Általános információk, a diplomaterv szerkezete

A diplomaterv szerkezete a BME Villamosmérnöki és Informatikai Karán:

1. Diplomaterv feladatkiírás
2. Címoldal
3. Tartalomjegyzék
4. A diplomatervező nyilatkozata az önálló munkáról és az elektronikus adatok kezeléséről
5. Tartalmi összefoglaló magyarul és angolul
6. Bevezetés: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása
7. A feladatkiírás pontosítása és részletes elemzése
8. Előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések
9. A tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása
10. A megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek
11. Esetleges köszönetnyilvánítások
12. Részletes és pontos irodalomjegyzék
13. Függelék(ek)

Felhasználható a következő oldaltól kezdődő LaTeXdiplomatervsablon dokumentum tartalma.

A diplomaterv szabványos méretű A4-es lapokra kerüljön. Az oldalak tükörmargóval készüljenek (mindenhol 2,5 cm, baloldalon 1 cm-es kötéssel). Az alapértelmezett betűkészlet a 12 pontos Times New Roman, másfeles sorközzel, de ettől kismértékben el lehet térni, ill. más betűtípus használata is megengedett.

Minden oldalon – az első négy szerkezeti elem kivételével – szerepelnie kell az oldalszámnak.

A fejezeteket decimális beosztással kell ellátni. Az ábrákat a megfelelő helyre be kell illeszteni, fejezetenként decimális számmal és kifejező címmel kell ellátni. A fejezeteket decimális aláosztással számozzuk, maximálisan 3 aláosztás mélységben (pl. 2.3.4.1.). Az ábrákat, táblázatokat és képleteket célszerű fejezetenként külön számozni (pl. 2.4. ábra, 4.2. táblázat vagy képletnél (3.2)). A fejezetcímeket igazítsuk balra, a normál szövegnél viszont használjunk sorkiegyenlítést. Az ábrákat, táblázatokat és a hozzájuk tartozó címet igazítsuk középre. A cím a jelölt rész alatt helyezkedjen el.

A képeket lehetőleg rajzoló programmal készítsék el, az egyenleteket egyenlet-szerkesztő segítségével írják le (A LaTeX ehhez kézenfekvő megoldásokat nyújt).

Az irodalomjegyzék szövegközi hivatkozása történhet sorszámozva (ez a preferált megoldás) vagy a Harvard-rendszerben (a szerző és az évszám megadásával). A teljes lista névsor szerinti sorrendben a szöveg végén szerepeljen (sorszámozott irodalmi hivatkozások esetén hivatkozási sorrendben). A szakirodalmi források címeit azonban mindig az eredeti nyelven kell megadni, esetleg zárójelben a fordítással. A listában szereplő valamennyi publikációra hivatkozni kell a szövegben (a LaTeX-sablon a BibTeX segítségével mindezt automatikusan kezeli). Minden publikáció a szerzők után a következő adatok szerepelnek: folyóirat cikkeknél a pontos cím, a folyóirat címe, évfolyam, szám, oldalszám tól-ig. A folyóiratok címét csak akkor rövidítsük, ha azok nagyon közismertek vagy nagyon hosszúak. Internetes hivatkozások megadásakor fontos, hogy az elérési út előtt megadjuk az oldal tulajdonosát és tartalmát (mivel a link egy idő után akár elérhetetlenné is válhat), valamint az elérés időpontját.

Fontos:

- A szakdolgozatkészítő / diplomatervező nyilatkozata (a jelen sablonban szereplő szövegtartalommal) kötelező előírás, Karunkon ennek hiányában a szakdolgozat/diplomaterv nem bírálható és nem védhető!
- Mind a dolgozat, mind a melléklet maximálisan 15 MB méretű lehet!

Jó munkát, sikeres szakdolgozatkészítést, ill. diplomatervezést kívánunk!

FELADATKIÍRÁS

A feladatkiírást a tanszéki adminisztrációban lehet átvenni, és a leadott munkába eredeti, tanszéki pecséttel ellátott és a tanszékvezető által aláírt lapot kell belefűzni (ezen oldal *helyett*, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell beleszerkeszteni ezt a feladatkiírást.

