TANSZÉKVEZETŐ

SZAKDOLGOZAT FELADAT

Csák Csongor
villamosmérnök hallgató részére

Hálózati anomáliadetekció SVM-mel

A szupport vektor gépek (SVM – support vector machines) a gépi tanulás módszerei közül egy széles körben elterjedt, nem probabilisztikus, bináris és lineáris osztályozó algoritmusok. Anomália detekcióra gyakran és sikeresen alkalmazzák a szakirodalomban. A hallgató feladata az SVM algoritmus tesztelése és kiértékelése hálózati forgalmi adatokon.

A hallgató feladatai:

- Mutassa be az SVM-ek működését általánosan, valamint ezek közül azon modelleket, melyeket speciálisan anomáliadetekcióra fejlesztettek ki, megfelelően a legfrissebb szakirodalomnak.
- Az elkészitett modellt tesztelje helyi környezetben, több publikusan elérhető adatbázison. Továbbá ezeket az eredményeket értékelje ki, ellenőrizze a modell hatékonyságát. Az eredményeket vesse össze a szakirodalommal.
- A felhasznált hálózati forgalmi adatok legalább kettő különböző támadást tartalmazzanak.

Tanszéki konzulens: Lestyán Szilvia, doktorandusz
Külső konzulens: -

Budapest, 2020. oktober 19.

/ Dr. …… /
egyetemi tanár
tanszékvezető

(Tanszéki levélpapír hivatalos lábrésze)

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

1111 Budapest, Goldmann Gy. tér 3. IV. em. 433.
Telefon: 463-2870, Fax: 463-2871
www.aut.bme.hu • Postacím: 1521 Budapest

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Hálózati anomáliadetekció SVM-mel

Szakdolgozat

Készítette
Csák Csongor

Konzulens
Lestyán Szilvia
Sági Gábor János

2020. december 10.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Csák Csongor*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot 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, 2020. december 10.

Csák Csongor
hallgató

Kivonat

Jelen dokumentumban mutatom be szakdolgozatomat, melyet a Paripa Program keretei között készítettem el. A témám a hálózati anomáliadetekció Support Vector Machine-nel, melyre segítséget a két konzulensemtől kaptam, Lestyán Szilviától a Budapesti Műszaki és Gazdaságtudományi Egyetemről. A szakdolgozatomban bemutatom a Support Vector Machine-eket, mint Machine Learning osztályozó algoritmusokat, leírom a tanulási folyamathoz használt adatbázisokat. Taglalom továbbá a támadásokat, melyekre betanítottam a fent említett SVM-et. Bemutatom az adatbázis dúsítására használt algoritmusomat. Végső soron bemutatom az eredményeket, amelyek ezen szakdolgozat alatt születtek.

Abstract

In this document, I introduce my thesis, which I wrote under the Paripa Program scholarship. The main theme of this thesis is Anomaly Detection with the usage of Support Vector Machines, to which I got help from my two consultant, Szilvia Lestyán from the Budapest Institute of Technology (BME). In this thesis I present the Support Vector Machines, as classification algorithms, and the datasets I used to teach the SVM model. I write about the types of attacks to which the model is trained to recognise. Furthermore, I present the algorithm I wrote to enrich the data. Lastly I exhibit the results of the Support Vector Machine.

1. fejezet

Bevezetés

A jelen szakdolgozat a hálózati anomália detektálásával foglalkozik, amihez Support Vector Machine gépi tanulási módszert használtam. A témán a Paripa Program alatt kezdtem el dolgozni, eleinte társammal Molnár Péterrel, akivel a kutatási területünk hamar szét is vált.

A dolgozatomban a Support Vector Machine klasszifikáló gépi tanulási metódussal sikeresen detektáltam DDoS és SSH Brute-force támadásokat kettő különböző dataszettben. Ehhez egy hosszú adatdúsítási, majd szűrési folyamaton kellett az adatnak végigmennie, majd ezzel az adattal tanítottam be az SVM modellt. A dolgozat végére a modell pontossága elérte a 98%-ot, ami a szakirodalomban kapott eredményekkel egyezik meg. A szakdolgozatom felépítése a következő:

A második fejezetben bemutatom az anomália detekciót, mint fogalomkört, beszélek az anomáliákról, és taglalom, hogy milyen nehézségek rejtőznek az anomália detekcióban. Érvekkel támasztom alá azt az állításom, hogy az anomália detekció egy fundamentálisan nehezebben megoldható feladat, mint bármelyik másik kutatási terület, ahol gépi tanulást alkalmaznak. Ezekután a különböző Machine Learning módszerek használatát mutatom be a cyber security viszonylatában, illetve az anomália detekcióban.

A harmadik fejezetben ismertetem az UGR'16-os adatbázist, ami alapját képezi a szakdolgozatomnak. Bemutatom a benne implementált támadásokat, a szerző által detektált anomáliákat. Bemutatom a fejezetben a másik adatbázist a CICIDS2017-et, amit összehasonlításként választottam. Ez a kettő adatbázis nagyban különbözött egymástól. Ezen felül bemutatom a forgalmi adatok formáját, a Netflow szabványt. Ismertetem azt az adatbázist, amit én hoztam létre még a témával való foglalkozásom legelején. A negyedik fejezetben leírom azt a kettő támadást, amire kihegyeztem jelen szakdolgozatomat. Taglalom ezeknek a felépítését, logikájukat, motivációt használatuk mögött.

Az ötödik fejezetben részletesen ismertetem az UGR'16-os adatoknak a feldolgozására írt algoritmusomat. Leírom a programom részleteit, és motivációit. Ismertetem a megfontolásokat minden egyes döntésem mögött. Bemutatom az eredményt és végezetül a kódot is. Ejtek pár szót a méretbeli problémákról is, amikkel meg kellett birkózni, és hogy milyen döntéseket hoztam ennek érdekében.

A hatodik fejezetben bemutatom a Support Vector Machine klasszifikáló modellt általánosságban, majd azt a lényegében 3 fajtáját, amit magam valósítottam meg. Ezeknek az eredményének a formai különbségét is leírom, majd a futtatási környezetről esik pár szó.

A hetedik fejezetben mutatom be az eredményeket mind a kettő adatforrás viszonylatában. Minden verziót elemzek, kiértékelek.

Az utolsó fejezetben összegzem a tanulságokat és a következő lépéseket, amiket a téma tökéletesítése érdekében lehetne tenni.

2. fejezet

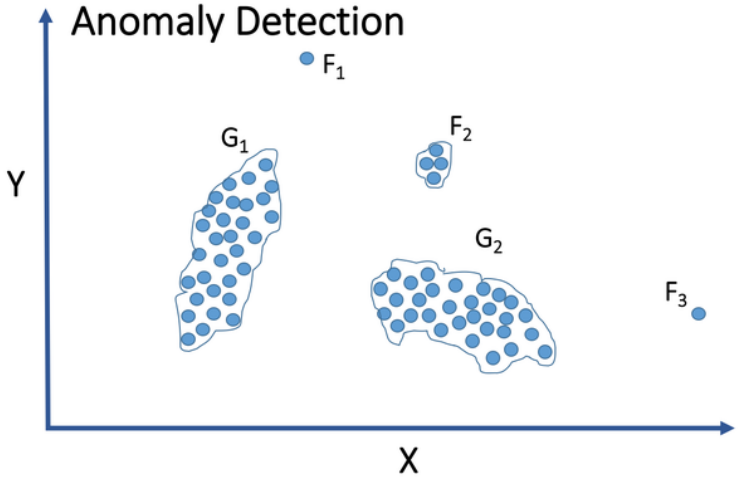
Anomália detekció

Az anomália detekció egy olyan problémával foglalkozik, ahol az adatnak olyan mintázatát keressük, amely nem összeegyeztethető az elvárt viselkedéssel[3]. Ezeket az összeegyeztethetetlen eseteket, adatokat nevezzük többek között anomáliáknak. Az anomália detekciónak számos alkalmazási területe létezik, mint például káros bankkártya használat detektálás, biztosítási és egészségügyi ellátási csalások detektálása, behatolásdetektálás a kiberbiztonság területén, meghibásodás vizsgálat a biztonságkritikus rendszereknél és ellenséges mozgolódás detektálás a harcászatban.

Az anomália detekció fontossága elsősorban abban rejlik, hogy legtöbbször az adatokban előforduló anomáliák valami fontos, sok esetben káros tevékenységre, történésre utalnak. Hálózatoknál az anomália éppen zajló külső támadásokra utalhat vagy akár káros belső tevékenységekre.

2.1. Anomáliák

Anomáliáknak nevezzük azokat a mintákat, melyek nem egyeznek meg egy adathalmaz normális, elvárt viselkedésével.



2.1. ábra. Példa anomáliákra egy 2D-s adathalmazon.[5]

A 2.1. ábrán látható egy egyszerű, 2 dimenziós adathalmaz. Az adathalmazhoz kettő darab normális rész tartozik, a G_1 és a G_2 halmaz. Az F_1, F_2 és az F_3 halmazok

elemei távol helyezkednek el a normálisnak tekintett elemektől, így őket anomáliák közé soroltuk.

Ahhoz, hogy hasznosan fel tudjuk dolgozni a detektált anomáliakat, és értelmesen használni tudjuk ezeket, kulcsfontosságú, hogy ki tudjuk választani a számunkra értékes és hasznos attribútumokat. Ez a hasznosság, értékesség különbözteti meg többek között az anomáliákat a zajtól. A zaj elemzés során érdektelen adat, tehát ezt is ki kell tudni szűrni.

2.2. Nehézségek

A fentebb leírtak alapján az anomália detekció a normális adathalmazba nem illő elemeknek a detektálásáról szól. Tehát jó megközelítés, hogyha a definiáljuk a normális halmazt, és minden olyan adat, ami nem eleme ennek, anomália lesz. Azonban ezen megoldás is számos nehézséggel jár:

- A normál tartomány teljes definiálás nagyon nehéz feladat, főleg hálózati anomália detekciónál, mivel gyakran az anomália nagyon hasonlít a normális viselkedéshez, és a határ, ami alapján osztályozunk, nem minden esetben egyértelmű és pontos. Illetve lehetnek olyan normális adatok, melyek nagyon közel helyezkednek el a határhoz, esetleg át is lógnak azon. Egyértelműen ennek az ellentéte is előfordulhat, miszerint a káros tevékenység, vagy anomália lóg át a határ másik oldalára.

- Sok alkalmazási területen a normális viselkedés folyamatosan változik, fejlődik, ezért a mindenkori modell hamar elavaul, pontatlanná válik.

- Káros tevékenységből származó anomália sokszor nehezen detektálható, hiszen a támadó direkt próbálja meg utánozni a normális viselkedéshez közeli mintázatot, hogy ezzel is elrejtse magát. Ez önmagában lényegesen megnehezíti a detektálást.

- Különböző alkalmazási területeken különböző modellek felállítására van szükség, mivel ami anomáliának számít egy alkalmazási területen, az nem feltétlenül számít annak egy másiknál.

- Sokszor tartalmaz az adathalmaz zajt, ami hasonlíthat az anomáliákra, de mégsem az. Így a zajok szűrése is gondot okozhat, és torzíthatja a kiértékelést.

- A rendelkezésre álló címkézett adatbázisok száma meglehetősen limitált, ezért különböző algoritmusok betanításához sok esetben nem áll rendelkezésre elég adat.

Ezek miatt az anomália detekció fundamentálisan egy nehezen megoldható feladat. Merem állítani, hogy nincsen tökéletes megoldás. A problémáknak gyakran csak egy specifikus részét lehet megoldani egy fajta szemlélettel, egy másik része azonban egy teljesen más megközelítést igényel.

