Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
TÁVKÖZLÉSI ÉS MÉDIAINFORMATIKAI TANSZÉK

Offenberger Attila

MULTIVEKTOR DDOS TÁMADÁSOK FELISMERÉSE MÉLYTANULÁSI MODELLEKKEL

KONZULENS

Dr. Orosz Péter

BUDAPEST, 2022

Tartalomjegyzék

Összefoglaló

Napjaink egyik meghatározó kiberbiztonsági kihívása a DDoS fenyegetések elleni hatékony védekezés. A pandémiás időszakban, majd pedig azt követően a támadások nagysága, intenzitása és bonyolultsága is folyamatosan növekedett. Felismerésüket nehezíti a folyamatosan változó forgalmi mintázat, ezért az alkalmazott detekciós algoritmusok hamar elavulhatnak. A problémára megoldást nyújthat a manapság széles körben elterjedt gépi tanulás, amely folyamatos tanulással lehetőséget nyújt arra, hogy a detekciós rendszerek lépést tarthassanak a potenciális fenyegetésekkel.

A dolgozatban egy általam megvalósított DDoS detekciós módszert és annak megvalósítását ismertetem, amely Pythonban készült a TensorFlow könyvtár felhasználásával. A módszer két fokozatú azonosításon alapszik, ahol az első fokozat egy random forest algoritmuson alapuló anomália detektor, amelynek eredménye támadás detektálása esetén átkerül a második fokozatba, ahol a különböző típusú támadási vektorokra tanított neurális hálózatok végzik a támadás-típusok felismerését. A bemenetek feldolgozását szintén Pythonban végeztem a Pyshark könyvtár segítségével, így a későbbiekben a detektor könnyedén csatolható élő hálózatra. Ismertetem a forrásadatok feldolgozásának módszerét, a modellalkotás folyamatát, majd a modellek felépítését és a hiperparaméterek hangolását. Végezetül beszámolok a modell validációjának eredményeiről, melyet független forgalmi adatokon, laboratóriumi körülmények között végeztem.

Abstract

Effective defense against DDoS threats is one of today's defining cyber security challenges. During and after the pandemic, the size, intensity and complexity of the attacks also increased continuously. Their recognition is made difficult by the constantly changing traffic pattern, which is why the used detection algorithms can quickly become obsolete. The solution to this problem can be provided by machine learning, which is widespread today, and with continuous learning, it provides the opportunity for detection systems to keep up with potential threats.

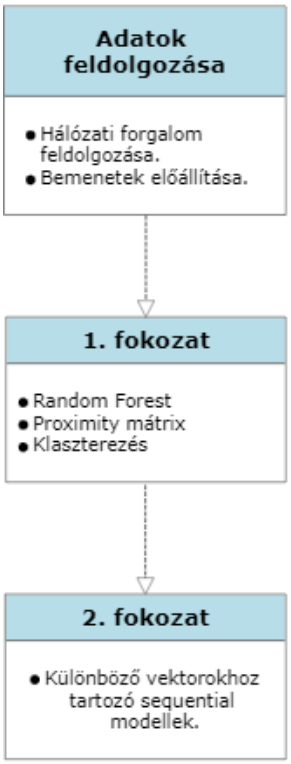
In this thesis, I describe a DDoS detection method I implemented and its implementation, which was created in Python using the TensorFlow library. The method is based on two-stage identification, where the first stage is an anomaly detector based on a random forest model, the result of which is transferred to the second stage when an attack is detected, where multiple neural networks trained on different types of attack vectors perform the recognition of attack types. I also processed the inputs in Python using the Pyshark library, so the detector can be easily connected to a live network later on. I describe the method of processing the source data, the process of model creation, then the structure of the models and the tuning of the hyper-parameters. Finally, I will report on the results of the model validation, which I carried out on independent traffic traces under laboratory conditions.

1. Bevezetés, motiváció

Manapság egy meghatározó kiberbiztonsági terület a DDoS támadások detekciója és elhárítása. A rosszindulatú tevékenység sok problémát okoz a szolgáltatást nyújtó szférában tevékenykedő vállalatoknak vagy az állami szolgáltatásokban. A digitalizáció felgyorsulását eredményező pandémiás időszak, ezen tevékenységek számát is megemelte, mivel sok szolgáltatás átköltözött az online térbe. Ez a folyamat megemelte a támadó felek érdeklődését is. A támadások elterjedésével a bonyolultságuk és gyorsaságuk is nagyban megnövekedett, ami a védekezést nehezíti.

Az ilyen káros tevékenységek elleni védekezés az azonosításnál kezdődik, ezért erre a feladatra próbáltam egy új fajta megoldást létrehozni. Az általam létrehozott DDoS detektor két fokozatból áll, amelyek különféle mélytanulási módszereken alapulnak. A hagyományos módon működő algoritmusok problémája, hogy nehezen tudják követni az egyre gyorsabban változó támadási mintázatokat, ezáltal folyamatos javításokra, változtatásokra van szükség a kódunkban és mindig a támadó fél mögött maradunk egy lépéssel. Ezért választottam a gépi tanulást, mint detekciós módszert, hiszen így ezeket a problémákat ki tudjuk küszöbölni. A hatásos védekezés érdekében fontos, hogy milyen gyorsan tudunk reagálni és emiatt az azonosításnak valósidőben kell működnie.

A kidolgozott módszer arra a hiányosságra szeretne megoldást adni, hogy a felismerés valósidőben történjen és a különböző típusú vektorokat be tudjuk azonosítani, ne csak a támadás tényét. A detektor két fokozatból épül fel. Az első lépésben egy felügyeletmentes random forest modellen alapuló anomália detektor felelős a forgalom elemzéséért, hogy támadás vagy normális forgalom érkezett-e. Az eredményeket felhasználva a második lépésben a különböző fajta vektorokat próbálom azonosítani egy szekvenciális modell segítségével, hiszen az előzőtől eltérő módon itt ismerjük a konkrét mintázatokat, amik egy adott támadási vektorra jellemzőek. Ezáltal a támadások pontosabban felfedezhetőek és emellett vektoronként azonosíthatóak.



1.1.1. ábra. A detektor modellje

Az 1.1. ábrán látható a két fokozat leegyszerűsített modellje és a bennük történő nagyobb megvalósított lépések. Látható, hogy a különböző fokozatokban a két említett gépi tanulási metódus problémakör szerint elkülönül és az adatok feldolgozása egy nulladik lépésben történik, ami a bemenetek előállítása céljából szükséges. A bemenetek létrehozása és a modellek elkészítése is Python környezetben történt a TensorFlow és a Pyshark könyvtárak felhasználásával. A modell hiperparamtétereinek optimalizálását és validálását laborkörülmények között nagy méretű adathalmazon végeztem.

2. Irodalomkutatás

Ahogy azt már fentebb említettem, a legkézenfekvőbb megoldás egy DDoS támadás detekciójára egy hagyományos algoritmuson alapuló eljárás lenne. Ezzel a megoldással viszont abba a problémába ütközünk, hogy a gyorsan változó és fejlődő támadások hamar radar alá tudnak kerülni. Minden alkalommal amikor változtatnak a támadók a módszerükön, a kódunkat meg kell változtatnunk és újra validálnunk aszerint, hogy éppen milyen vektorokat kevertek az aktuális támadáshoz. Emellett a hétköznapi forgalmi mintázatok jellegzetességei is változnak idővel és így előfordulhat, hogy tévesen ítélünk meg egy átlagos felhasználót, amit szeretnénk elkerülni.