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. május 11.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Bulla Ádám*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot/ diplomatervet (nem kívánt törlendő) 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. május 11.

Bulla Ádám
hallgató

Kivonat

Jelen dokumentum egy diplomaterv sablon, amely formai keretet ad a BME Villamosmérnöki és Informatikai Karán végző hallgatók által elkészítendő szakdolgozatnak és diplomatervnek. A sablon használata opcionális. Ez a sablon LaTeX alapú, a *TeXLive* TeX-implementációval és a PDF-LaTeX fordítóval működőképes.

Abstract

This document is a LaTeX-based skeleton for BSc/MSc theses of students at the Electrical Engineering and Informatics Faculty, Budapest University of Technology and Economics. The usage of this skeleton is optional. It has been tested with the *TeXLive* TeX implementation, and it requires the PDF-LaTeX compiler.

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 átlathatatlanná 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.

A dolgozat felépítése:

- Az 1 fejezet a dolgozat valamint a projekt célját definiálja és járja körbe, gyors bemutatót adva a felhasznált technológiák főbb tulajdonságairól.
- A 2

1.2. Security Information and Event Management

A Security Information and Event Management az informatikai rendszer részeinek monitorozásával foglalkozik biztonsági szempontból. Az infrastruktúrában található eszközökhöz hozzáfér a SIEM megoldás, és ez végzi az adatok feldolgozását. Ehhez a rendszerhez kapcsolódnak kliensek, amelyek működésükkel kapcsolatos információkat biztosítanak a szerver irányába valamilyen formában, általában log sorokként. A SIEM szerver ezeket feldolgozza, és úgynevezett biztonsági incidenseket hoz belőlük létre, akár egyéb forrásokból érkező információk felhasználásával. Ilyen egyéb forrás lehet hálózati forgalom, valamilyen adatbázisból lekért adatok, vagy előre definiált, a szerverre feltöltött adatok.

Egy egy incidens jelképez egy olyan történést, amely az operátor számára információval bírhat. Ilyen incidensből emberi léptékkel szinte feldolgozhatatlan mennyiségű érkezik, főleg egy nagy infrastruktúra esetén, ezért a SIEM megoldások támogatnak valamilyen szabályrendszert, amellyel meg lehet adni az egyes incidensek kiértékelésének módját. Ezt általában arra használják, hogy bizonyos paraméterek és szabályok alapján szűrjék a feldolgozandó incidenseket, így az operátornak már egy szűkebb halmazt kell csak végignéznie, valamint ez alapján könnyebb az események látványos kivezetése a felhasználói felületre is.

Jelen applikáció megvalósítása szempontjából a szerverre feltöltött adatok lesznek a legfontosabbak, mivel ezeken keresztül tudunk dinamikus szabályokat létrehozni az incidensek kiértékeléséhez. A dolgozatnak nem célja mélyebben belemenni a ezeknek a biztonsági szabályoknak a használatába vagy működésébe, a megoldandó probléma egyszerűen a kinyert adatok felkínálása volt a SIEM rendszer számára.

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 hasznos információkat, amelyek szükségesek a rendszer hatékony használatához. Ezt a létrejött entitást beosztja a megadott információk szerint a megfelelő, előre definiált csoportokba, 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 projekt 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 pontos feladata egy olyan megoldás fejlesztése, mely lehetővé teszi a fent említett technológiák közti integrációt és az adatok specifikált áramlását. Nem cél egy általános megoldás késztése amely bármilyen, a fenti szektorba tartozó alkalmazás által használható, mindössze egy olyan program fejlesztése, amely az IBM két terméke, az IBM Security QRadar SIEM[1], valamint az IBM Security Identity Manager (ISIM, [2]) közötti integrációt valósítja meg. Erre a feladatra eddig nem volt automatikus megoldás, ezért ez a projekt ezt a hiányt hivatott betölteni.

