_REQUESTOR	Igénylő neve felhasználókkal kapcsolatos eseményeknél
PL_REQUESTOR_DN	Igénylő DN-je felhasználókkal kapcsolatos eseményeknél
PL_REQUESTOR_TYPE	Igénylő típusa felhasználókkal kapcsolatos eseményeknél
PL_SMALL_NEW_DATA	Új adatok az adatváltozási esemény során, ha azok mérete kisebb mind egy limit.