Erre a problémára orvoslást nyújthat a gépi tanulás módszerei. Az így alkotott detektorok könnyebben tudnak alkalmazkodni a változásokhoz, hiszen ilyenkor csak arra van szükség, hogy az alkalmazást újra tanítsuk az új forgalmi mintázatokkal, ezáltal növelhető a pontosság és a „karbantartási" igény csökkenthető. Az elmúlt években több megoldás is született, amik különböző féle gépi tanulási módszereket használtak fel DDoS detekcióra. Az említett pozitívumok okán választottam én is ezt az eszközt a munkám során, ahol két különböző metódust alkalmaztam a problémák tulajdonsága szerint. Jiangtao Pei és csapata a [1] cikkben beszámol egy mestersége intelligenciát felhasználó megoldásról. Ennél a megoldásnál 3 gyakori támadási minta detekciójára ad megoldást, amikre magas pontosságot ér el. A detektálási metódus random forest modellen alapszik. A hiányossága a megvalósításnak, hogy csak megadott vektorokra működik a rendszer és ezáltal egy összetett több vektort használó támadás így észrevétlenül keresztüljuthat rajta. Erre a problémára megoldást nyújthat az anomália detekció, hiszen ilyenkor a hasznos forgalomtól eltérést ismerjük fel, bár a pontosságunk kisebb lesz valamivel, ez amiatt van mert nem egy adott mintázatot keresünk.

A különböző osztályozási módszereket felhasználó megoldásokról a [2] cikkben beszámol Khundrakpam Johnson Singh és csapata. A megoldások forgalmi viselkedés alapján próbálnak osztályozást végezni. A munkájuk nem egy általános hálózati modellre készült, de ettől a támadások lényege független. A különböző módszereket összehasonlítja és látható, hogy magas pontosságú detekciót ér el velük. Ezzel az előzőleg megemlített hiányosságra ad megoldást. Ennél a felhasználásnál viszont a támadások típusát nem tudjuk meg. Ezeken az alapokon elindulva terveztem a saját megoldásomat,

hogy mindkettő problémára megoldást adjon. Az általam megvalósított megoldásban a kettő ötvözésével beazonosítjuk, hogy milyen típusú DDoS vektorokból áll a támadás. Ez abban az esetben is igaz, ha új vektorokat vetnek be a rosszakarók, mivel a normális forgalomtól való eltérés felismerhető marad.

Saikat Das és csapata a [3] cikkben összehasonlít több különböző osztályozási módszert. Ennek a megoldásnak hatékonysága, hogy a támadások tényét detektálhatjuk, de sajnos a különböző fajta támadások nem megkülönböztethetők. A különböző osztályozások összegzése magas pontosságot ér el, viszont nagy a tanítási igénye. Ramin Fadaei Fouladi a [4] munka során statisztikai megoldásúakat használ fel a DDoS detekcióra. A módszer az eloszlás alakjának az aszimmetriáját méri, ezzel meglepően magas pontosságot elérve. Sajnos ennek a módszernek is hiányossága, hogy magát a támadást nem ismeri fel csak a tényét. Xiaoyu Liang [5] beszámol számos osztályozás összehasonlításáról, de ezeket csak TCP és ICMP alapú támadásokra végzi. A két támadás gyakori, bár sok vektoros támadások vagy más protokollokat használók könnyedén átjuthatnak. Ezáltal a védelem hiányosnak bizonyulhat.

Természetesen a mesterséges intelligencián alapuló megoldásoknak is vannak problémái. Megfelelő tanítási alapanyag kell és ennek minősége nagyban befolyásolja az eredményt. Ezentúl az újra tanítás időigényes tevékenység és ezt egy 24 órában működő szolgáltatásnál is el kell végeznünk, aminél természetesen nem engedhető meg a kikapcsolás. Ezzel a problémakörrel nem foglalkoztam ebben a dolgozatban, de a munka folytatásában lehetséges, hogy a jövőben erre a nehézségre is próbálok megoldást találni a detektor fejlesztéséhez.

3. Elméleti és technológiai háttér

3.1 A DDoS támadások technikája

Napjainkban az internetes szolgáltatások egyre több területet hódítanak meg életünkben. Nem csak munkára használják az emberek, de szórakozásra, tanulásra, egyéb szolgáltatásokra vagy akár egészségügyekre is alkalmazzák. A közelmúlt eseményei miatt az alapból létező digitalizációs folyamatok felgyorsultak. Több távmunkás jelent meg, az oktatás és sok egyéb terület is nagyot lépett az internet világa felé. Sajnos ennek a fejlődésnek van egy árnyoldala is. Mivel nőtt a szerepe és a felhasználtsága ezeknek a szolgáltatásoknak, ennél fogva célponttá váltak rosszindulatú tevékenységeknek. Ezen támadások egyik fajtája a Distributed Denial of Service (DDoS), ami nagy teret nyert magának az elmúlt időszakban.

A DDoS támadások lényege, hogy a támadó megpróbál egy számítógépet vagy szolgáltatást túlterhelni, hogy azt az átlagos felhasználók ne tudják használni. A támadó a generált nagy forgalommal a célpont eszközeit hosszú időre működésképtelenné tudja tenni, mert egyszerűen azok nem tudják ezt kezelni. Ezt úgy hajtja végre, hogy több internetképes eszközt (számítógépek, telefonok, IoT eszközök stb.) malwarek segítségével megfertőz, majd ezekkel egy nagyobb volumenű támadást tud bevetni. A megfertőzött gépek csoportját botnetnek hívjuk. A botnet segítségével akár Tbyte méretű támadások is elérhetőek.

3.1.1 DDoS támadási trendek

A támadások, mint gyakoriságban, mint bonyolultságban is folyamatosan fejlődnek. A 2019-ben induló Covid járvány nagyot lendített ezen rosszindulatú tevékenységek növekedésén, mivel sok szolgáltatás költözött át az internetre és a meglévőkre is nagyobb igény jelentkezett.

Míg 2019-ben a havi támadások átlaga kb. 710.000 volt, addig 2020-ra ez 840.000-re nőtt. A gyakoriság mellett nagyobb volumenű multivektor támadások jelentek meg, ahol már akár a több mint 20 vektort is elérték. A növekvő tendenciákról a [6] ír bővebben. Ezek megnehezítik a védelmet hiszen több különböző támadástípust könnyebb elrejteni az átlagos felhasználói forgalom között vagy esetleg egy nagy sávszélességet foglalóval elrejthető pár kisebb. Ezeket a különböző típusokat a támadók bárhogy

vegyíthetik kedvük szerint. Az 3.1. ábrán jól látható, hogy a több vektor használatával jóval nagyobb támadás érhető el, mint az eddigiekben. A multivektorok elterjedéséről bővebb leírás található a [7], [6] cikkekben.

3.1. ábra. Multivektor támadás. [8]

Ezentúl a támadások gyorsasága is megnövekedett, amivel a detektálásra kis reakcióidőt hagynak és az eszközöket is jobban megterhelik. A „hit and run" típusú támadások gyakorisága megnőtt, ahol csak tizedmásodperces nagyságú időszeletekben hatalmas mennyiségű adatot zúdítanak az áldozatra és ezt időközönként ismétlik, így növelve a védekezés nehézségét.