2.3. Machine learning a cyber security-ben

A Cyber Security olyan technológiák, folyamatok összessége, mely rendszerek, hálózatok számítógépek, programok és adatok védelmére szolgálnak támadások, illetéktelen belépések, változtatások és megsemmisítések ellen[2].Cyber Security rendszerek hálózati biztonsági rendszerekből és számítógép védelmi rendszerekből állnak. Mindegyikük fel van szerelve legalább egy tűzfallal, egy antivírussal és behatolás detektáló rendszerrel (IDS). Az IDS-ek segítenek felfedezni és beazonosítani illetéktelen behatolásokat, adat másolásokat és törléseket[14]. A biztonsági áttörések lehetnek külső forrásúak, tehát egy külső szervezet indít támadást a rendszerünk ellen, vagy belső forrásúak, amikor a szervezetünk belsejéből indít támadást a szervezet ellen.

2.3.1. Elemzési módszerek

Három fajta elemzési módszertant különböztetünk meg az IDS-ek tekintetében: az anomália alapú elemzést, a szignatúra alapú elemzést, illetve a kettő ötvözetét, a hibrid elemzést.

A szignatúra alapú elemzésr arra tervezték, hogy az ismert támadások szignatúrái után keressünk. Ez effektív megoldás az ismert támadások detektálására anélkül, hogy magas számú fals pozitív riasztást generálnánk. Azonban ez az adatbázis folyamatos frissítésével jár. A szignatúra alapú elemzés nem képes kimutatni 0-ik napi támadásokat.

Az anomália alapú elemzés modellezi a hálózat vagy rendszer normális működését, és az ettől való eltérést ismeri fel. Ez a fajta elemzés képes kimutatni a 0-ik napi támadásokat is. Egy másik előny, hogy minden rendszernek, hálózatnak, applikációnak más és más a rájuk vonatkozó normális működése, ezért egy támadó nem tudja előre, hogy milyen cselekményeket követhet el észrevétlenül. Ezen kívül az anomáliákra, amire a rendszer riaszt, később szabályt lehet írni, hogy ezeket szignatúra alapján ki tudjuk szűrni. A legnagyobb hátránya az anomália alapú elemzéseknek a fals pozitívok aránya, mivel egy előzetesen nem látott, de teljesen legitim viselkedést is anomáliának jelez a rendszer.

A hibrid elemzési módszerek ötvözik a szignatúra alapú és az anomália alapú elemzési technikákat. Ezek alkalmazása megnöveli a nem ismert támadások detektálásának arányát, és lecsökkenti a fals pozitívok arányát. A valóságban nem nagyon lehet tisztán anomália alapú elemzést végezni, a legtöbb ilyen metódus valójában hibrid. Ezzel párhuzamosan amikor Machine Learning metódusokról beszélünk, akkor az anomália és hibrid metódusokat együttesen kezeljük.

2.3.2. Machine learning az IDS-ekben

Machine Learning technikák használta az Intrusion Detection Systemeknél azon alapszik, hogy felépítünk egy modellt a gépi tanulási módszerrel, és így végezzük később az osztályzást[21]. Ehhez betanítási adatra van szükség. Az adatnak tartalmaznia kell minden egyes vizsgálandó adattípusra jellemző arcvonásokat. Optimális esetben rendelkezésre áll az adatbázisban a címke, ami alapján a rendszer tudja, hogy anomália-e, avagy támadás az adott adat. Ezen felül érdemes nem egy összesített anomália címkét tenni támadásokra, hanem specifikus jellemzőkre specifikus címkét tenni. A címke elérhetőségét figyelembe véve 3 különböző osztályát tudjuk

megkülönböztetni a Machine Learning tanításában: Felügyelt Tanulás, Felügyeletlen Tanulás, Nagyjából Felügyelt Tanulás.

Elterjedtebb Machine Learning módszerek az IDS területen:

- Bayesian Hálózat
- Markov Modellek
- Neurális Hálók
- Fuzzy Rendszerek
- Clusterizáló módszerek
- Support Vector Machine
- Választási Fák

3. fejezet

Felhasznált adatbázisok

A címkézett adatt egy Machine Learning algoritmus tanításához kimagaslóan fontos. Éppen ezért elengedhetetlen volt, hogy a találjak egy olyan adatbázist, amely címkézett hálózati folyamokat tartalmaz, és az SVM-et be tudjam tanítani a címkézett adatokon. A Következő alfejezetekben bemutatom a felhasznált adatbázisokat.

3.1. Netflow

A Cisco V9-es Netflow szabványát [4] vettem alapul szakdolgozatom adatfelhasználása tekintetében. Ez egy router által feldolgozott hálózati forgalmi adat. A folyam attribútumai között szerepel a Timestamp, ami UNIX timestamp kódolású, és a folyam kezdetét jelölő időpontot tárolja, a Duration azt jelzi, hogy mennyi ideig tartott a folyam. A Source IP attributum tárolja a kommunikáció forrásának IP címét, míg a Destination IP a cél IP címét. Source és Destination Port tárolja a forrás és cél portszámot. A kommunikáció protokollját is tárolja a Netflow adat, illetve a TCP flageket is egyaránt. Továbbá tárolja, hogy kommunikáció alatt mennyi és mekkora méretű packet haladt a két fél között.

Anomália detekciónál nem ritka, hogy hálózati folyamokat vesznek alapul[11].

Sperotto [20] adatbázisa egy folyam alapú adatbázis. Ez az adatbázis címkézett, nem szintetikus folyamok összessége. A legnagyobb hátránya, hogy méretében limitált, mivel erőforrás hiányában egyetlen honeypot-ot monitorozik egyetemi környezetben. 6 nap a monitorozási időtartam.

A [13] alatt leírt UNSW-NB15 dataszettben az író egy támadás automatizáló program segítségével, az úgynevezett IXIA PerfectStorm használatával 9 különböző valós támadási családot implementált több szerveren. 2 millió flow-t sikerült összegyűjtenie 31 óra alatt. Ez az adatbázis szintetikus forgalmat ír le, tehát az internetnek nem a valós viselkedését.

3.2. UGR '16

Az én választásom többek között az UGR '16 adatbázisra esett. Ez az adatszett valós forgalomra épül friss támadásokkal. Más adatbázisokkal ellentétben ez egy viszonylag hosszú időintervallumot ölel fel, ami az IDS-eknél nagy előny, mivel látható a forgalom periodicitása.

Az UGR 16-os Dataszett egy címkézett Netflow adat, amely 2016 májusától egészen június végéig tartalmaz hálózati folyam logokat. Ezeket a dataset megalkotóinak és a köré írt tanulmánynak otthont adó granadai egyetem routerje logolta. Normálisnak tekintett internetes forgalmat dúsítottak szintetikus támadásokkal. Ezeknek a támadásokank az implementációját később kifejtem. Az adatot címkézték támadások szerint. A Datasett attribútumai megegyeznek a Netflow V9-es szabványának elemeivel, annyi eltéréssel, hogy az adatszerkezet utolsó oszlopában szerepel a címke.

3.2.1. Támadás implementáció

Mivel Netflow került rögzítésre, és így az egyes kérések payloadja nincs rögzítve ilyen formában, ezért olyan támadások nem kerültek implementálásra, amiket a payload vizsgálata során detektálhatóak lehetnének. Emiatt csak hálózathoz közeli támadásokat implementáltak a szerzők:

- Low-rate DoS: TCP SYN packetek kerülnek küldésre az áldozat felé a 'hping' eszköz használatával. A cél port száma 80, így a forgalom keveredni fog normális, háttér webes forgalommal. Minden csomag mérete 1280 bit és a ráta pedig 100 packet másodpercenként. Mint látható a támadás frekvenciája viszonylag kicsi, azért, hogy a hálózat normális működését ne akadályozzák. Három különböző, ilyen alapokkal rendelkező DoS támadást hajtottak végre:
  - DoS11: 1 az 1 ellen DoS támadás, ahol a támadót az A1-el és a célpontot a V21-el jelölték. A teljes időtartama a DoS11-nek 3 perc.
  - DoS53s: Az öt támadó, A1-A5 támadja a három célpontot, mindegyiket egy külön hálózatról három percig. Pontosan, a támadás struktúrája a következőképpen néz ki: (A1, A2) $\rightarrow$ V21, (A3, A4) $\rightarrow$ V31 és A5 $\rightarrow$ V41. Az 's' betű a DoS53s végén a szinkron támadást jelöli, ami annyit jelent, hogy a támadások szinkron módon kezdődtek, egy időben. A szinkronizáció miatt ezeknek a támadásoknak az egész időtartama is 3 perc.
  - DoS53a: a támadások ugyanúgy működnek, mint a DoS53s alatt, annyi változással, hogy a célpontok szekvenciálisan kerültek kiválasztásra. 3 percig tartott a támadás, amit egy 30 másodperces inaktív periódus követett, aztán jött a következő célpont. Így a támadás teljes időtartama 10 perc lett. Ebben az esetben az 'a' betű a névben az aszinkront jelenti.
- Port Scan: Egy folyamatos SYN flag scan a célpont általános portjaira 3 percig, az nmap eszközt használva. Ennek a támadásnak 2 verzióját implementálták:
  - Scan11: 1 az 1 elleni scan támadás, ahol a támadó az A1 és a célpont, akinek a portjait scanneli, a V41.
  - Scan44: Négy a négy elleni scan támadás, ahol a támadók az A1, A2, A3 és A4 kezdik egyszerre scannelni V21, V11, V31 és V41 portjait. Párhuzamosan hajtották végre a támadást, egy kezdeti időponttal, így a támadás időtartama 3 perc volt itt is.
- Botnet: Ez a fajta támadás számomra irreleváns, szakdolgozatom alatt nem foglalkoztam Botnet támadással.

Ezen felül, mivel a dataszett valós, normál hálózat adatait tartalmazza, szintetikus támadásokkal, biztosak lehetünk benne, hogy valós támadás is detektálásra került. Ezeket a szerzők anomáliaként címkézték. Ezeknek is több fajtáját címkézték:

- SSH Scan anomália: Az 16/04/10 – 16/04/21 intervallumban megnövekedett SSH kérések jelentkeztek a rendszerben. Ezt a szerzők manuálisan megvizsgálták, és azt találták, hogy egy gépről érkeztek a kérések. Ráadásul ez a gép szokatlanul nagy portscan tevékenységet is folytat, emiatt osztályozták be az erről a gépről érkező SSH kéréseket "anomaly-sshscan" címke alá. Ez az anomália, amit én is felhasználtam szakdolgozatom alatt.
- UDP Scan: Ezt az anomáliát sem vettem figyelembe, így nem részletezem a szerzők detekciós metódusát.
- SPAM: Ugyancsak nem vettem figyelembe, így ennek sem részletezem a szerzők detekciós metódusát.

3.3. CIC-IDS2017

A Canadian Institue for Cybersecurity (CIC) hosszú évek óta gyárt dataszetteket IDS-eknek. A 2017-es dataszettjük normális hálózati forgalmat és támadást is egyaránt tartalmaz, amik valós, igazi adatokra hasonlítanak [6]. A dataszettben megtalálhatóak a hálózati forgalom analizálását végző CICFlowMeter eredményei, amik a címkézett folyamok Timestamp alapján rendezve, forrás és cél IP címeket, portokat tartalmazva, valamint a protokolt.

A fő célja a szerzőknek az adatbázis megalkotásakor, hogy reális háttér forgalmat generáljanak. Sharafaldin [16] B-Profile System-jét felhasználva modellezték 25 embernek a viselkedését. Ezek a viselkedések HTTP, HTTPS, FTP, SSH és email protocol forgalom alapján lettek modellezve, hogy valósnak tűnő háttérforgalmat kapjanak.

Az adat 2017 július 3-án hétfő reggel 9 órakor kezdődik és július 7-én pénteken délután 5 órakor fejeződik be, tehát összesen 5 napot ölel fel. A hétfői nap csak normális forgalmat tartalmaz. Az implementált támadások többek között Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet és DDoS támadások voltak. Ezek kedd, szerda, csütörtök és péntek reggel és délután kerültek futtatásra.