Az integráció megvalósításához a Java alapú technológiát választottuk. A döntést indokolja, hogy jól illeszkedik a tipikus nagyvállalati környezethez, hogy az IBM kiterjedt tapasztalattal és eszközkészlettel rendelkezik ezen a téren, valamint, hogy 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. **??**) lekérésére és módosítására, így egy később felmerülő projektben is hasznos lehet.

[1] Lásd **??**
[2] Lásd **??**

Ezzel az integrációs modullal 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.
- Valid alkalmazott entitással nem rendelkező felhasználói fiókok - Ezek az árva fiókok biztonsági kockázatokat jelenthetnek, mert a hozzájuk tartozó műveletekért nincs kit felelősségre vonni. Ezért hasznos lehet egy olyan QRadar szabály, ami kifejezetten az ilyen esetekben jelez.
- Adott erőforráshoz hozzáféréssel rendelkező személyek - Ez az információ olyan esetben lehet hasznos, ha mondjuk egy támadás kiinduló pontjaként sikerül azonosítani egy eszközt. Ekkor feltételezhető, hogy az elkövető már rendelkezett ahhoz az erőforráshoz hozzáféréssel, így szűkíthető a potenciális támadók listája.

Az integráció megvalósítására két architektúrát is kidolgoztunk.

1.4.1. Integráció TDI segítségével

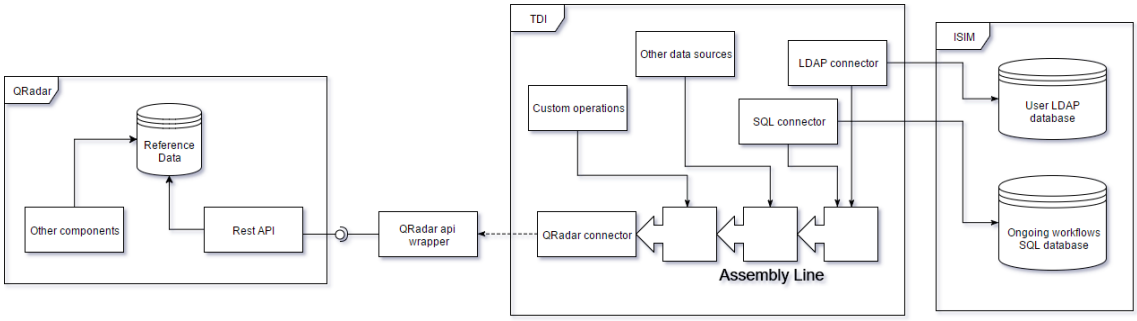
A Tivoli Directory Integrator (TDI [3]) egy gyakran használt, általános célú integrációs eszköz. Ehhez készítettem egy connector-t, amely a fent említett wrapper könyvtár segítségével képes adatokat feltölteni és lekérni a QRadartól. Ez a connector később szabadon újra felhasználható nem csak az ISIM-mel, hanem bármely egyéb forrás integrációjára is.

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

1.4.2. WAS alkalmazás formájában

Ennél a megoldásnál egy IBM WebSphere (WAS[4]) alkalmazást tervezünk elkészíteni, amely a QRadarral való kommunikációra felhasználja az általam fejlesztett wrapper-t.

[3] Lásd **??**
[4] Lásd **??**



1.1. ábra. Architektúra TDI esetén

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.

Ebben a megvalósításban TDI alapú megoldáshoz képest előny, hogy az integrációs adatokhoz tartozó szinkronizáció pontos implementációja saját fejlesztésű, így hozzáigazítható a pontos igényekhez. Emellett a WAS egy menedzselt környezetet biztosít a futtatáshoz, így jobban felügyelhetők az egyes feladatok. A TDI-t használó architektúrával szemben a hátránya, hogy a megoldás ISIM specifikus, más forrásokkal való integrációhoz szükség van a forráskód módosítására.