A támadások, ahogy azt az előbb említettem számos vektorból is állhatnak, ezek a támadásitípusok az OSI modell több rétegében is megtalálhatóak a különböző protokollokat kihasználva, mint például a HTTP, DNS az alkalmazási rétegből és a TCP, UDP a szállítási rétegből. A támadásoknál a célpont internetes eszközeit terhelik túl oly módon, hogy azoknak a hardveres és szoftveres képességeit kimerítik. Gyakori támadási típusokról a [9] számol be.

3.1.2 A botnetek terjedése a világhálón

A botnetek növekedését az elmúlt időszakban két tényező segítette elő nagyobb mértékben. Az egyik az IoT rendületlen eszközmennyiség növekedése, ami mára már elérte a több 10 milliárdos mennyiséget, a másik pedig a pandémia alatti sok eszköz vásárlása.

Az IoT eszközök főbb problémája, hogy általában gyenge védelemmel vannak ellátva és mivel valahova kihelyezik őket, ennél fogva a későbbiekben nem igazán lehetséges ezen eszközök szoftveres frissítése, javítása vagy legalábbis nehézkes.

A gyártás során esetleges hibákat könnyen kihasználhatják a támadók. Emellett nehéz rájönni, hogy megfertőzték az eszközünket, hiszen ezeknek a káros programoknak általában nem mi vagyunk a célpontjai, emiatt nem okoznak semmilyen problémát számunkra. Az igazat megvallva a felhasználók többsége pedig nem szokta ellenőrizni, hogy az okos mosógépe egy botnet eszközzé vált-e vagy sem. Ha megnézzük az eszközmennyiséget, a gyenge védelmet és a felügyelet hiányát könnyen rájöhetünk, hogy az IoT könnyűszerrel növelheti a DDoS támadások erősségét.

Az egyik legelterjedtebb malware a Mirai. Ennek célpontjai kifejezetten IoT eszközök, amiken Linux működik és az előbb említett gyenge védelmet kihasználva fertőzik meg azokat. A malware a gyakori gyári alapbeállításokban megadott felhasználónevekre és jelszavakra próbálkozik és magában az eszközben semmi kárt nem tesz. Az egyik nagyobb támadás, amit 2016 októberében hajtottak végre a Dyn internetes szolgáltatást nyújtó cégen. A támadás rengeteg internetképes eszközt felhasználva történt, mint például IP kamerák, nyomtatók stb. A támadásról és a Mirai-ról a [10] és [11] számol be bővebben.

Természetesen nem minden IoT eszköznél ekkora a probléma. Vannak területek, ahol a védelemre több energiát fordítanak, de sajnos a mennyiségük miatt komoly erőt adnak a támadók kezébe.

3.1.3 Támadó és áldozat

A DDoS támadásoknak általában nagyobb vállalatok az áldozatai, de persze akár egy magánszemély is lehet. Különböző internetes szolgáltatást nyújtó cégek a gyakrabban kiszemelt célpontok. Lehetnek bankok, szórakoztatást nyújtó (filmes tartalmak vagy internetes játékok), online vásárlást kínáló oldalak, de valamilyen közcélt ellátó netes felület is pl. egészségügy, oktatás vagy esetleg az internetszolgáltató. A támadásnak több oka is lehet. Esetenként csak viccből, erőfitoktatásból történnek ezek esetleg egy hacker csoporttól, de az ok származhat anyagi, politikai alapokból vagy akár bosszúból.

A DDoS erőssége az egyszerűségében rejlik, mivel a támadásokat roppant könnyen végrehajthatjuk és ezáltal nagy károkat okozhatunk. A probléma ezzel szemben az, hogy a védekezés nehéz és költséges ellene. Egy rosszakaró a dark weben pár dollárért vehet egy támadást bárki ellen és ha valaki kicsit is ért ezekhez, akkor akár maga is végrehajthatja a támadást. A rosszakaró identitását pedig szinte lehetetlen kideríteni, hiszen gyakran az eszközök, ahonnan a támadás érkezett nem az ő tulajdonában vannak,

hanem más emberekében, akiknek ehhez semmi köze, csak az ő eszközeik lettek megfertőzve. Persze az a tény, hogy egyszerűek a támadások nem jelenti azt, hogy ne lehetne bonyolítani őket, ahogy az előző fejezetben is említettem már a vektorok számának növelésével és új típusú vektorok készítésével. A védelem bővülésével egyes típusok eltűnnek, ritkábbak lesznek és ezáltal új támadási lehetőségekre van szükség. A zero-day támadások elleni védelem emiatt mindig nehézkes.

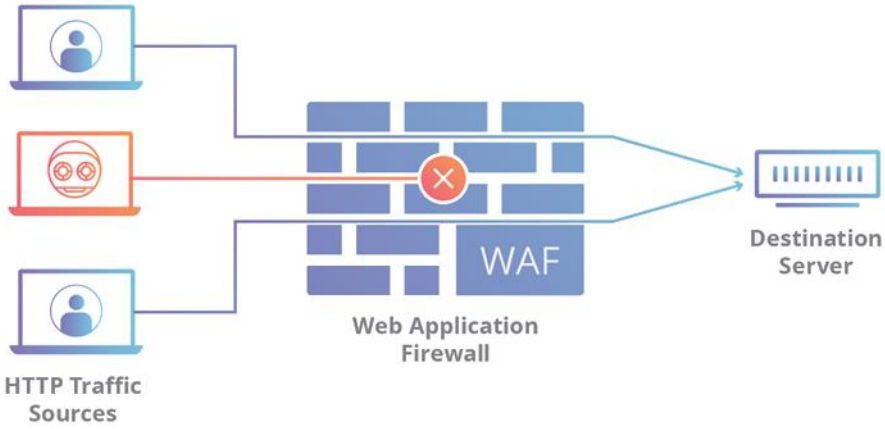
3.1.4 A támadások jellemzői és védekezés

A DDoS támadásoknak van pár jellegzetessége. A nehézséget az okozza a felismerésében, hogy a támadókat a normális, átlagos felhasználóktól kell megkülönböztetnünk. Sajnos az, hogy nagy a forgalom nem garancia mert lehet, hogy egy új terméket szeretnének aznap eladni és emiatt a felhasználók száma nagyban megnövekedett és nem lenne szerencsés, ha az újonnan érkező potenciális vásárlóinkat tiltanánk le.

A védekezés emiatt a detekciónál kezdődik, hogy megkülönböztessük a hasznos és káros forgalmat. A legfontosabb jellegzetessége a támadásoknak a hirtelen megnövekedett adatforgalom, de ahogy az előbb említettem ez nem garancia. Ezenkívül még az is fontos, hogy ez a forgalom hány IP címről érkezik, a támadásban résztvevő eszközök gyakran ugyanabból az IP cím tartományból származnak. Természetesen az is egy ismérv, ha gyanúsan ugyanolyan termékekről érkezik (vagy egy adott helységből, böngésző verzióból), mivel ilyenkor lehet, hogy ott felfedeztek valami hibát a védelemben és ezt kihasználva teremtették meg a botnetjüket a támadók. Az idő is kulcsényező, mint a hirtelen „tüskés alakú” nagy forgalom minták és mint a napszak szempontjából is, hogyha eltér az átlagostól (pl. egy online webáruházat hajnali háromkor nem feltétlen fogja sok ember felkeresni). A különböző támadás típusoknak lehetnek sajátos jellemzői is. Erre például: a TCP ACK támadás típusnál az ACK aránya egy adott TCP folyamban adhat árulkodó jeleket vagy a DNS Amplified-nál, ha olyan DNS válaszokat kapunk, amikre sose küldtünk DNS kéréseket.