Az előző dataszett kiértékelő keretrendszerüknél [7] 11 kritériát jegyeztek fel, ami nélkülözhetetlen egy megbízható dataszett létrehozásánál. Ezelőtti adatbázisaik közül semelyik nem rendelkezett mind a 11 kritériával. Néhány feltétel:

- Teljes hálózati konfiguráció: egy teljes hálózati topológia, ami tartalmaz Modemet, Tűzfalat, Switcheket, Routereket és különböző operációs rendszerek vannak felhasználóként a rendszerben, mint például Windows, Ubuntu, és Mac OS X.
- Teljes forgalom: Egy felhasználói profil megalkotó rendszerrel, 12 felhasználóval a célpont-hálózatban, és valós támadásokkal a támadó-hálózatban.
- Címkézett adat: A teljes adat címkézve van, előre kigondolt ütemezésben jönnek a támadások.

- Teljes interakció: A két hálózaton belül is kommunikálnak az eszközök és a hálózatok között felépített internetkapcsolatot is használva a két LAN között is van kommunikáció.
- Teljes felvétel: Az összes hálózati kommunikáció fel lett véve, mind a két hálózatban.
- Elérhető protokollok: Minden általánosságban használatos protokoll jelen volt, mint például: HTTP, HTTPS, FTP, SSH és email protkollok.
- Támadás diverzitás: A legelterjedtebb támadások a 2016-os McAfee report szerint, mint a Web alapú, Brute force, DoS, DDoS, Infiltration, Heart-bleed, Bot és Scan is jelen van a dataszettben.
- Heterogenitás: A hálózati forgalom a fő Switchen lett felvéve, minden támadás esetén.
- Metadata: Teljesen elmagyarázott metadata, ami az időt, a támadásokat, a címkéket és a folyamokat is tartalmazzák.

3.3.1. Adat formája

A CIC-IDS2017 dataszett formája merőben különbözik az UGR'16-tól. Teljesen Machine Learning-re van optimalizálva, meglehetősen sok, dúsításból eredeztetett attribútummal, ami az UGR'16-ban nem fellelhető. Ez redundánssá változtatja azt a programomat, amelyet az UGR'16 dúsítására írtam.

3.4. Ip feketelista

Munkám legelején AZ UGR'16-os datasett köré írt tanulmány nyomán egy relatíve egyszerűbb, önmagában helytálló feladatot választottam későbbi adatszűrés szempontjából. Ehhez a feladathoz is az inspirációt a tanulmány adta, miszerint feketelistázott IP címekről érkező forgalmakat címkézik. Ez a címkézés nem szűrést jelent, ezek a forgalmak a hálózati kommunikációban jelen vannak. Én a jelen hiányosságot pótolandó, első körben egy összesített IP blacklist adatbázist akartam felállítani. Természetesen tudtam, hogy az én feketelistám közelébe sem fog érni az évtizedes múlttal rendelkező, internetes reputációs adatbázisoknak, mint például az AbuseIPDB adatbázisnak. Ez az adatbázis széleskörben használatos hivatalos, nagyvállalati körökben is.

Én ingyenesen elérhető adatbázisokból szedtem össze az IP-ket. Ezek az adatbázisok a következőek voltak: [1], [12] és [17].

Ezeket letöltöttem, összefűztem, majd minden esetleges duplikációt kiszűrtem az adatbázisból. Az összesített adatbázis mérete nem lett hatalmas az abuseipdb.com-hoz hasonlítva. Mindössze tizenötezer feketelistázott IP. Azonban ez elindulásnak teljesen tökéletes volt.

4. fejezet

Támadások

Mint ahogy azt a 2.3.1-ben is leírtam, a Machine Learningre alapuló anomália detekciós módszerek nagyrészt hibrid elemzéssel működnek. Tehát normális hálózati forgalomra illetve támadás szignatúrára van szükségünk, hogy jól be tudjuk tanítani az algoritmusunkat. Hogy milyen támadás detektálását fogja elvégezni az SVM, az nagyban múlik a dataszett kiválasztásán. Én a 3.2 alapján az UGR 16-os dataszettet választottam első körben. Ebben a dataszettben limitált számú, és fajtájú támadás volt implementálva. Ezek közül a Denial of Service típusú támadások mellett döntöttem, és egy Scanning típusú támadás mellet, nevezetesen az SSH Scanning mellett. Ez Anomáliaként szerepel az UGR 16-os adatbázisban. Jelen fejezetben kifejtem ezen támadásokat elméleti szinten.

4.1. Denial of Service

Denial of Service (DoS) támadás, vagyis magyarul szolgáltatás megtagadás az egyik legrégebbi kiber támadási forma. Ahogy a nevében is benne van, a DoS támadás célja hogy megtagadja a megtámadott fél szolgáltatását a legitim felhasználóval szemben. Példának okáért, egy sikeres DoS támadás egy webáruház ellen ellehetetlenítené a vásárlókat, hogy rendelést adjanak le, vagy akár meglátogassák a webáruház oldalát.

A Distributed Denial of Service (DDoS), vagy elosztott szolgáltatás megtagadás támadás egy DoS forma, ahol a támadást nem egy készülék hajtja végre, hanem egy rakás készülék, a világon elszórva. Ezt gyakran nevezik botnet hálózatnak. A DoS ellenben egy hálózatra csatlakozott gép folytatja és árasztja el kérésekkel a célpontot. [9]

Általánosítva, a DoS támadások 3 csoportba sorolhatóak:

1. **Méret alapú támadások:** Ide tartoznak az UDP elárasztások, ICMP elárasztások, és más hamis-packet árasztásos támadások. A támadás célja, hogy telítse a célpont hálózatának a sávszélességét. A támadás mértékegysége Bit per másodperc (Bps).

2. **Protocol támadások:** Ide tartoznak a SYN árasztások, a fregmentált packet támadások, Ping of Death, Smurf DDoS és a többi. Ez a támadási forma a szerver erőforrásait emészti fel, vagy pedig a közöttes kommunikációs eszközöket, például tűzfalakat, a terhelés elosztókat. A támadás mértékegysége packet per másodperc (Pps).

3. **Alkalmazási réteg támadások:** Ide tartoznak az úgynevezett low-and-slow támadások, GET/POST árasztások, az olyan támadások, amelyek az Apache szerver, Windows vagy OpenBSD sebezhetőségeket célozzák. Legitimnek tűnő kéréseket tartalmazva az a cél, hogy a szervert terheljék túl, így okozva leállást. A mértékegység itt kérés per másodperc(Rps).

Néhány ismertebb DoS/DDoS támadás:

- Az UDP elárasztásos támadás, definíció szerint egy olyan DDoS támadás, amely elárasztja a célpontot User Datagram Protokol(UDP)-beli csomagokkal. Az UDP packet egy "megbízhatatlan" packet. Ez azt jelenti lényegében, hogy a küldő felet nem érdekli, hogy a fogadó fél megkapta-e a csomagot. A támadás célja, hogy a remote host random porjait árassza el UDP packetekkel. Ez azt eredményezi, hogy a host folyamatosan megvizsgálja milyen applikáció figyel az adott porton, és ha nem talál semmilyen applikációt, egy ICMP 'Destination Unreachable', 'avagy cél nem érhető el' csomaggal válaszol. Ez az egész folyamat a host erőforrásait annyira leterheli, hogy elérhetetlenné válik.

- ICMP (Ping) árasztás hasonló elven működik mint az UDP árasztás. Túlterhelik a célpontot ICMP Visszhang Kérésekkel (ping), olyan gyorsan ahogy csak tudják, nem várva a host válaszára. Ez a támadás mind a bejövő, mind a kimenő sávszélességet felemészti, egy rendszerszintű lassulást eredményezve.

- SYN Flood kihasználja a TCP kapcsolat "three-way handshake"-jének a gyengeségét, ahol a SYN kérésre, egy TCP kapcsolat létrehozásához, SYN-ACK válasszal kell válaszolnia a hostnak, amit aztán egy ACK válasszal kell megerőstenie a kérelmezőnek. Egy SYN árasztásnál a támadó több SYN kérést küld, de vagy nem is válaszol a host SYN-ACK válaszára, vagy egy hamisított IP címről küld SYN kérést. Lényegében a host rendszer folyamatosan vár a másik fél kézfogására az összes kérésnél, ezzel lekötve az erőforrásait, addig a pillanatig, amíg semennyi új kapcsolat nem tud létrejönni, ezzel tagadva meg a szolgáltatást.

DDoS támadások manapság is hatalmas népszerűségnek örvendenek. Az utóbbi években is sebesen nő a számuk mind előfordulásuk szerint, mind nagyságuk szerint. A trend manapság a kevesebb támadási idő, de nagyobb packet per másodpercű támadások népszerűségét mutatja. [9]

A támadók típusai motivációjuk szerint csoportosítva:

- **Ideológia** - Az úgynevezett "hacktivisták" DDoS támadásának célkeresztjében olyan websiteok állnak, akikkel ideológiai szempontból nem értenek egyet.

- **Business nézeteltérések** - Cégek a konkurens cégrk weboldalait tudják elérhetetlenné tenni egy DDoS támadással, hogy a másik cég hátrányba kerüljön például Fekete Pénteken.

- **Unalom** - A cyber "fenegyerekek" előre megírt scripteket futtatva indítanak DDoS támadásokat. A támadókat általában az unalom vezérli, egy kis adrenalin löketet várva.

- **Nyerészkedés** - A DDoS a támadó eszköze, hogy pénzt zsaroljon ki a célponttól.
- **Kiber Háború** - Kormány által engedélyezett DDoS támadások az ellenség infrastruktúráját tudja bedönteni, ezzel szerezve előnyt az ellenféllel szemben.

DDoS támadások ellen különböző védelmi eszközök tudnak hatékony védelmet nyújtani. Például a Radware tűzfal egy széles körben használatos védelmi eszköz. Általában Radware a hálózatunk elején helyezkedik el, mint elsőszámú védelmi vonal. Ez a tűzfal szűri meg a bejövő forgalmat DDoS támadásoktól, különböző szignatúrákra figyelve. Ezek a szignatúrák a gyártó által előre definiáltak is lehetnek, de a felhasználó is tud szabályt létrehozni, hogy a saját hálózatára szabja a védelmet. Lehetőség van gépi tanulási módszereket is alkalmazni a Radware-en belül.

A Machine Learning a DDoS védelemben hatékonyan alkalmazható, hogyha adottak a kedvező körülmények. Kell ismernünk a saját hálózatunk méreteit. Hogy mekkora az átlagos bejövő forgalmunk, mekkora az átlagos kimenő. Tudnunk kell, hogy ezek hogyan aránylanak egymáshoz napközben, este, munkanap, illetve hétvégén. Ezek fontos számok, hiszen ezekhez viszonyítva az aktuális forgalmat, jó előérzetünk lehet, hogy DDoS támadás áldozatai vagyunk-e avagy sem. A saját hálózatunk ismeretében akár tresholdokat, avagy küszöböket adhatunk meg, amelyek átlépésével adott irányú forgalmakat eldobhatunk, mivel jóeséllyel támadás alatt állunk.

Hibrid megoldásként egy adott szignatúrát keresve, és ezt hasonlítva a saját hálózatunkra viszonyított átlagos viselkedéshez mondhatjuk el, hogy támadás ért minket. Például ha átlagoshoz viszonyítva erőteljesen megszaporodik a hibás three-way-handshake, akkor eldobjuk ezeket.