2. fejezet

Irodalomkutatás és a felhasznált technológiák

2.1. A felhasznált technológiák ismertetése

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

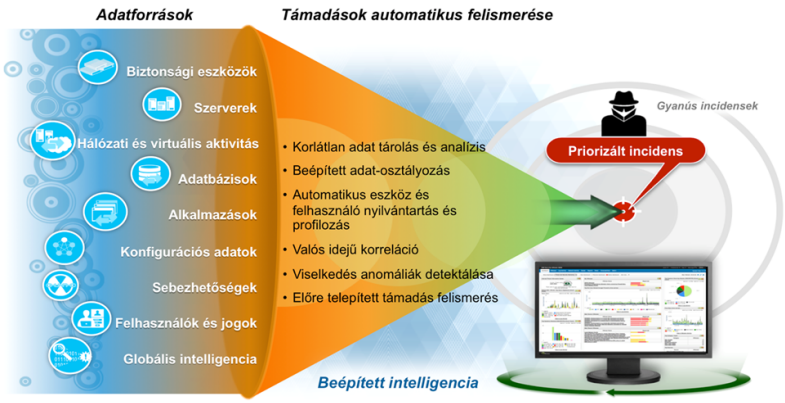
2.1.1. IBM Security QRadar SIEM

Az IBM Security QRadar SIEM az IBM security information and event manager rendszere, ami lehetővé teszi hálózatra csatlakoztatott eszközöknek a megfigyelését biztonsági szempontból. A hálózaton elosztott több ezernyi eszközvégpontból és alkalmazásból származó napló fájl eseményadatait összesíti, és a nyers adatokon azonnali normalizálási és összesítési műveleteket végez, ezáltal képes megkülönböztetni a valódi fenyegetéseket a téves riasztásoktól. Az eseménynaplók betöltésére számtalan automatikus módszer áll rendelkezésre, többek közt olyan közismert protokollok mint a SYSLOG, SNMP, FTP, SCP. Az IBM Security QRadar SIEM ugyancsak képes a rendszer sebezhetőségeinek és az esemény- és hálózati adatoknak az összevetésére, ezáltal segítséget nyújt a biztonsági incidensek rangsorolásában. Emellett lehetőség van egyéb adatforrások felvételére a felhasználó által is, amelyek szintén használhatók a fenyegetések és az incidensek detektálásában. Ezek jelentősége elsősorban a dinamikus szabályok létrehozásában játszik nagy szerepet, mivel ezek segítségével egy automatizált programmal mindig a lehető legfrissebben tarthatók az adatok.

Felvehetők a SIEM-be bizonyos sablonok alapján összeállítható szabályok, amelyeket a rule engine kiértékel a biztonsági incidensekre. A kiértékelés alapján az eseményeket besorolja a megfelelő csoportokba súlyosságuk és egyéb tulajdonságaik alapján, vagy ha szükséges létrehoz egy új, különálló eseményt. Ha az esemény megfelelően súlyos besorolást kap, akkor a felhasználói felületre kikerül egy értesítés erről, de 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 akár 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 dinamikus szabályokban. Ugyanis ezekkel a szabályokkal érhető el, hogy más adatforrásokból (jelen esetben az ISIM-ből) frissen feltöltött információk alapján változzon a kiértékelés, és ha valamilyen adat frissül,



2.1. ábra

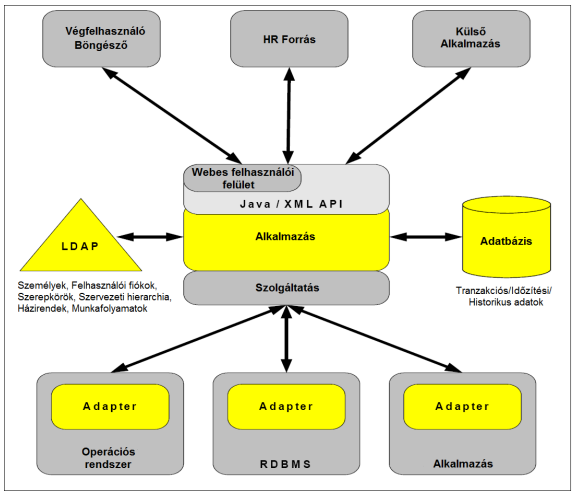
akkor naprakész maradjon a szabály által talált incidensek halmaza. A dinamikusan feltölthető adathalmazok (összefoglaló nevükön reference data) elérhetők egy REST API-n keresztül, így könnyen hozzájuk lehet férni és módosítani őket. Négy féle ilyen adathalmaz áll rendelkezésre:

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

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

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

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



2.2. ábra. Vázlatos példa az ISIM működésére

2.1.2. IBM Security Identity Manager - ISIM

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

Az ISIM tartja karban a kapcsolatot a vállalatnak dolgozó személyek és e személyek IT hozzáférései, jogosultságai között, gondoskodva mind a személyekben, mind a fiókokban bekövetkezett változások az aktuális biztonsági házirend alapján történő szinkronizálásáról. Ennek megfelelően két fő folyamatot definiál a rendszer. Egyrészt a HR forrásokban, tehát a személyek adataiban bekövetkező változások hatásait kell érvénybe léptetni. Másrészt, szükséges a menedzselt rendszeren 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, valamint használ egy relációs adatbázist a tranzakciós adatok, azaz a workflow példányok futási kontextusa (például aktív jog igénylések), audit bejegyzések, ideiglenes szimulációs és ütemezési adatok. Az integrációs modul használata folyamán ebből a két adattárolóból nyerjük ki az adott use-case-hez szükséges információkat.

2.1.3. Tivoli Directory Integrator - TDI

A Tivoli Directory Integrator egy általános célú integrációs eszköz, ami lehetővé teszi több, különböző adatforrás koordinálását és integrációját. Mivel a legtöbb forrás más formátumot használ, és máshogy tárolja az adatot, egy ilyen integrációs lépés során szükséges bizonyos átalakításokat elvégezni az adatok között, 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 ilyen assembly line jelképez egy adat transzfert, a kezdeti adatok felolvasásától az átalakításokon át, a végső kimenet feltöltéséig. A ki- és bemeneti interakció ún. connectorokon keresztül történik, amelyek egy egységes interfészt implementálnak, és valamilyen külső adatforráshoz való kapcsolódást valósítanak meg. Mivel a legtöbb adatforrás más formátumban tárolja

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

3. fejezet

Projekt 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 résztvettem 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, amely alaposan leírja milyen 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 referense set hozható létre, a GET metódussal pedig lekérhető a rendelkezésre álló setek listája.

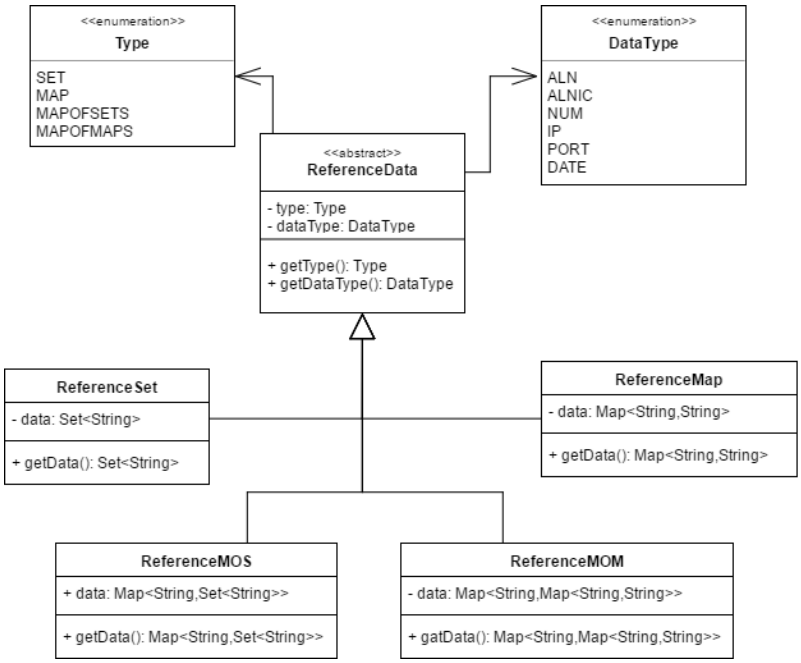
  /{name} - GET, POST, DELETE. Az URL-ben megadott paramétert a QRadar a reference set neveként értelmezi, és ezen keresztül érhető el a set lekérése (GET), teljes törlése (DELETE), valamint egy elemi adat feltöltése (POST).