Többféle módszerrel is védekezhetünk a támadás ellen. A védelem hatékonyságát befolyásolja, hogy éppen milyen vektorokat kell elhárítani, hiszen ezeknek esetleg más jellemzői vannak vagy teljesen másik OSI rétegben találhatóak. A két legelterjedtebb detekciós eszköz az adatforgalom-alapú elemző rendszer (pl. Intrustion Detection System, IDS) és a hálózati tűzfal. Miután rájöttünk, hogy támadás alatt állunk több

lehetőségünk is van. A két gyakori opció a forgalom limitálása (ezzel növelve a szerverünk biztonságát) vagy lehetséges adott felhasználók letiltása is. Abban az esetben, ha mindenféleképpen megszeretnénk védeni az eszközeink működőképességét, akkor használhatjuk a „blackhole routingot". Ebben az esetben a forgalmat eldobjuk függetlenül mindentől, ilyenkor sajnos a hasznos forgalom is erre a sorsra jut, tehát a támadás összességében eredményes lesz, mert maga a szerver a kívánt célt nem képes ellátni és az emberek azt nem tudják elérni. A „blackhole routing" módszerről a [12] szolgál több információt. Az előbb említett forgalom limitálásnak is megvannak a gyengeségei. Egy „brute force" támadás ellen hasznosnak bizonyulhat, de egy összetett multivektor támadás esetén önmagában nem feltétlen lesz elég.



3.2. ábra. A WAF működése. [13]

Layer 7 védelmet a Web Application Firewall (WAF) segíti, ami a HTTP forgalmat szűri és felügyeli. Ezzel a védelmünket kibővíthetjük és más vektorokat fedhetünk le vele. A 3.2 ábrán látható, hogy a WAF még a szerver előtt található és a forgalomból előre megadott irányelvek alapján próbálja szűrni a károsnak vélt tevékenységeket. A WAF technológia használatáról DDoS támadások ellen bővebben ír a [14] cikk. Más védekezési és detektálási típusokról számol be az [15] és [12] cikk.

3.2 A felhasznált mesterséges intelligencia modellek

A mesterséges intelligencia alapötlete már régóta létezik a számítástechnikában és napjainkra egyre nagyobb területeket hódít meg. A módszer az élőlények gondolkodásán alapszik. Felhasználása számos új lehetőséget teremtett a programozók számára.

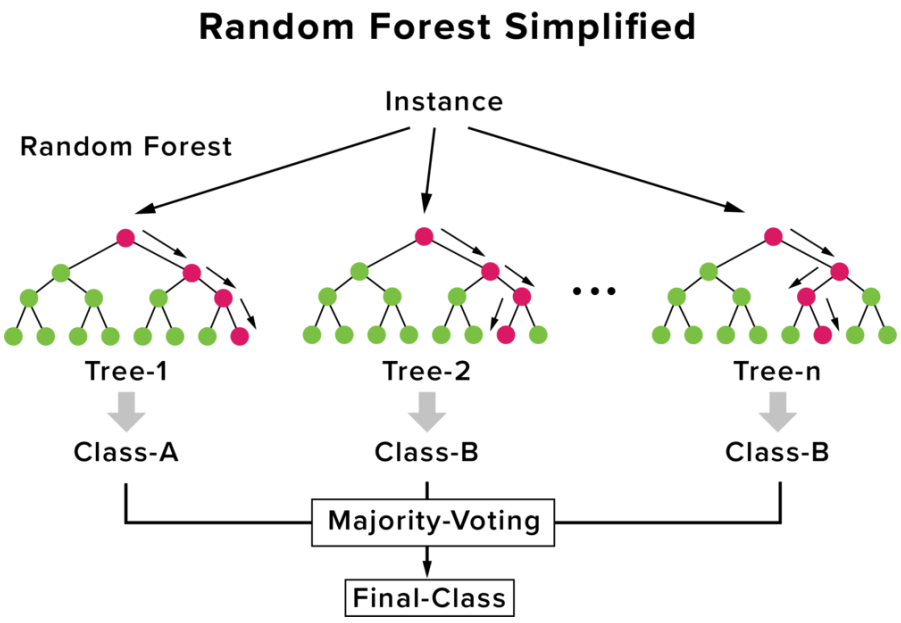
A korábbiakban már említett pozitívumok miatt én is ezt használtam fel a munkám során. A detektorban a mesterséges intelligencián belüli gépi tanulás területéről származó metódusokat alkalmaztam. Ezek a korábban már megemlített random forest és szekvenciális modellek. A gépi tanulás alapja, hogy egy adathalmazból próbál mintázatokat felismerni, majd ezek alapján osztályokat megkülönböztetni.

Alapból kétféle tanítási módszer létezik, ezek a felügyelt és felügyeletmentes (supervised, unsupervised). A felügyeletes tanításnál előre meghatározott osztályok közötti különbségekre próbáljuk rátanítani a programunkat, míg a felügyeletmentesnél a mélyben rejlő mintázatok felismerését adjuk ki feladatnak és ez alapján keletkeznek különböző osztályok.

3.2.1 A random forest modell működése

A random forest módszere a döntési fán alapszik, ahol ebből többet felhasználva jön létre a kiválasztás. A döntési fának a lényege roppant egyszerű. A fával képesek vagyunk osztályozást végrehajtani, ami egy felügyelt tanulás eredetileg, de ezt meglehet oldani, hogy felügyeletmentesen is működjön. Erre azért van szükség, mert az első fázisban anomáliákat keresünk és erre a felügyelt tanítási módszerek nem alkalmasak, viszont így a random forest modell jó tulajdonságait meglehet tartani. Ezt a módszert az 5. fejezetben majd ismertetem.

Az osztályozás úgy működik, hogy a fa gyökeréből kiindulva tulajdonságok szerint részhalmazokra bontjuk a halmazunkat és ezt rekurzívan addig ismételjük míg a halmaz tulajdonságok meg nem egyeznek az osztályokkal. Ezeket a fákat felhasználva kapjuk meg az erdőt. Az erdőben a különböző fák véletlenszerűen kapnak adathalmazokat ezáltal leküzdve a döntési fa adat érzékenységét, hiszen az erdőben a fák eredményének átlagát fogjuk felhasználni és ezzel a random forest erősen zajtűrő lesz. A megoldás eredménye hasonló a k-fold cross validation-hez, ezáltal magas pontosságot tudunk elérni akár kevés adattal is. A 3.3. ábrán jól látható ahogyan a különböző fák a szétosztott adathalmazok miatt nem feltétlen ugyanarra az eredményre jutnak, de a többségi szavazás módján a félreeső értékek nem zavarják meg a végeredményt.



3.3. ábra. Random Forest működése. [16]

Számos jó tulajdonsággal rendelkezik a random forest algoritmus. Ezek között van a nagy mennyiségű bemenet kezelése, kevés adattal is képes működni, magas pontossággal és a zajra érzéketlenebb, mint a társai. Ezáltal egy robosztus megoldást tud nyújtani a sebesség csökkenésének árán. A random forest modellről és annak DDoS védelemben használatáról bővebben beszámol a [17], [18] cikk.