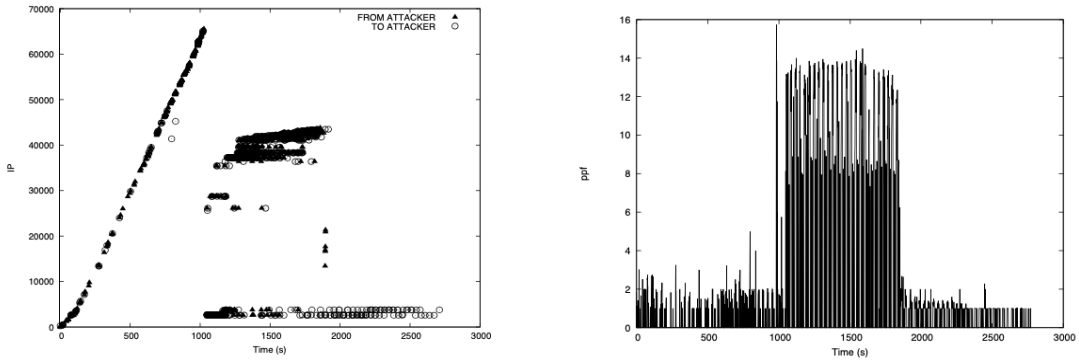
Az én szakdolgozatomban hasonló logikával próbáltam meg a DDoS támadásokat, és az SSH támadásokat detektálni. Részletesen leírom a megközelítésemet a 5. fejezetben.

4.2. SSH Brute-force

Egy nagy népszerűségnek örvendő támadás a Secure SHell-t (SSH) célozza, amivel a támadó hozzáférést szerez, vagy akár teljesen átveszi a remote host feletti irányítást. Amint bejutott, a támadó nem csak a célpontot tudja szabotálni, de akár további célpontok támadására is tudja használni. A támadás detektálása, SSH tekintetében különösen fontos, hogy elkerüljük a károkat amik a rendszert érhetik. [8]

A legtöbb IDS csomag alapú, tehát a csomagok payloadjait vizsgálják. Azonban ez a megközelítés a nagy gyorsaságú hálózatoknál nem feltétlenül járható út. Emellett a titkosított forgalomnál, mint amilyen az SSH forgalom, a csomagok payloadjainak vizsgálata igen problémás feladat. Emiatt a folyam alapú vizsgálat helytállóbbnak tűnik.

Sperotto a munkáiban [19], [18] analizálta az SSH brute-force támadásokat, és felismerte a karakterisztikájukat folyami szinten. Megmutatta, hogy a brute-force SSH támadások tipikusan három fázisból állnak, amiket fel lehet ismerni a folyamuk karakterisztikájából. Kiderült, hogy egyértelműen látszik a támadás előrehaladása az időben.



(a) IP címek időbeli eloszlása **(b)** Csomagok időbeli eloszlása

4.1. ábra. Forgalom a [19]-ben

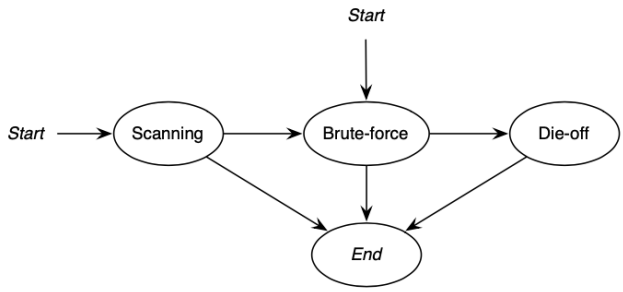
4.1a ábra mutatja a támadó kapcsolódását egy IP cím blokkra. Az ábrán minden pont különböző IP címekhez tartozó folyamot jelent. A [19] és [18] alatt meghatározott 3 támadási fázis:

1. **Szkennelési fázis** - A támadó egy IP cím blokkon szkennel keresztül, hogy megtalálja melyik IP címen van nyitva az SSH port (TCP port 22).

2. **Brute-force fázis** - A támadó megpróbálja felvenni az SSH kapcsolatot azokkal a hostokkal, akiknél ez lehetséges. Nagy számú felhasználónév/jelszó kombinációval próbálkozik belépni.

3. **Elhalási szakasz** - Egy sikeres bejutás után is fennmarad a kapcsolat a támadó és a célpont között. Ez annak köszönhető, hogy a támadó parancsokat hajt végre a célpontnál.

Egy sikeres támadást mutat meg a 4.1 ábra. A szkennelési fázis t = 1000s időpontig folyik, amit egyből a brute-force fázis követ t = 1750s-ig. Ezután a harmadik fázis látható egészen t = 2750s-ig, ahol a t a támadás kezdetét jelöli.



4.2. ábra. Támadás folyamatábrája

4.1b megmutatja, hogyan alakult a csomag per folyam(PPF) szám a támadás alatt. Szkennelés alatt kisebb számú PPF volt jellemző, brute-force alatt szignifikánsan megnövekedett a PPF illetve az időegység alatti folyamok száma, míg az elhalási

fázisban szintén relatíve kicsi forgalom volt jellemző. Azonban nem minden támadás halad át ezen a három lépésen. 4.2 ábrán látható a támadások az alakulásának a folyamatábrája. Vannak támadások, amelyek egyből a brute-force fázisban kezd, és van olyan, ami egyből a brute-force fázis után leáll.

5. fejezet

Adatfeldolgozás

Az adatok előfeldolgozása jelentette a leghosszadalmasabb munkát. A Machine Learning metódusoknál a betanítási folyamat, és az ahhoz tartozó adat előfeldolgozása teszi ki a munka legalább több mint 80 százalékát.[10] Ez jelen szakdolgozatom alatt sem alakult másképp.

5.1. Megfontolások

Mint ahogy elöljáróban azt leírtam az 4.1 pontban, nagyon fontos, hogy ismerjük a hálózatunkat, vagy esetemben az adatbázist, hogy abban DDoS támadásra utoló jeleket tudjunk találni. Ahogy azt a 3.2 pontban bemutattam, az UGR'16-os adatszerkezetnél különböző módon hajtották végre a DoS és DDoS támadásokat. Ezek alapjukat tekintve azonosak voltak, de mégis háromféleképpen fordultak elő, mint szintetikus támadások.

Dolgozatomban az adatok feldolgozásánál a hálózati forgalom alakulását vizsgáltam, és azt, hogy egyes folyamok hogyan viszonyulnak a forgalom alakuláshoz. A 4.2 szekcióban megismertek szerint egy átlagos SSH támadás teljes időtartama nagyjából 2750 sec, tehát nagyjából 45 perc. Figyelembe véve azt, hogy nem mindegyik SSH támadás megy végig mind a három fázison, és hogy a DDoS támadások manapság rövidebbek, mint régebben, egy 30 perces időablak racionálisnak tűnik, mint vizsgált tartomány. A hálózat változását 30 percig visszamenőlegesen nézem, ebben a 30 percben számolom ki, hogy átlagosan mennyi folyam volt, mekkorák ezeknek a folyamoknak a méreteik, és hogy mennyi packet volt a forgalomban. Ezt vizsgálom az egész folyamra, és az adott forrás IP címre is.

A DDoS támadásoknál, különösképpen a SYN flood támadásnál nagy szerepet játszik a TCP flag felismerése a folyamban. Mivel a Netflow v9 egy attribútumban tárolja az összes TCP flaget olyan formában, hogy ha a flag állításra került a folyam során akkor megjelenik a flag kezdőbetűje, azonban ha nem került állításra, egy pont jelenik meg a flag betűje helyett. A flageknek rögzített sorrendjük van, a Netflowban való jelölésük is kötött sorrendben jelentkezik. Mivel az SVM egy matimatikai alapokon nyugvó Machine Learning metódus, ezért nincsen sok értelme egy stringet átadni az algoritmusnak. Ehelyett One-Hot enkódolással bontom fel a TCP flageket jelölő oszlopot több oszlopra. Minden egyes új oszlop egy flagnek felel meg, és értéke 1 vagy 0 annak függvényében, hogy a folyamban szerepelt-e az adott flag.

5.2. Megvalósítás

Az UGR'16-os dataszett feldolgozására egy python skriptet írtam. A script lényegében dúsítja az adatot, megszűri azt és egy adott mennyiséget random módon kiválaszt. Ez a dúsítás jelen esetben a hálózati forgalom előéletét veszi alapul. Megnézi, hogy egy időpillanathoz számított fél óráig visszamenőlegesen milyen trend volt igaz a hálózati forgalomra. Kiszámítja, hogy összesen mennyi és mekkora csomagok haladtak a hálózaton, illetve ezeknek a csomagoknak az átlagos mérete mekkora volt, és átlagosan mennyi csomagot küldtek. Továbbá, ugyanezeket, az adott pillanatban lévő folyamnak a forrás IP címére is megállapítja. Tehát egy adott címről érkező átlagos forgalom és összes forgalom kerül megállapításra. Ezeket a dúsító algoritmus az összes folyamra kiszámolja, majd új attribútum formájában a sorok végére fűzi. Ennek a megoldása a következőképpen zajlott:

- A python file elején látható az attribútumokat dúsító algoritmus, avagy a `prev30ipsum` python függvény. Futási idő szempontjából ez a függvény messze túlhaladja bármelyik másik lépést. 25 Gigás file esetében, processzor magjai között párhuzamosítva 6 órás futási idővel kell számolni. Ezt az időt saját 2017-es MacBook Pro laptopomon mértem. 4 magos Intel i7-es processzorral, 16Gb RAM-mal.

- A függvénynek egy objektumot adok át, ami eleinte kettő, különálló változó volt. Azonban számítási optimalizálásnál használatos processzor magjain paralellizáló függvény (`p.map`, ahol a 'p' a Python Multiprocessing könyvtárnak Pool objektuma) egy objektummal kerül meghívásra, ezért volt szükséges a saját függvényemnek is egy objektumban átadnom a lényegében kettő attribútumot. Ez az objektum egy dictionary tömb, amiben szerepel az egész adatbázis egy `pandas DataFrame`-ként tárolva, és egy i futóváltozó, ami a `DataFrame` minden egyes elemén fut végig. Azért van szükség egy külön futóváltozót is átadni a függvénynek, mivel a `DataFrame` minden sorára meghívom az egész `DataFrame`-et, hogy hátrafele tudjon 'iterálni' lényegében amíg el nem éri a fél órával ezelőtti időpontot.

- A függvény kibontja a dictionary-t és kiveszi belőle a fent említett `DataFrame`-et és az adott i elemet.

- A függvény következő lépésben megnézi, hogy az adott elemnek az indexe ezerrel osztható-e, és hogyha igen, akkor kiírja az elem indexét. Ezzel a módszerrel belátható időn belül mindig visszaigazolást ad a program az aktuális adatról, ahol tart. Ezzel megoldva azt a porblémát, hogy jelzés nélkül órákon át fut a program. És emellett el sem teliti a képernyőt, mivel "csak minden ezredik sornál jelez". Ez persze csak a DDoS támadások vizsgálatán igaz, az SSH támadásoknál nem, mivel az SSH támadásnál az adat feldolgozásánál ki kell szűrni minden olyan kapcsolatot, amely nem a 22-es porton kommunikál.

- Ezután a függvény elmenti az adott elemhez tartozó pillanatnyi időpontot. Ezt eltárolja a `current` változóban. Ahhoz, hogy ezt értelmesen meg tudja valósítani a program, a függvény hívása előtt át kell konvertálni a `Timestamp` attribútum elemeit UNIX time kódolásba. Ez lehetővé teszi, hogy kényelmesen, napokat, órákat, perceket adjunk hozzá vagy vonjunk ki az időpillanatokból.