  /{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 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 token-t, amit minden híváskor fel kell küldeniük. A wrapper osztály ezt konstruktorában kéri, és minden automatikusan minden kérésnél elküldi.

Magának a HTTP forgalomnak és a REST hívásoknak a lebonyolítására az Apache Wink[1] framework-öt használtam. Ez egy egyszerű Java alapú framework, melynek része egy JAX-RS kompatiblis szerver, és egy kifejezetten REST hívások lebonyolítására kiélezett HTTP kliens. A projektben a kliens komponens RestClient osztályát használtam, valamint a framworkkel együtt érkező JSON4J csomagot, a JSON inputok parseolására, valamint a szöveges outputok generálására. A fejlesztés közben külön kihívást jelentett a JSON4J csomag megfelelő osztályainak használata, valamint egymásba ágyazása. Ez a csomag ugyanis

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

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, ami túl nagy adathalmazoknál lassabb működést eredményezhet.

3.2. QRadar connector fejlesztése TDI-hoz

A projekt következő lépése a connector fejlesztése volt a TDI és a QRadar közti integrációhoz. Ehhez segítségemre volt a hivatalos útmutató[2] connector fejlesztéshez, ami az IBM oldalán megtalálható. Egy saját connector fejlesztése TDI alatt abból áll, hogy létrehozunk egy új Java osztályt ami implementálja a megfelelő, TDI által specifikált interfészt, ennek metódusain belül elkészítjük a kívánt üzleti logikát, majd egy xml leíró fájllal együtt, a megfelelő struktúrában becsomagoljuk azt egy JAR fájlba, amit TDI által használt könyvtárak egyikébe másolunk.

Első lépésként tehát elkészítettem egy QRadarConnector 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. Alapvetően hat működési módja lehet egy connector-nak, amit a feljebb említett dokumentum specifikál. Ezek:

TODO !rövid leírás a módokról

- Iterator -
- AddOnly -
- Lookup -
- Delete -
- Update -
- Delta -

Ezek a módok meghatározzák, hogy a fejlesztendő connector-nak milyen metódusokat muszáj implementálnia a megfelelő működéshez. Például az AddOnly mód csak az Initialize, putEntry, és a terminate metódusokat használja, így ha a connector csak ezt akarja használni, elég ezeket implementálni. Ezzel szemben például az Update mód ezeken felül használja a findEntry, és a modEntry metódust is. A fejlesztés első iterációjában egyelőre elégségnek ítéltük meg az Update mód, a Delete mód, és az AddOnly mód használatát, amelyet később még igény szerint kiegészítünk a Delta mód használatával.

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

[2] `https://www.ibm.com/support/knowledgecenter/SSCQGF_7.1.1/com.ibm.IBMDI.doc_7.1.1/referenceguide177.htm`

elején meghívódik. A connector objektum egészen az assembly line végéig életben marad, majd annak végén mielőtt a destruktor meghívódna, lefut az objektum terminate metódusa. Erre azért van szükség, hogy a connector megfelelően felszabadíthassa az általa foglalt erőforrásokat.

A connector életciklus ismeretében, valamint a QRadar és a REST API-jának működése és telejsítménye miatt az alábbi megoldás mellett döntöttünk: az assembly line futása közben nem azonnal kerülnek fel az adatok a QRadar megfelelő reference data-jába, hanem azok először a connector egy változójában akkumulálódnak, és az életciklus végén, a terminate metódus segítségével kommitálódnak.