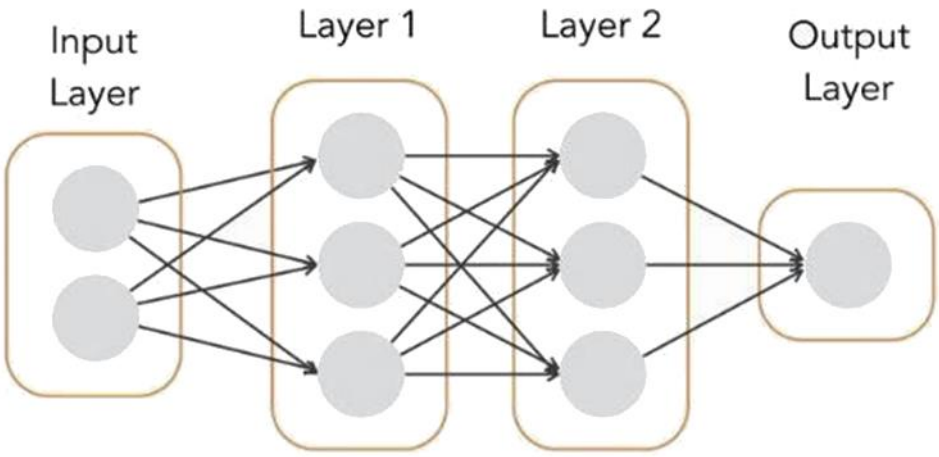
3.2.2 A 2. fokozatban használt szekvenciális modell ismertetése

Ez a gépi tanuláson belül található mélytanulási metódusok közé tartozik, amely egy a Keras-ban (Python könyvtár) található tanuló algoritmus. A legalapvetőbb formátumot felhasználva, ami az emberi agyban lévő neuron mintázatára létrehozott perceptronok hálózatát jelenti.

Egy perceptron és a belőle létrehozott többrétegű modell jól használható bináris osztályozásra, ennél fogva tökéletes a feladatra, hogy eldöntse, hogy egy támadásban megtalálható az adott vektor vagy sem. Így ellátva a detektor 2. fokozatának feladatát és meghatározható a támadásokban felhasznált különböző vektorok száma és milyensége.

A többrétegű neurális hálózat az ANN (Artificial neural network) családjába tartozik és ez a metódus is egy felügyelt tanítási módszer. A felépítése lineárisan elhelyezett perceptron rétegekből áll. Három réteg típust különböztetünk meg: a bemeneti réteg és kimeneti réteg között találhatóak a rejtett rétegek. A rejtett rétegek számát és a neuronok számát tetszőlegesen határozhatjuk meg, ahol a neuronok matematikai

műveletekkel valósítanak meg egy komplexebb matematikai problémát. A tanítás során ezeknek a csomópontoknak a súlyozását állítjuk be. A kimeneti rétegben csak egy neuron található hiszen az esetünkben csak egy „0-1” választ várunk, vagyis a támadás a megadott vektor-e vagy sem. Ez egy teljesen összekötött modell, ami annyit jelent, hogy a rétegekben található neuronok összes kimenete bele van kötve a következő réteg minden bemenetébe. Az 3.4. ábrán ez jól látható a modell felépítése mellett. A [19] és [20] cikk foglalkozik bővebben az ANN típusú modellekkel.



3.4. ábra. Többrétegű modell felépítése. [21]

3.3 A detektor fejlesztési környezete

Az egész detektor Python-os környezetben lett implementálva. A python3 alap könyvtárain kívül még két könyvtár volt alkalmazva, amikben lévő függvények az alapját adják a megvalósított részeknek. Három nagyobb részből áll a program.

Az első az internetes forgalom feldolgozása és a bemenetek előállítása. Ez a rész a Pyshark könyvtár felhasználásával készült, ami a Wireshark parancs sori változatára, a Tsharkra alapszik. Ezzel a hálózati forgalmakat rögzítő PCAP fájlokat tudjuk olvasni és a benne lévő adatok alapján a bemeneteket generálni, amik egy CSV-be kerülnek. Ezek a modellek tanítására és tesztelésére használhatók.

A 2. és 3. részben felhasználásra került a TensorFlow és a Keras, ami egy alkalmazásprogramozási felületet nyújt a TensorFlow-hoz. Ezek a könyvtárak a felhasználásra kerülő modellek megalkotásában segítettek, amikben megtalálható mindkettő a korábbiakban bemutatott modellek létrehozásához szükséges alapok.

4. Kutatási módszertan bemutatása

4.1 Időablak-alapú, valós idejű forgalomfeldolgozás

A feladat célkitűzésében szerepelt a valósidejűség megvalósítása is. Ennek érdekében és a támadások egy jellegzetességéből fakadóan az időt egységekben kezeltem. A támadások egyik jellegzetessége a burst-ös jelleg, amit ezáltal könnyedén észrevehetünk az időszeletekben érkezett csomag- és adatmennyiség alapján. Az implementációban ez az alapegység 10 milliszekundumra volt állítva, de ezt egy változó átírásával könnyedén meglehet változtatni annak megfelelően, hogy a detektort milyen környezetben használjuk. Így a detekció egyszerűen a felhasználási helyhez adaptálható. Mivel a felhasznált időablakok igen kicsik, ezért a megvalósítás szempontjából tekinthető valós idejű feldolgozásnak annak ellenére, hogy pár csomag összevárásra kerül.

Az időegység alapján van minden bemenet előállítva, ezáltal az időablakok tulajdonságait vizsgáljuk és ez alapján hozzuk meg a döntéseket. Az internetes forgalmat időszeletekben tárolva könnyebben tudjuk kezelni és mozgatni a lépések között és a támadás nagyságát, összetettségét is jobban reprezentálja.

Ez lehetőséget ad arra, hogy összehasonlításokat végezzünk köztük, amivel a különbségek jobban kijönnek. Az átlagtól való eltérés vizsgálata így egyszerűvé válik. Annak érdekében, hogy ez reprezentatívabb legyen ezt csak a vizsgált egység előtti megadott egységek összességére végezzük, így a változás jobban érzékelhető. Ennek megvalósításáról bővebben beszámolok a bemenetek előállításáról szóló részben.

4.2 Modellek kiválasztása problémakör alapján

Ahogy azt már korábban is említettem a detektor két fázisa különböző elven működik, mivel más a megadott probléma.

A második fázisban egy osztályba tartozást vizsgálunk, ezért a korábban bemutatott többrétegű neurális hálózat kézenfekvő megoldásnak bizonyosodott az egyszerűsége és pontossága miatt.

Az első fázis anomália detektora ennél több problémát okozott, hiszen itt nem csak egy egyszerű igen nem kérdést teszünk fel. A kapott válasz, emberi szempontból természetesen az, hogy támadásról van-e szó vagy sem, viszont a gépi tanulási

módszerünk ezen továbblép és a támadásivektorokat képes osztályokba sorolni. Természetesen nem ugyanazok a tulajdonságai egy banki tranzakciónak, mint egy videó streaming forgalomnak, mégis mind a kettő hétköznapi szolgáltatásnak számít az interneten. Ez természetesen a támadásokra is igaz, hiszen ott is a különböző támadások más-más jellegzetességekkel rendelkeznek. Ezen okból ezek külön osztályokat alkotnak, hiába mi az emberi oldalról ezt a kérdést egy eldöntendő kérdésként kezeljük.