- A függvény kivon 30 percet a `current` változóból, és elmenti a `last` váltózóban.
- Az aktuális elem `Source Address`-jét, azaz a forrás IP-címét elmenti az `sa` változóba.
- A lényegi részhez elérve a függvény a jelenlegi időponttól kezdve végigmegy az egész adatbázison visszafele addig amíg el nem ér a fél órával előtti időponthoz. Ezt az adathalmazt, elmenti egy új `DataFrame`-be, nevezetesen a `newdf`-be.
- Az előző lépésre építve, és az előző lépésből megkapott adathalmazból kiszedi azokat az elemeket, amiknek a forrás IP-címük egyezik. Ezt elmenti egy újabb `DataFrame`-be, most a `newdf2`-be.
- A függvény visszatérési értékként egy listák listáját ad. Ezt minden egyes elemre külön adja vissza. A lista elemei a következők:
    1. Az aktuális fél órában, az azonos IP címekről érkező kommunikációs folyamoknak csomagban mért teljes mérete.
    2. Az aktuális fél órában, az azonos IP címekről érkező kommunikációs folyamoknak csomagban mért átlagos mérete.
    3. Az aktuális fél órában, az azonos IP címekről érkező kommunikációs folyamoknak byteban mért teljes mérete.
    4. Az aktuális fél órában, az azonos IP címekről érkező kommunikációs folyamoknak byteban mért átlagos mérete.
    5. Az aktuális fél órában, az összes folyamnak a csomagban mért teljes mérete.
    6. Az aktuális fél órában, az összes folyamnak a byteban mért teljes mérete.
- A következő lépésben a program megnyitja az aktuális dataszettet. Az UGR'16-os dataszett 3 hónapot ölel fel és az adatok hetenként vannak külön szedve. Ebből a eredeztetve a beolvasási logikát, én is heti bontásban olvastam be az adatokat. Az adatok körülbelül 80Gigabyte-os CSV fileokban voltak. Kivétel volt az Márciusi 3. hét, ami az első hétnek számít az adatbázist tekintve. Ez nagyjából 25Giga volt.
- A megnyitott CSV file-t tízmillió soronként menti el egy `DataFrame`-be. Erre triviálisan azért volt szükség, mivel egyben egy 80 Gigás CSV-t beolvasni a memóriába lehetetlen feladat, hogyha nem áll rendelkezésre közel sem elég memória. A `DataFrame` egy optimalizáltabb adatstruktúra, aminek a tárolásához nem kell ennyi erőforrás. Tehát ha tízmillió soronként elmentem egy adatszerkezetbe és összefűzöm dinamikusan ezeket, akkor végeredményben ugyanazt kapom, csak RAM szempontjából kezelhetőbb úton jutok el odáig.
- SSH támadás vizsgálatnál ezen a ponton történik meg a 22-es Destination Portra való szűrés is.
- Ugyancsak itt történik a Timestamp UNIX time formátumra való konvertálás is.

- Ilyenkor szűröm ki a forgalomból a feketelistázott IP-címeket is. Ilyenkor az UGR'16 saját feketelistázott címkéjét veszem figyelembe.
- Ahhoz, hogy a fentebb említett tanácstalanságot el tudjam kerülni, nevezetes azt, amikor hosszú órákig fut a program anélkül, hogy bármi információval szolgálna futás alakulásáról, a program információval szolgál az adatom hosszáról. Ezt később a dúsító függvényben megjelenítendő aktuális helyzettel összemérve tudok következtetni arra, hogy még nagyjából mennyi van hátra a program futásából.
- Következő lépésben beolvasásra kerül a 3.4 szekcióban leírt saját feketelistázott IP címeket tartalmazó adatbázisom.
- Összehasonlítom a folyamokat tartalmazó adatbázis forrás IP címeit a feketelistázott IP címeket tartalmazó adatszerkezettel. A különbséget elmentem a `res` nevű Dataframe-be.
- A következő blokk a processzor magjain való párhuzamos futtatás céljából vannak jelen. Első körben megvizsgálom a mindenkori számítógép processzorának a magjait, amit elmentek egy változóba.
- Létrehozok annyi `Pool`-t, amennyi magom van. Ezek a `Pool`-ok lényegében ekvivalens futási szálak. Minden magra jut egy.
- Deklarálásra kerül az a Dictionary lista, amiről korábban beszéltem. Erre a párhuzamos számítás miatt van szükség, pontosabban a `p.map` függvény miatt, ami csak egy attribútumot tud fogadni a futtatandó függvény megadása mellett.
- Meghívásra kerül az oly sokat emlegetett függvény párhuzamosítva. A futási idő emiatt 6 óra lett a 25Gigabyte-nyi kezdő CSV mellett. Ez az idő párhuzamos futás nélkül nagyjából egy nap lehetett a saját 2017-es MacbookPro laptopomon.
- Miután a függvény lefutott, visszatér hat attribútumból álló listák listájával. A listák listái az eredeti DataFrame új oszlopai lesznek.
- Bármilyen olyan adatot, amely NaN értékkel szerepel, konvencióképpen 0-val helyettesítem.
- A programban ezen a ponton vizsgáljuk meg a támadásokhoz tartozó folyamokat, amik SSH támadások esetében az UGR'16-os datasettnél külön CSV-ben találhatóak. Egy ilyen részletet mutat a 5.1 ábra. Erre az adathalmazra is lefuttatásra kerül a fent említett dúsítás.
- Következőkben mintát veszek a nagy adatbázisból random módon. Pontosan annyit, amennyi flow szerepelt a támadásoknál. Erre azért van szükség, mivel a Machine Learning metódusok tanítási fázisban el tudnak tolódni egy irányba, hogyha az ember nem figyel. Hogyha túlnyomóan sok adatot ad az ember egy fajta adatból, könnyen az algoritmus elfogult (biased) tud lenni. Ennek eredményeképpen kapok 2 darab, azonos elemszámú DataFramet. Az egyik a

2016-03-18 19:07:38	304.844	37.189.69.206	42.219.156.231	22	52765	TCP	.AP...	0	0	602940	878364400	anomaly-sshscan
2016-03-18 19:09:18	99.436	37.189.69.206	42.219.156.231	22	52765	TCP	.AP..F	0	0	145452	210837984	anomaly-sshscan
2016-03-18 19:10:13	0.560	37.189.69.206	42.219.156.231	22	52962	TCP	.AP.S.	0	0	13	3057	anomaly-sshscan
2016-03-18 19:10:43	10.312	37.189.69.206	42.219.156.231	22	52962	TCP	.AP..F	0	0	37855	6955484	anomaly-sshscan
2016-03-18 19:11:23	17.164	37.189.69.206	42.219.156.231	22	52974	TCP	.AP.SF	0	0	34850	6736949	anomaly-sshscan
2016-03-18 19:12:44	15.192	37.189.69.206	42.219.156.231	22	52983	TCP	.AP.SF	0	0	37487	6916425	anomaly-sshscan
2016-03-18 19:13:20	14.948	37.189.69.206	42.219.156.231	22	53002	TCP	.AP.SF	0	0	35695	6797353	anomaly-sshscan

5.1. ábra. SSH támadások eredeti formája

támadások alatti folyamokat tartalmazza dúsítva, a másik pedig a normális viselkedéshez tartozó folyamokat tartalmazza dúsítva. Ezt a kettőt összefűzöm.

- Ezután a rövid adathalmazon lefuttatok egy `One-Hot Encoding` algoritmust. Ezt effektíven lehet használni az én esetemben a TCP flageket tartalmazó oszlopban. Itt az összes TCP flag egy attribútumban szerepel. Ezeket felbontja a One-Hot Encoding, és elemenként hoz létre új attribútumokat, amik akkor kapnak 1-es értéket, ha az eredeti oszlopban az adott flag szerepelt, ellenkező esetben 0-s értéket vesznek fel. Mivel az eredeti flageket tartalmazó oszlop redundánssá válik, ezért azt eldobom.

- Végezetül a program az eredményt kiírja egy CSV file-ba.

16-03-18 19:07:38	304.844	37.189.69.206	42.219.156.231	22	52765	TCP	0	0	602940	878364400	anomaly-sshscan	1003461	501730.5	1483726225	741863112.5	1003461	1003461	1.0	0.0	1.0	0.0	0.0
16-03-18 19:09:18	99.436	37.189.69.206	42.219.156.231	22	52765	TCP	0	0	145452	210837984	anomaly-sshscan	1606401	535467.0	2362090625	787363541.6666666	1606401	1606401	0.0	0.0	0.0	0.0	1.0
16-03-18 19:10:13	0.56	37.189.69.206	42.219.156.231	22	52962	TCP	0	0	13	3057	anomaly-sshscan	1751853	437963.25	2572928609	643232152.25	1751853	1751853	1.0	0.0	0.0	0.0	0.0
16-03-18 19:10:43	10.312000000000001	37.189.69.206	42.219.156.231	22	52962	TCP	0	0	37855	6955484	anomaly-sshscan	1751866	350373.2	2572931666	514586333.2	1751866	1751866	0.0	0.0	0.0	0.0	1.0
16-03-18 19:11:23	17.164	37.189.69.206	42.219.156.231	22	52974	TCP	0	0	34850	6736949	anomaly-sshscan	1789721	298286.8333333333	2579887150	429981191.6666667	1789721	1789721	1.0	1.0	1.0	0.0	1.0
16-03-18 19:12:44	15.192	37.189.69.206	42.219.156.231	22	52983	TCP	0	0	37487	6916425	anomaly-sshscan	1824571	260653.0	2586624099	369517728.4285714	1824571	1824571	0.0	0.0	0.0	0.0	1.0
16-03-18 19:13:20	14.948	37.189.69.206	42.219.156.231	22	53002	TCP	0	0	35695	6797353	anomaly-sshscan	1862058	232757.25	2593540524	324192565.5	1862058	1862058	0.0	0.0	0.0	0.0	1.0

5.2. ábra. SSH támadások dúsítás utáni formája

Összehasonlítva a 5.1 és a 5.2 ábrákat egyből láthatóvá válik a dúsítás mértéke. Látható, hogy eltűnt a TCP flageket tartalmazó oszlop, helyette megjelnt a sor végén öt darab oszlop. Ezek a One-Hot encoding eredményei. Megjelent továbbá a hat oszlop, amely a hálózat alakulását mutatja.

A fontosabb algoritmusok, és megvalósításuk a fentebb leírtak alapján:

```
def prev_30_ip_sum(obj):
    df = obj['df']
    i = obj['i']

    # For Followability
    if i % 1000 == 0:
        print(i)

    # Current time from current row
    current = df.loc[i, 'ts']

    # Timestamp of last 30 minutes
    last = current - timedelta(minutes=30)

    # Current source address
    sa = df.loc[i, 'sa']

    # New dataframe for timestamp less than 30 min and same ip as current one
    new_df = df[(last <= df['ts']) & (current > df['ts'])]
    new_df2 = new_df[new_df['sa'] == sa]

    # Return sum and mean
```

```
    return new_df2['pkt'].sum(), new_df2['pkt'].mean(), new_df2['byt'].sum(), new_df2['byt'].mean(),
     new_df['pkt'].sum(), new_df['byt'].sum()
```

5.1. lista. Adatbázist dúsító függvény

```
# Count the CPU Cores
cores=mp.cpu_count()

# Create the multiprocessing pool
pool = Pool(cores)

# Create Dict
dictarray = [{'df': res, 'i': i} for i in res.index]

# process the DataFrame by mapping function to each df across the pool
with Pool(cores) as p:
    result = p.map(prev_30_ip_sum, dictarray)
```

5.2. lista. Párhuzamos függvényhívás

```
# One-hot encoding. MultiLabelBinarizer
y = df3['flg']
one_hot = MultiLabelBinarizer()
df2 = pd.DataFrame(one_hot.fit_transform(y), columns=one_hot.classes_)
df2 = df2.drop(['.'], axis=1)
df3 = df3.drop(['flg'], axis=1)
df3 = df3.join(df2)
```

5.3. lista. Adatbázist dúsító függvény

5.3. Adatbázis méretéből adódó problémák és megalkuvások

Ahogy korábban említettem az UGR'16-os adatbázis heti bontásából az első a legrövidebb. Ez egy nagyjából 25Gb-s CSV-t jelent. Erre a file-ra a feldolgozási futási idő meghaladja a 6 órát, úgy, hogy erőteljesen optimalizálva lett a program. Nem állítom, hogy ez az állapot a legjobban optimalizált állapot, amit el lehet érni ebben a témában, de jelen körülmények között ez a legtöbb, amit futási idő csökkentés szempontjából ki tudtam hozni a feladatból.