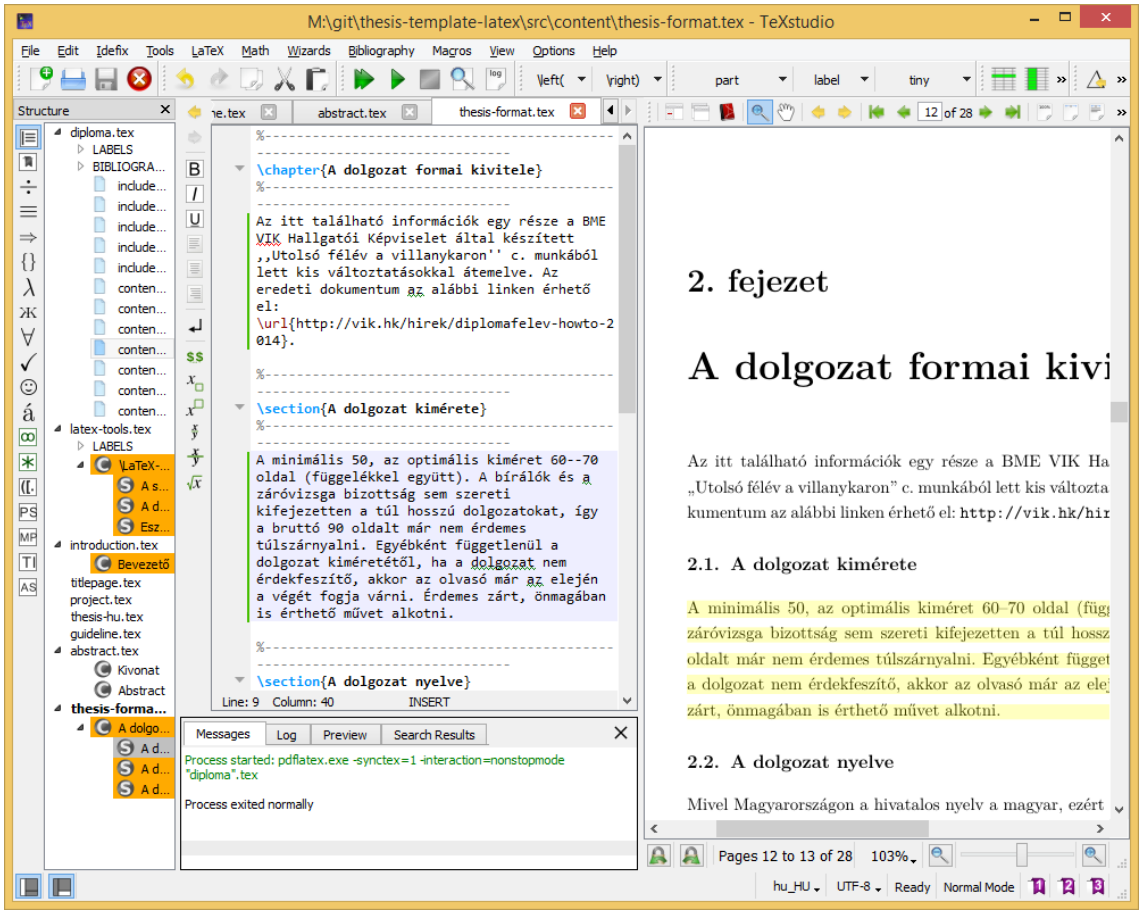
Köszönetnyilvánítás

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

Irodalomjegyzék

Függelék

F.1. A TeXstudio felülete



F.1.1. ábra. A TeXstudio LaTeX-szerkesztő.

F.2. Válasz az „Élet, a világmindenség, meg minden” kérdésére

A Pitagorasz-tételből levezetve

$$c^2 = a^2 + b^2 = 42. \tag{F.2.1}$$

A Faraday-indukciós törvényből levezetve

$$\operatorname{rot} E = -\frac{dB}{dt} \qquad \longrightarrow \qquad U_i = \oint\limits_{\mathbf{L}} \mathbf{Edl} = -\frac{d}{dt} \int\limits_{A} \mathbf{Bda} = 42. \tag{F.2.2}$$