Erre a problémára különböző klaszterező megoldások léteznek, ezek összehasonlításáról és használatáról DDoS detektorokban a [22], [23] és [24] cikkek számolnak be. Az én választásom a random forest modellre esett számos jó tulajdonsága miatt. Ez alapból egy felügyelt tanítási módon működő megoldás, ami ezáltal nem használható anomália detekcióhoz. Ez a probléma viszont megoldható átalakítások segítésével és a random forest algoritmus egy klaszterezés kiinduló állapotává tehető, megtartva ezzel a hasznos tulajdonságait. A módszer megvalósításáról a 5. fejezetben bővebben beszámolok.

Az ötletet először ismert klaszterezési problémákon teszteltem és finomítottam. Ezek a tesztelések először az ismert Iris és később az Adult adathalmazokon történtek. A biztató eredmények után alkalmaztam a megoldást a DDoS detektornál. Természetesen a probléma nem teljesen egyezik meg, de ígéretesnek bizonyult.

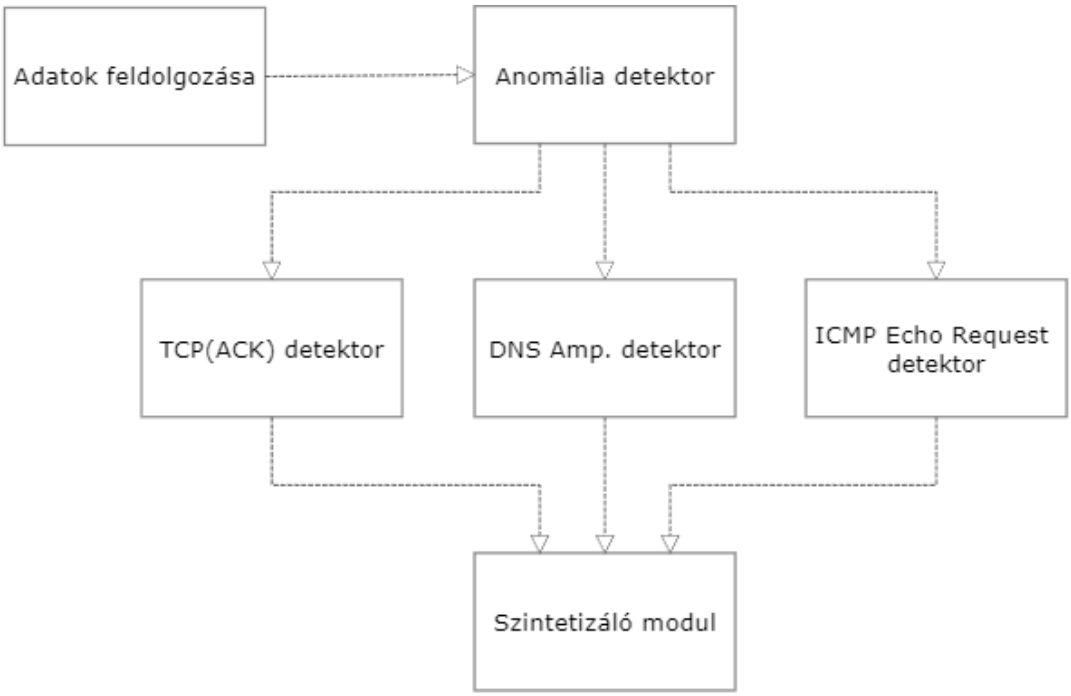
4.3 Modellek tesztelésének és finomításának módszere

A detektor elemeinek elkészítése után a megoldások finomítása és tesztelése következett. Az adathalmazok, amiket használtam a munkám során valós internetes forgalomból származnak. Ezeket használtam fel egyaránt a tanításhoz és a teszteléshez is. Az első fokozatban lévő anomália detektor és a második fokozatban lévő TCP(ACK) elárasztás és DNS Amlified detektorok mindegyikénél a valós forgalomból származó adathalmazzal dolgoztam. A második fázisban található harmadik detektornál viszont általam generált adathalmazt használtam fel, mivel ez egy ICMP Echo Request elárasztás támadás felismerésén dolgozó detektor és az egyszerűsége miatt, könnyebb volt generált adathalmazzal tanítani és tesztelni.

A sorozatos tesztelések eredményét felhasználva optimalizáltam a modellek hiperparamétereit. A finomítások leginkább a tesztelések mentén empirikus módon történtek.

5. A detekciós modell és a tervezett architektúra

A detektor architektúrája megfigyelhető a 5.1. ábrán. Jól látható a két elkülönülő fokozat, azok elemei és az internetes forgalomból származó adatok feldolgozásáért felelős előszakasz. Az eredmények kiértékelését pedig egy szintetizáló modul végzi, ahol a különböző detektor eredmények összegzésre kerülnek. A modellek felépítését, implementációját és működését ebben a fejezetben részletesen ismertetem.



5.1. ábra. A DDoS detektor architektúrája.

5.1 Az adatok feldolgozása és a bemenetek előállítása

A bemenetek előállítása az előszakaszban történik, ahol az internetes hálózaton rögzített PCAP fájlokból a detektor mesterséges intelligencia modelljeinek bemenetére kerülő CSV fájlok készülnek. A bemenetek tisztasága és jól összeválogatása nagyban befolyásolja a végeredmény sikerességét, ezért ez egy fontos része az egész modellnek.

Az anomália detektor bemenetére összesen 9 különböző paraméter kerül. A CSV-ben 10 adat található, mert a 9 bemeneten kívül az időegységhez tartozó címke is megtalálható, hogy támadás-e vagy sem. Erre a tanítás és a tesztelés során van szükségünk. A bemenetek előállításának módszere jól látható a 5.2. ábrán. Az összes bemenet ugyanerre a sémára épült, ahol az időablakban lévő csomagokhoz tartozó adatok

alapján kiszámításra kerül az éppen megadott bemenet. A hasonlóságuknak köszönhetően jól párhuzamosítható a folyamat, ezáltal az adott egységhez tartozó bemenetek gyorsan számíthatóak. A sémában a különböző időegységhez tartozó elemek egymás után vannak fűzve majd a különböző bemenetek is összefűzésre kerülnek, ezzel megalkotva a CSV fájlt, ami a modellbe kerül majd. A bemenetek előállításánál a forgalom irányát aszerint vesszük, hogy mi vagyunk a fogadó fél, hiszen magunkat védjük a támadástól. Ezek alapján a PCAP fájlokban a MAC cím alapján döntjük ezt el. A Pythonban a saját MAC címünket egy egyszerű függvény felhasználásával megkaphatjuk, de megadtam a lehetőséget arra is, ha mi szeretnénk MAC címet vagy MAC címeket megadni, ezt ilyenkor csak egy változóba kell beírni.

```
154  #4 Erkezesi ido
155  def arrival_time(cap, n_packet_intime, where, data):
156      value = 0
157      time = 0
158      text = 'Arrival_time'
159      if n_packet_intime == 0:
160          data = concater(text, value, data)
161      else:
162          j = where
163          j = int(j)
164          time = cap[j].sniff_time
165          value = time.hour
166          data = concater(text, value, data)
167      return data
168
169  def arrival_time_maker(cap, n_packet_intime):
170      data = pd.DataFrame()
171      where = 0
172      for i in range(len(n_packet_intime)):
173          data = arrival_time(cap, int(n_packet_intime[i]), where, data)
174          where = where + n_packet_intime[i]
175      return data
```

5.2. ábra. A bemenetek előállításaának sémája.