A soron következő héthez tartozó CSV file több mint 80Gb. Több mint egy napon keresztül futott a gépen, és még közel sem volt vége. Emiatt fontos döntéseket kellett hoznom, amiket a továbbiakban ki is fejtek.

Redukáltam minden hétnek a feldolgozandó adatját csupán 20 millió sorra, ami körülbelül 2Gb-os CSV file-nak felel meg. Erre a futási idő elenyésző.

Ezzel a megoldással élve magabiztosan tudom állítani, hogy a feladat érdemi része nem szenvedett kárt. A teljes adatbázis, a közel egy Terrabyte-nyi adat helyett 20 Gb-nyi adaton futott le a feldolgozási algoritmus. Azonban mivel erőteljesen leszűröm a végső lépésben az adott adatot, amikor ugyanannyi mintát veszek a normális folyamból, mint amennyi támadáshoz tartozó folyam van, az adat 99%-a elveszik végérvényesen. Hogyha egymást követő folyamokat tartok meg, akkor a dúsítás is megtartja értelmét, ezzel ellentétben, hogyha random módon dobálnám ki az adat 98%-át, akkor hamis dúsított eredményt kapnék.

Április első hetétől kezdve az UGR'16 dataszettben erőteljesen SSH scanning anomáliák találhatóak. Hogyha csak a címkézett anomáliákat válogatjuk ki, ez hozzávetőlegesen 500Mb-os CSV file-t eredményez, nagyjából öt millió sorral. Egyértelműen az algoritmusom ezen a ponton is egy teljes napi futás után is csak a 300

ezredik sornál tartott. Tehát az anomáliákat tartalmazó CSV file beolvasását is redukálnom kellett. 100 ezer sort olvastam ezek után be. A futási idő természetesen nagyban csökkent, körülbelül 5 percre. Azonban ezen a ponton is szerkesztésre szorult a program, mivel a 100 ezer SSH anomália lényegesen több volt, mint a normális forgalomnak a folyamából kiszedett SSH kapcsolat. Szintén az SVM Machine Learning algoritmus részrehajlásának elkerülésére, a dúsítás után, ugyancsak random módon kiválogatok 500 elemet mind a két folyamból.

Ezen tapasztalatokat figyelembevéve döntöttem az erőteljes adatredukálás mellett. A továbbiakban ezek a megfontolások mellett futtattam a programom.

6. fejezet

Support Vector Machine

A Support Vector Machine egy klasszifikáló Machine Learning modell. A működési elve, hogy az SVM keres egy metsző síkot az adatban a kettő klasszifikációs osztály között, úgy hogy a távolság a sík és a két klasszifikációs osztályhoz tartozó legközelebbi elemek közötti távolság a maximális legyen[2].

Ez a megközelítés az osztályozási hiba minimalizálásán alapszik, mintsem az optimális klasszifikáláson. Az SVM-ek jól ismertek a csoportosítási képességükről, és kifejezetten hasznosak, amikor az attribútumok száma, m nagyobb mint az adatpontok n ($m \gg n$). Amikor a két osztály nem választható el egyértelműen, akkor segéd térrészek kerülnek bevezetésre, amik egy laza vagy puha határként szolgálnak. Az ezekben található adatokhoz súlyokat rendel az SVM algoritmus. A határsík kiszámolására egy kvadratikus optimalizálással kerül sor, így az SVM a gyorsabb algoritmusok közé sorolható, még akkor is, ha sok atributuma van az adatoknak.

A határsík kiszámolásakor több fajta kiindulási kernelt lehet használni: lineáris, polinomiális, tangens hiperbolikus vagy Gaussi RBF kernelt. Az SVM fundamentálisan egy bináris osztályozó metódus, hogyha több osztályt akarunk implementálni, azt úgy lehet megoldani, hogy a két alap osztályban kell külön még SVM-eket definiálni.

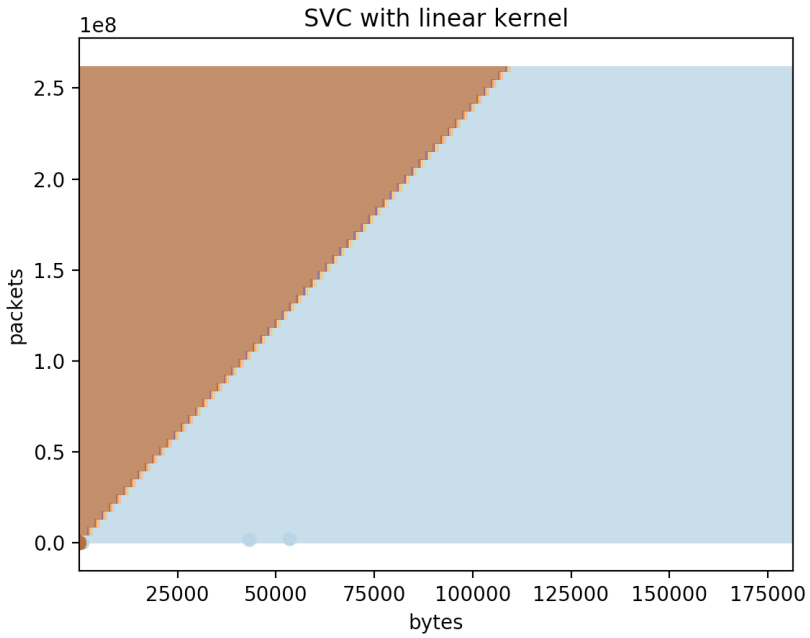
Önálló Laboratórium tárgy féléve alatt három darab, különböző SVM-et implementáltam.

6.1. Első verzió

Az SVM első implementációja egy egyedülálló python osztály volt. Ennek a célja nem a használat volt, hanem a belső működésnek a szélesebb megértése volt. Az implementáció szorosan követte a [15]-ban leírtakat. Ez a megvalósítás nem használ fel semmilyen előre elkészített SVM algoritmust, nem kell külön könyvtárat importálni hozzá. A program a semmiből építi fel az SVM osztályt. Mint említettem, ennek a lényege teljes egészében a belső működés jobb megértése volt, nem az optimalizálás.

6.2. Második verzió

Az SVM második implementációja egy széles körben elterjedt python library-re épült. Ennek a bemenete a vizsgálandó adathalmaz releváns attribútumjai, és kimenete egy diagramm, amely ábrázolva mutatja meg a Support Vector Machine



6.1. ábra. SVM klasszifikáció

klasszifikálásának eredményét. Fókuszomban a szintetikus SSH támadás detektálása volt, amelyhez többnyire a tanulmányban leírtakat követtem.

Először az adathalmaz két attribútumát választottam ki, modell működésének tesztelésére: a packet számot, és packet méretet. Ez a kettő természetesen nem elegendő, hogy eredményesen detektáljunk SSH támadásokat annak tulajdonságai miatt. De elindulásnak elfogadhatónak bizonyult. Ahhoz, hogy értelmes adatokat kapjon az SVM modellem, kiszűrtem a Datasetből azokat a folyamokat, amelyek SSH kéréshez tartoznak. Ehhez a Destination Portra kellett szűrnöm, pontosabban a 22-es portra. Az SVM-nek címkézett adatot kell szolgáltatni, hogy rá tudjon tanulni a klasszikálásra. Ez a mi adatunknál rendelkezésre áll, azonban String cimkékkel nem tud mit kezdeni. Ennek kiküszöbölésére átírtam a címkét egy jelzőbitre, aminek 0-s értéke azt jelzi, hogy a folyam egy normális tevékenységhez tartozik, míg az 1-es érték azt, hogy egy káros tevékenységhez. Ezek után megadtam az SVM bemenetének a jelen esetben vizsgálandó, fent említett attribútumokat. Az SVM program lefutása után a kimeneten megjelent a várt grafikon az SVM klasszifikálásával. Ezt a grafikont látjuk az 6.1. ábrán.

Az eredmény kiértékelése nem triviális feladat. Ha ráközelítenénk az ábrára, azt látnánk, hogy nagyon sok a helytelen klasszifikálás. Nem egyértelmű, hogy az ábrán látott klasszifikálás - barna és kék síkrész - egy jó klasszifikálás. Jobb ráközelítés mellett látható az SVM-re jellemző puha határvonal.

6.3. Harmadik verzió

A harmadik SVM verziója implementálásban is és kimenetben is eltér az első kettőtől. Szintén egy bejáratott python libraryt vettem alapul az SVM modul elkészítéséhez, azonban mégis egy érdemileg más SVM-et készítettem. Ennek a kimenete már nem grafikus volt, hanem számszerűsített eredményt adott az SVM klasszifi-

kálás "jóságára". Az eredmény egy úgynevezett `Confusion Matrix`, ami a helyes és helytelen predikciókat mátrix formában adja meg. Az "Y tengelyen" a valós címke, "X tengelyen" a klasszifikált címke szerepel. Tehát a Mátrixból ki lehet olvasni, hogy egy valójában támadást az SVM támadásnak klasszifikált-e vagy nem, és fordítva. Ezáltal kapunk hamis és jó klasszikálási eredményeket. Egy ilyen Confusion Matix látható a 6.2. ábrán.

		Actual Class	
		0	1
Predicted Class	0	82	1
	1	75	194

6.2. ábra. Confusion Matrix TimeDelta attribútum nélkül

Ahhoz, hogy ezt a confusion mátrixot megkapjuk, ugyanazzal a bemenő feltételekkel kellett futtatnom a harmadik SVM-et, mint a másodikat. Tehát csak a Packet számot, és méretet vesszük figyelembe. A már számszerűsített eredményt könnyebben ki lehet értékelni.

Azt látjuk, hogy a jó predikciók, azaz a jobb felső és bal alsó sarokban elhelyezkedő számok egészen magasak. Azonban a fals predikálások száma is relatíve magas. Ez azért van, mivel nem teljes egészében az SSH támadásokra jellemző attribútumokat vettük figyelembe, hanem csak kettő attribútumot.

A Dataset köré írt tanulmányból kiderül, hogyha közelről megvizsgálja az ember a Datasetben megjelenő SSH támadásokat, arra lehet figyelmes, hogy jellemzően időben közel egymáshoz történnek ezek a támadások. Tehát a követhező lépésben meg kellett határozni, hogy milyen közel indultak egymáshoz az SSH kéréseknek a folyamai.

Az adatbázisban a Timestamp mezőt átváltottam Unix Time-ra, majd abból tudtam TimeDeltát számolni. Minden egyes folyamhoz az azt megelőző folyammal hasonlítottam össze, és a létrejött TimeDelta attribútumot egy cellában tároltam el. Hogyha ezt az új attribútumot is az SVM bemenetére adjuk, pontosabb predikcióban reménykedhetünk. Ennek az ereménye látható a 6.2. ábrán.

		Actual Class	
		0	1
Predicted Class	0	102	28
	1	39	183

6.3. ábra. Confusion Matrix TimeDelta attribútummal

Az eredménymátrixon látható, hogy a jó predikciók száma megnőtt, és a fals predikciók száma összességében lecsökkent. Tehát megállapítható, hogy egy pontosabb klasszifikációt kaptunk.

Természetesen ez sem elég az SSH támadások pontos detektálására.