A forgalom feldolgozása előtt meghatároztam, hogy az adott PCAP-ben támadás vagy normális forgalom található és eszerint külön dolgoztam fel őket, hogy a felcímkézésük egyszerű legyen. A normális forgalom megkapta a „0” címkét és a támadásos forgalmak pedig az „1” címkét. A tanításnál fontos, hogy az implementálási hely forgalmából legyen tanítva, mivel a forgalom tulajdonságai nagyban eltérhetnek attól függően, hogy hol lett a detektor telepítve. Azokat az időegységeket, ahol egy csomag se érkezett vagy egy se lett elküldve nem tartottam számon, mivel egy csomag

mentes egység nem lehet támadás semmiféleképpen, ezáltal a tanításban semmi szükség nincs rá és alapból is csak egy kilógó értelmetlen adathalmaz lenne. Az 5.1. táblázatban a bemenetek kerültek felsorolásra és egy példaérték, hogy reprezentatívabb legyen a továbbiakban történő magyarázatuk. A táblázatban bemutatott adatok egy támadás időszeletéhez tartoznak. A bemeneteket a 5.1.1 – 5.1.9 fejezetekben ismertetem bővebben.

Packet_time_ratio	Tx_Rx_rate	Avg_packet_size	Arrival_time	Sum_src_IP	Sum_prot_num	Delta_burst	TCP/UDP_ratio	Paylod_entropy
2734	1,0	131,7048	12	936	6	83481	0,00109	7,9647

5.1. táblázat. Az anomália detektor bemenetei.

5.1.1 Csomagok érkezési és küldési sebessége

Ezzel a támadások azon tulajdonságát tudjuk vizsgálni, hogy a csomagok mennyisége nagyban megnő. Természetesen ez önmagában nem feltétlen elég, hiszen az átlagos felhasználói forgalom is lehet hirtelen magas, például egy új termék piacra dobásakor, de egy jó kiindulási alapot adhat számunkra.

Az első bemenet egyben az irányadó is a többi számítása során. Itt azt számoljuk meg, hogy egy adott időegységben hány darab csomagot fogadtunk és küldtünk el összesen. Ezeket a PCAP fájlban lévő abszolút idejük alapján végezzük. Az első csomag érkezési idejét vesszük az idő kezdetének és az innentől az időablakok alatt beérkező csomagokat egységenként összeszámoljuk. Valójában ez nem a darabmennyiség dimenziójú lesz, hanem egy sebességet eredményez a számunkra, hiszen az eredmények csomag/időegység mértékegységgel rendelkeznek. A továbbiakban ezt a csomagszámot feltudjuk arra használni, hogy tudjuk mennyi csomagot kell feldolgoznunk a különböző bemeneteteknél időegységenként.

5.1.2 Küldött és fogadott csomagok aránya

Az internetes forgalom két fél kommunikációját jelenti lényegében. Abban az esetben, ha ez nagyon eltolódik a támadás gyanúját keltheti bennünk főleg olyankor, amikor ez megnövekedett csomagmennyiséghez társul. Ez nem azt jelenti, hogy 0,5 aránynak kell lennie a kommunikáló felek között, mivel ez a forgalom típusától is függ. Egy 4K felbontású videós tartalom esetén az arány a csomagok között lehet 0,98.

Ennek a paraméternek a meghatározása meglehetősen egyszerű. Ahogy már korábban említettem a MAC címünket könnyedén megszerezhetjük és ezáltal a csomagokban különbséget tudunk tenni, küldő és fogadó között. Megszámoljuk a küldött csomagok számát az adott időegységben és ezt elosztjuk az összes csomaggal, amit már ismertünk az első bemenetnek köszönhetően, hiszen ott ez már megszámolásra került.

5.1.3 A csomagok átlagos mérete

Ilyenkor mindkét véglet kirívó lehet számunkra. A csomagok túlzott nagysága arra utalhat, hogy esetleg szeméttel megpakolt csomagokat kapunk és ezzel próbálják túlterhelni az eszközünket. ICMP üzenetek elárasztásánál pedig a csomagméretünk az átlagosnál kisebb lesz.

Az előzőhez hasonló módon ez sem túl bonyolult. Itt is azzal dolgozunk, hogy az időegységben lévő csomagok számát már ismerjük. Emiatt elég összegeznünk az időegységben lévő összes csomag méretét, majd pedig ezt elosztani a csomagok számával, ily módon könnyen megkapható a csomagok méretének az átlaga.

5.1.4 A csomagok érkezésének ideje

Az emberi szokások elég megszokott mintázatokat mutatnak. A felhasználói szokásokból adódó időtartományok jó nyom lehet sok esetben. A hajnali órák kifejezetten gyanúsak, mivel nem túl valószínű, hogy egy ruha bolt webáruházát hajnali háromkor a vásárlók tömege ostromolja. Mindenesetre ez sem garancia, mert esetleg még az előző is megtörténhet, de a támadó is szervezheti akcióját az emberek által gyakran használt idősávokba.

Ezt az adatot csak a nap óráira bontva mértem, mert ennél pontosabban felesleges számontartani, mivel nem szolgáltat számunkra több hasznos adatot a pontosabb idő. A PCAP fájlban időbélyeggel kerülnek rögzítésre a csomagok, ezt felhasználva meg is tudjuk szerezni a csomag érkezésének óráját. Az 5.2. ábrán látható kódrészlet ennek a bemenetnek a meghatározását ábrázolja. Az adott időegységek egyértelműen egy adott órába tartoznak, ezért ez nem okoz nehézséget.

5.1.5 A forrás IP címek száma

A DDoS támadások lényege abból fakad, hogy sok eszközről küldenek internetes forgalmat, ezért ez egy fontos paramétere a támadások észlelésénél. Ahogy az előző bemeneteknél ez a bemenet is nagyban függ attól, hogy hol használjuk a detektort.

Ennél a paraméternél kezelnünk kell a tényt, hogy létezik IPv4 és IPv6 címtartomány, mivel ezt a Pyshark külön kezeli. Ahogy már korábban, itt is a MAC cím alapján határoztam meg, hogy mely csomagok a fogadottak és ezeknek a csomagoknak a forrás IP címét vizsgáltam. Az új IP címeket egy tömbben tároltam és mindig ebben a tömbben vizsgáltam, hogy a csomag forrás IP címe megtalálható-e benne vagy sem, ha nem akkor felvettem ebbe. Az időegység végére érve összeszámoltam a tömb elemeit és így megkaptam az IP-k számát, amiről csomagok érkeztek hozzánk.

5.1.6 A különböző protokollok száma

Az adott egységben megjelenő protokollok száma is egy jó indikátor lehet támadás esetén. A sok különböző protokoll egy összetett magas vektorszámú támadás esetén történhet. Amikor meg csak rengeteg egyforma csomag érkezik az lehet egy „brute force” támadás, ami például egy UDP fragmentation vektort használ.