6.4. Python kód

Ismertetem a harmadik SVM implementációt, amelynek eredményéül a mátrixot kaptam.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix

df = pd.read_csv('/Users/noodlot/Desktop/april.week2teach1000mergedssh.csv', header = None, names=['
    te','td','sa','da','sp','dp','pr','flg','fwd','stos','pkt','byt','lbl','delta'])

teach1 = df[['pkt','byt','delta']]
teach1.shape[1]
pd.to_numeric(teach1['pkt'])
pd.to_numeric(teach1['byt'])

X = teach1.to_numpy()

y = df['lbl']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)

svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)

y_pred = svclassifier.predict(X_test)

print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

A kód elején találhatóak a beimportált python könyvtárak. Utána beimportálom egy Pandas Dataframe struktúrába az előre formázott adathalmazom. Ezek után még egy kevés formázást hajtok végre kettő attribútumon, hogy biztos szám formátumú legyenek. Majd kiválasztom az SVM nek beadandó adatokat, és az eredményeket tartalmazó oszlopot. Ezek után megadom a tanuló és teszt adathalmazok méretbeni arányát. Az SVM megalkotása után pedig kiírom a confusion mátrixot, majd egy riportot a klasszifikálásról.

6.5. Végleges verzió

A végleges verzió implementálása a Support Vector Machine-nek ugyanaz maradt, mint a harmadik verzióban, azonban az adat, amin a betanítást elvégeztem, teljes mértékben megváltozott. A harmadik verzióban az aktuális folyamnak a packetjét és a byte-bani méretét vettem alapul, plusz a timedeltát, ami két folyam között eltelt idő. Azonban ennyi adat fundamentálisan nem tudja az SSH támadásokat detektálni. Sokkal több információra van szükségünk. Relevánsabb információra. Ezeknek az attribútumoknak az egész kommunikációról kell mondaniuk valamit, mert kontextus nélkül, egy kiragadott folyam legtöbb esetben nem hordoz magában semmilyen információt. Tehát ezen megfontolásból, és a támadások attribútumainak figyelembevételével alkottam meg az adat dúsító algoritmusomat, amely releváns információval tud szolgálni az addigi hálózati történésekről. Ezek az attribútumok a 5. fejezetben leírtak alapján a folyamokat 30 percig visszamenőlegesen vizsgálja az

összes byte és packet szempontból, valamit az egyes IP címek felől érkező forgalmat is számolja. Az összes forgalmat, és az átlagos folyam méretet is. Ezekkel a mértékekkel hatásosabban lehet egy folyamot összehasonlítani a többivel.

6.5.1. Futtatás

6.5.1.1. Otthoni környezet

A saját 2017-es Macbook Pro laptopomon, amiben egy négy magos Intel i7 van és 16 Gb RAM, futtatam a Support Vector Machine betanítására szolgáló programot 1206 SSH anomália adaton. Ez az április első hetéig bezárólagos SSH anomáliákat és az azonos számú normális veselkedéshez tartozó folyamokat tartalmazó adathalmaz volt. Azért csak az április első hetéig tartó adatokkal számoltam, mivel már így is a gépem kapacitását erőteljesen meghaladtam, ezért valahol meg kellett húznom a határt.

Ez a program, az 1206 darab adattal 3 napig futott, azonban közel sem volt még vége, így a program terminálása mellett döntöttem.

Azután random módon kiválogattam 50 elemet a már amúgy is erőteljesen leszűrt adathalmazból, és azon futtattam a programot. Ez a konstrukció 2 és fél napi futás után lett terminálva. Ekkor döntöttem amellett, hogy regisztrálok a Google Cloud Computing Platformjára.

6.5.1.2. Google Cloud Computing Platform

A Google Cloud egyik szolgáltatása, hogy az erőforrásait használatra kínálja bárkinek, aki hajlandó fizetni érte. Ez lényegében egy VM (Virtual Machine), aminek a regisztrációjakor ki tudja választani az ember, hogy mire szeretné használni. Egyik lehetőség a számításra optimalizált szerver. Erre regisztráltam. Néhány alap konfigurációs beáll1tás után lehetősége nyílik az embernek 'be SSH-zni' a VM-re, és így használni azt. Telepítettem a futtatáshoz szükséges dependenciákat, és futtattam első körben az 50 adatra az SVM betanítását.

A program futása fél percig sem tartott. Az eredményeket a 7 fejezetben mutatom be. Ez a tény nagyon jókor jött, hiszen a saját gépemen több hétnek megfelelő időn keresztül futottak a programjaim. Ezek után futtattam az 100 adaton a programot. Ennek a futási ideje azért jelentősen megnőtt. A 200 adatra már nem is sikerült futtatni, mivel olyan mértékben megnőtt a futási idő.

7. fejezet

Eredmények

7.1. UGR'16 - SSH Brute-Force detektálás

A Google Cloud Computing Platformon való futtatás bizonyult a legjobb módszernek, túlnyomóan azért, mert viszonylag rövid határidőn belül eredményt adott.

7.1.1. UGR'16 - SSH - 50 elemű adathalmaz

Az első futtatás egy 50 elemű adathalmazon zajlott, a 6.5.1 alfejezetben leírtak alapján. A futtatás kifejezetten rövid volt a többi esethez képest. Azonban érdemesnek tartottam ennek az eredményét is kiértékelni.

A 7.1 ábrán látható az eredményül kapott Confusion Mátrix. A Support Vector Machine betanítására, és tesztelésére az 50 adatból a 80%, azaz 40 eset a betanulást szolgálja, a maradék 20%-on, tehát 10 adaton pedig a tesztelést végzi az adat. Ezt a besorolást is véletlenszerűen hajtja végre a program.

		Actual Class	
		0	1
Predicted Class	0	3	0
	1	3	4

7.1. ábra. Confusion Matrix UGR'16 SSH 40/10 elemre

Ha az eredményt értelmezzük, azt látjuk, hogy 70%-os pontossággal sorolja be valósan az adatot a Support Vector Machine. Valós támadás 3 darab volt, valósan normális adat pedig 4. Ezeket sorolta be pontosan a program. Azonban 3 darab normálisnak vélt forgalmat nem normálisnak, hanem támadásnak sorolt be. Ez az úgynevezett fals pozitív.

Ez az eredmény önmagában nem a legrosszabb eredmény, hiszen egy anomália detektálásnál, főleg, hogyha kritikus rendszerekről beszélünk, a fals pozitívak inkább jobbak, mint a fals negatívak. Ez a klasszifikáló modell a fele normális forgalmat támadásnak osztályozott, de egyetlen egy támadást sem engedett át a rendszer. Ez természetesen ilyen kevés adatnál várható eredmény, hiszen nem remélhetünk pontos

eredményt, hogyha ilyen kevés adatot használunk a betanításra. Minél több adatunk van betanításra, annál pontosabb lesz a modellünk a végén.

7.1.2. UGR'16 - SSH - 100 elemű adathalmaz

A következő eredmény egy hasonló módszerrel kapott, 100 elemű adathalmazon készült. Az eredmény teljesen az elvárt módon alakult.

		Actual Class	
		0	1
Predicted Class	0	7	1
	1	1	11

7.2. ábra. Confusion Matrix UGR'16 SSH 80/20 elemre

Látható, hogy a pontossága a modellnek, ezzel a random módon kiválasztott 100 elemmel, jócskán megnőtt. 100 elemű halmaznak 80%-a lett tanításra, a maradék 20%-a lett tesztre használva hasonló módon. A 20 teszt adatból 18-at pontosan klasszifikált a modell. Ez egy 90%-os pontosságnak felel meg, ami jobb eredmény, mint várt volna az ember csak 100 elemtől. Persze ebben az is benne lehet, hogy szerencsésen választott a program adatot mind betanításra, mint tesztre.

Érdemes megjegyezni, hogy jelen pillanatban a rendszer produkált egy Fals Negatív ereményt. Ez nem a legjobb hír, az előző esetnél leírtak miatt.

A futtatási idő itt már lényegesen több volt, mint az 50-es esetben. A futási idő 2x nagyobb adat mellett, egyértelműen nem a 2x-ese lett.

7.2. UGR'16 - DDoS támadás detektálás

A DDoS támadás detektálása az SVM modell betanítása hasonlóképpen zajlott, mint az SSH támadás esetén, annyi különbséggel, hogy nem volt a 22-es portra szűrve az adatfolyam. Az eredmények nagyon lassan születtek meg az adat dúsítása miatt, plusz az SVM modell betanítása is igen hosszú ideig tartott több adat mellett.

7.2.1. UGR'16 - DDoS - 50 elemű adathalmaz

50 adat trviviálisan kevés. Szinte értelmetlenül kevés normális eredmény leszűrésére.

Az eredmény azonban még így sem volt elkeserítő. 70%-os pontosság ennyi adat mellett nem rossz eredmény.

7.2.2. UGR'16 - DDoS - 100 elemű adathalmaz

Az eredmény 100 elemre 75%-os pontosságot mutat. Ez egy fokkal jobb, mint az előző, egyértelműen a dupla mennyiségű adat miatt.

		Actual Class	
		0	1
Predicted Class	0	4	1
	1	2	3

7.3. ábra. Confusion Matrix UGR'16 DDoS 40/10 elemre

		Actual Class	
		0	1
Predicted Class	0	8	4
	1	1	7

7.4. ábra. Confusion Matrix UGR'16 DDoS 80/20 elemre

7.3. CICIDS2017 - DDoS támadás detektálás

7.3.1. CICIDS2017 - DDoS - 100 elemű adathalmaz

A CICIDS2017-es adatbázison való SVM modell tanításánál nem volt különösebb dolgom, csak az SVM számára feldolgozhatóvá tenni az adatot, és megvágni azt. A támadás az adatban a pénteki napon történt.

A futtatások gyorsabbak voltak, mint az UGR'16-os dataszetnél, az én általam dúsított formájában.

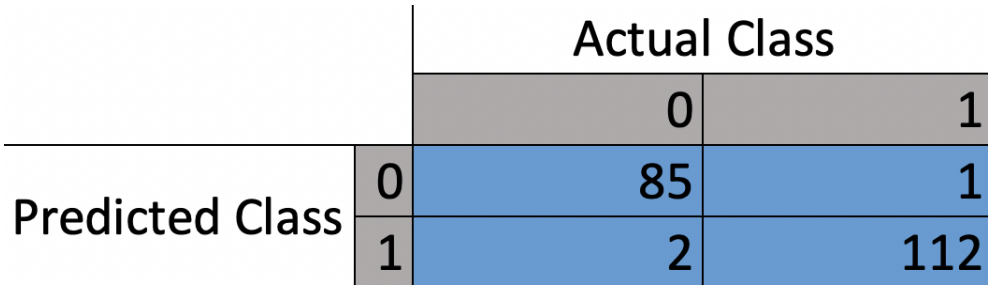
		Actual Class	
		0	1
Predicted Class	0	7	3
	1	1	9

7.5. ábra. Confusion Matrix CICIDS2017 DDoS 80/20 elemre

Az első eredmény egy 80%-os pontosságú modell lett, amit 100 elemre kaptam. Ez egyelőre rosszabbnak bizonyul mint a saját modellem az UGR'16-on. De nem szabad elfelejteni, hogy ez ugyancsak nagyon kevés adat ezidáig.

7.3.2. CICIDS2017 - DDoS - 1000 elemű adathalmaz

Természetesen az 1000 elemű halmazon a futtatás meglehetősen hosszú ideig tartott, de az eredmény pontos és bíztató lett.



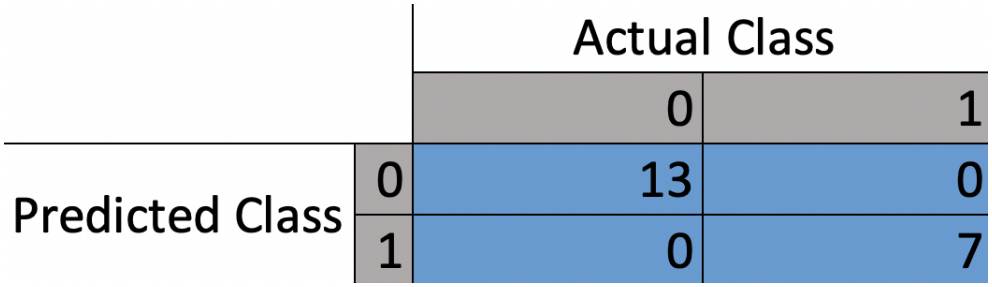
7.6. ábra. Confusion Matrix DDoS 800/200 elemre

200 tesztesetből helyesen sorolt be 197-et. Ez 98,5%-os pontosságot mutat. Mindössze 1 darab Fals Negatívval és 2 darab Fals Pozitívval a rendszerben, ez egy nagyon jó arány. Természetesen ez sem elegendő adat, de már így is nagyon jó eredményeket ad a Support Vector Machine Modell.

7.4. CICIDS2017 - SSH Brute-Force támadás detektálás

7.4.1. CICIDS2017 - SSH - 100 elemű adathalmaz

A keddi napon került futtatásra az SSH Brute-Force támadás a CICIDS2017 - es dataszettben. Először itt is 100 adaton tanítottam az SVM-et.



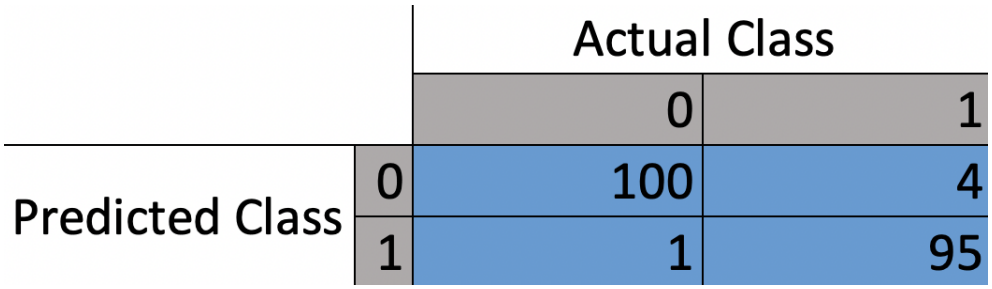
7.7. ábra. Confusion Matrix SSH Brute-Force 80/20 elemre

100%-os pontosságú modellt kaptam eredményül. Jó adatokat kapott az SVM, és jó adatok kerültek tesztelésre. Ilyen pontosságú eredmény nem várható el, több adat mellett.

7.4.2. CICIDS2017 - SSH - 1000 elemű adathalmaz

1000 elemre is jól látszik az SVM eredményessége.

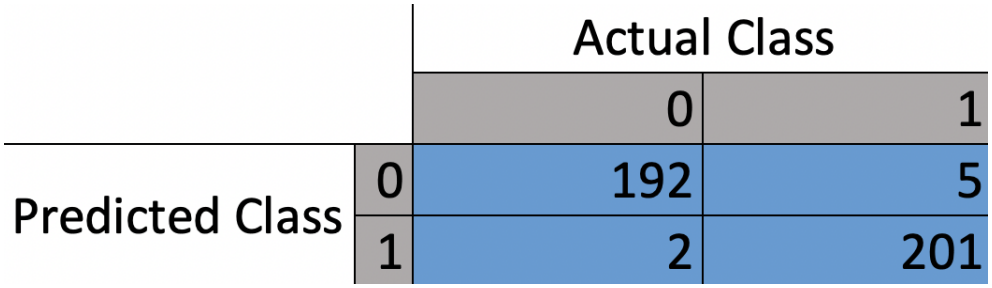
97%-os pontossággal volt képes helyes eredményt adnia. 4 alkalommal osztályozott hamisan negatívra és 1 alkalommal hamisan pozitívra. A szakirodalomban is ilyen, és ehhez hasonló eredmények születtek. Azonban mint már leírtam többször, egy pontosabb klasszifikáláshoz ezek az adatmennyiségek nem elegendőek.



7.8. ábra. Confusion Matrix SSH Brute-Force 800/200 elemre

7.4.3. CICIDS2017 - SSH - 2000 elemű adathalmaz

2000 elemre egyre pontosabbnak tűnik a rendszer.



7.9. ábra. Confusion Matrix SSH Brute-Force 1600/400 elemre

98%-os pontosságot produkált az SVM 2000 adatra. 5 alkalommal osztályozott hamisan negatívra és 2 alkalommal hamisan pozitívra. Ez nagyon kicsi eltérés összességében az 1000 adathoz képest, holott jelen esetben dupla akkora volt a teszt adat mennyisége. Bátran tudom állítani, hogy a pontosság ennél csak jobb lenne több adatra.

8. fejezet

Összefoglalás

A szakdolgozatomban bemutattam az Anomália Detekciót, és azon részeit, amik a szakdolgozatom számára relevánsak voltak. Ismertettem a Support Vector Machine modellt, amit felhasználtam káros forgalmak detektálására.

A betanítási folyamathoz 2 darab adatbázist használtam fel, az egyik az UGR'16, a másik a CICISD2017 volt. Az UGR'16 hoz az adatot dúsítaniom kellett, aminek a teljes menetét leírtam a 5. fejezetben. A CICIDS2017 adatbázishoz azonban nem kellett különösebben pluszt hozzátennem, ez az adatbázis már dúsítva volt.

Az UGR'16 dúsítása illetve SVM betanítása és tesztelése nagyon hosszú futási idővel járt. Emiatt a Google Cloud Platformban virtuális környezetben futtattam a programokat, amivel kicsit le tudtam faragni a hosszú időt, de sajnos sok esetben még így sem sikerült sok adattal betanítani az SVM-et, vagy dúsítani az adatbázist. Emiatt az UGR'16 adatbázison való tanításnál csak nagyon kevés adatot tudtam felhasználni. Ennek eredményeképpen ez a modell jelenlegi állapotában közelítő eredményt ad, illetve egy tendenciát mutat.

A CICIDS2017 adat esetében azonban sokkal több adaton sikerült a tanítás. Bár még mindig közel sem elegendő ez az adatmennyiség, de már lényegesen jobb eredmények születtek.

8.1. Eredmények összefoglalása

A kevés UGR'16-os adaton SSH Brute-Force detektálásnál a modell 90%-os pontosságot mutatott. Ez egy meredeken javuló eredmény pontosság szempontjából. Véleményem szerint, még több adatra a pontosság 95% körül alakulna, vagy akár még ennél is jobb lenne.

DDoS támadás detektálásnál ez a végső eredmény 75% volt, ami annak köszönhető, hogy minden típusú forgalom szerepelt az adatbázisban, és a különböző DDoS technikák különböző szignatúráira nem elég átfogóan terjedt ki a tanítási folyamat.

Azonban CICIDS2017 adatbázison való taníttatás mind DDoS, mind SSH támadás viszonylatában szakirodalom szerinti eredményt kaptam. 97-99%-s pontosság nagyon jó eredménynek számít. Ez mindenképpen a szerzők által elvégzett dúsításnak köszönhető.

8.2. További lehetőségek

Első és legfontosabb dolog, egy még jobb eredmény elérése érdekében, hogy a futtatást jobban kell megvalósítani mind erőforrás-kihasználtság szempontjából, mind program optimalizálás szempontjából. Ez lehetővé tenné, hogy ne csak csekély mennyiségű adatra tudjam elvégezni a tanítást.

Az adatdúsító algoritmust kibővíteném, optimalizálnám, logolnám az egyes támadási típusokra jellemző szignatúrákat, nem csak a hálózat alakulásához mérném hozzá az adott folyamot.

Bőviteni lehetne a programot sokkal több támadásra, vagy esetleg más fajta Machine Learning módszert alkalmazni különböző támadások deketálására, annak függvényében, hogy melyik milyen eredményt ad különböző esetekben.

Köszönetnyilvánítás

A fenti szakdolgozat nem jött volna létre a Paripa Program nélkül. Köszönöm a konzulenseimnek az odaadó munkájukat és segítségüket, főképpen Lestyán Szilviának!

Legnagyobb köszönet illeti a feleségemet, Nikit, aki fáradhatatlanul támogatott és helytállt gyermekünk nevelésében, amikor én jelen szakdolgozatomon dolgoztam.

Köszönöm!

Irodalomjegyzék

[1] Abuse.ch: Domain and ip blacklist: abuse.ch. URL https://abuse.ch/.

[2] Anna L Buczak – Erhan Guven: A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications surveys & tutorials*, 18. évf. (2015) 2. sz., 1153–1176. p.

[3] Varun Chandola – Arindam Banerjee – Vipin Kumar: Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41. évf. (2009) 3. sz., 1–58. p.

[4] Cisco: Cisco netflow v9. URL https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html.

[5] david: Global anomaly detection market 2020 research strategies and forecasts to 2027. URL https://bcfocus.com/global-anomaly-detection-market-2020-research-strategies-and-forecasts-to-2

[6] Canadian Institute for Cybersecurity: Intrusion detection evaluation dataset (cic-ids2017). URL https://www.unb.ca/cic/datasets/ids-2017.html.

[7] Amirhossein Gharib – Iman Sharafaldin – Arash Habibi Lashkari – Ali A Ghorbani: An evaluation framework for intrusion detection dataset. In *2016 International Conference on Information Science and Security (ICISS)* (konferenciaanyag). 2016, IEEE, 1–6. p.

[8] Laurens Hellemons – Luuk Hendriks – Rick Hofstede – Anna Sperotto – Ramin Sadre – Aiko Pras: Sshcure: a flow-based ssh intrusion detection system. In *IFIP International Conference on Autonomous Infrastructure, Management and Security* (konferenciaanyag). 2012, Springer, 86–97. p.

[9] Impreva.com: Ddos attack types and mitigation methods. URL https://www.imperva.com/learn/ddos/ddos-attacks/.

[10] Thomas H. Davenport Leandro DalleMule: What's your data strategy? the key is to balance offense and defense. URL https://hbr.org/2017/05/whats-your-data-strategy.

[11] Gabriel Maciá-Fernández – Camacho: Ugr '16: A new dataset for the evaluation of cyclostationarity-based network idss. *Computers & Security*, 73. évf. (2018), 411–424. p.

[12] Malwaredomainlist.com: Domain and ip blacklist: Malware domain list. URL https://www.malwaredomainlist.com/.

[13] Nour Moustafa – Jill Slay: Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (MilCIS)* (konferenciaanyag). 2015, IEEE, 1–6. p.

[14] Srinivas Mukkamala – Andrew Sung – Ajith Abraham: Cyber security challenges: Designing efficient intrusion detection systems and antivirus tools. *Vemuri, V. Rao, Enhancing Computer Security with Smart Technology.(Auerbach, 2006)*, 2005., 125–163. p.

[15] Sentdex: Beginning svm from scratch in python. URL https://pythonprogramming.net/svm-in-python-machine-learning-tutorial.

[16] Iman Sharafaldin – Arash Habibi Lashkari – Ali A Ghorbani: Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP* (konferenciaanyag). 2018, 108–116. p.

[17] Spamhaus.org: Domain and ip blacklist: Spamhaus.
URL https://www.spamhaus.org/.

[18] Anna Sperotto – Aiko Pras: Flow-based intrusion detection. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops* (konferenciaanyag). 2011, IEEE, 958–963. p.

[19] Anna Sperotto – Ramin Sadre – Pieter-Tjerk de Boer – Aiko Pras: Hidden markov model modeling of ssh brute-force attacks. In *International Workshop on Distributed Systems: Operations and Management* (konferenciaanyag). 2009, Springer, 164–176. p.

[20] Anna Sperotto – Ramin Sadre – Frank Van Vliet – Aiko Pras: A labeled data set for flow-based intrusion detection. In *International Workshop on IP Operations and Management* (konferenciaanyag). 2009, Springer, 39–50. p.

[21] Sharmila Kishor Wagh – Vinod K Pachghare – Satish R Kolhe: Survey on intrusion detection system using machine learning techniques. *International Journal of Computer Applications*, 78. évf. (2013) 16. sz.