Ennek a megvalósítása nagyon hasonló az ötödik bemenetéhez. Ott különböző IP címeket tároltunk egy tömbben, itt viszont a különböző legfelső protokollokat tároljuk egy tömbben és ennek számossága megadja a kívánt eredményt. Szerencsére a legfelső protokollt könnyen megtudjuk a Python könyvtár segítségével. Persze ebben az esetben is csak a fogadott csomagokat vizsgáljuk ugyanazon okból, hogy magunkat védjük a rosszindulatú tevékenyégektől.

5.1.7 Az átlag csomagmérettől való eltérés

Ez egy időtől független paraméter lesz. Itt arra vagyunk kíváncsiak, hogy az adott időegységben az átlagos csomagméret milyen eltérést mutat az elmúlt időben érkezett csomagok átlagos csomagméret átlagától. Annak érdekében, hogy ez értelmezhető adatot szolgáltasson és az időegységtől függetlenedjen, csak egy meghatározott mennyiségű időegységhez képest vizsgálom a változást. A megvalósított modellben ez az utolsó 10 időegységet jelentette, ami 100 milliszekundumot ölel fel.

Ennek megvalósítása kicsit bonyolultabb volt, mint az ezelőtt lévő bemeneteké. Dinamikus programozást alkalmaztam abból a célból, hogy ez ne járjon magas számítási

igényekkel. Kiszámoltam a 10 elem átlagát, majd a következő időegység csak az átlagot kapta meg és a tömb első elemét meg az új elemet, amik ennek megfelelően kivonódnak és hozzáadódnak az átlaghoz. A kapott átlagértékből pedig kivontam az adott időegységben található csomagok méretének átlagát. Ez a különbség természetesen lehet pozitív és negatív érték is attól függően, hogy nőtt vagy csökkent az átlag csomagméret az elmúlt 100 milliszekundumhoz képest.

5.1.8 A TCP és UDP aránya

A két protokoll a legelterjedtebben szállítási protokollok közé tartozik, emiatt az arányuk értékes információt szolgáltathat számunkra. A csomagok többsége használja ezt a réteget.

Az arányt a könyvtár segítségével könnyen megtudtam határozni. Annyi volt a dolgom, hogy megvizsgáljam a protokollok megtalálhatóak-e az adott csomagban, aztán ezek számát összeszámoltam majd elosztottam az összes csomaggal.

5.1.9 A Payload tartalma

Az összes bemenet közül talán ennek vizsgálata a legérdekesebb. A támadó a hatékonyság növelése érdekében gyakran teletöltheti szeméttel a küldött csomagokat. Annak eldöntése, hogy a csomag tartalma értelmes vagy sem, önmagában meglehetősen nehéz feladatnak bizonyulhat.

Elsősorban megvizsgáltam, hogy a csomagban egyáltalán található-e bármilyen tartalom vagy sem. Abban az esetben, ha találtam, akkor ezeket a tartalmakat összefűztem. Annak érdekében, hogy véletlenül se legyen az összefűzött tartalom túl nagy ezt csak akkor tettem, ha 100-nál kevesebb csomag érkezett az időegységben. Amikor ennél több érkezett, akkor is 100 elem kerül felhasználásra csak ilyenkor a csomagok számát elosztom százzal és ez lesz a lépésszám, ami szerint a csomagok tartalmát összefűzöm. Az összeszedett tartalmak vizsgálata még mindig nehézkes lenne. A megoldás, amit használtam az a tömörítés. Miután ennek eredménye nagy valószínűséggel vagy nagyon jó lesz, vagy nagyon rossz. Ezek után a tömörítések entrópiáját vizsgáltam meg, ami a tömörítést követően az eredeti adatot meglehetősen jól reprezentálni képes, hogy az valami programmal generált vagy sem abból a célból, hogy csak csomagokat tölthessünk fel vele.

5.2 Az anomália detektor

Az anomália detektor lényege az, hogy a megszokott forgalomtól eltérő tulajdonsággal rendelkező elemeket kiszűrje. Ennél a detektor elemnél jön a legjobban képbe a mesterséges intelligencián alapuló programok hasznos tulajdonsága. Az átlagos felhasználói forgalom rövid idő alatt képes sokat változni, ezért gyakran kell újra tanítani a modellt. Ebbe a szakaszba a hálózat összes forgalma beérkezik és mivel a hasznos forgalmak között is nagy különbségek lehetnek, így még nehezebb őket elkülöníteni a támadásoktól.

Ahogyan azt már korábban említettem, az anomália detektor egy random forest modellt használ, ami egy felügyelt elven működő tanulást valósít meg. További lépésekre van szükség annak érdekében, hogy ezt lehessen használni anomália detekcióra, erről még bővebben beszámol a [25].

5.2.1 Az átalakítás elve és implementációja

Ahhoz, hogy megtarthassuk a kívánt modellt át kell alakítanunk a felhasználását. A random forest modellt arra fogjuk alkalmazni, hogy egy klaszterező módszer bemenetét előállítsuk vele, ezzel megtartva a jó tulajdonságait. Ennek menete négy nagyobb lépésre bontható.

A szintetikus adatok és az új adathalmaz létrehozása

Először arra van szükségünk, hogy a felügyelettel történő tanítástól megszabaduljunk, mivel mi a rejtett mély mintázatokra vagyunk kíváncsiak a vizsgálatok során, nem pedig osztályokhoz tartozó jellegzetességekre, hiszen itt minden forgalomhoz külön osztályok tartoznának, amit nem tudnánk megvalósítani.

Mielőtt bármihez is hozzákezdenék, a biztonság kedvéért, ha lenne hibás adat azokat eldobom, hogy a későbbiekben ezek véletlenül se okozzanak problémát. Ezek után az adathalmazomat kettéválasztom és az első résszel végzem a tanítást. A következő megoldás végrehajtása erre az első adatrészre vonatkozik. A kitűzött célt úgy érjük el, hogy szintetikus adatot generálunk. Ennek a megoldásnak az a haszna, hogy ebben az adathalmazban nem léteznek mintázatok. A megvalósítás során én ezt egyszerűen úgy oldottam meg, hogy a CSV-ben lévő elemekben található bemeneteket véletlenszerűen összekevertem. A keverés nem egy elemen belüli adatok felcserélését jelenti, hanem az adathalmaz különböző időszeletei között keverem össze ugyanazokhoz a bemenetekhez

tartozó adatokat. Így eltüntetve az összes összefüggést az adathalmazomban. Ezek után lecserélem a címkét. A szintetikus adatot ellátom egy „Szintetikus” címkével és az eredeti forgalomból származó feldolgozott adatban pedig az eddigi támadás és normális forgalom címkéket lecserélem egységesen egy „Normál” címkére. Miután ezt elvégeztem van egy adathalmazom, aminek fele az eredeti adat és a másik fele pedig az ebből az eredeti adatból képzett szintetikus adat.

A „Normál” címkés adatokat ismét szétválasztom. Az első rész összefűzésre kerül a szintetikus adathalmazzal a másik fele pedig majd a fa teszteléséhez kell.

A Random Forest modell felhasználása

A második lépésben használjuk fel az eddigiekben többször is megemlített random forestet modellt. Az új adathalmazzal már tudjuk tanítani a modellt, mivel az új adat lényegében két osztályba tartozó elemeket foglal magában. Ha erre tanítjuk a fáinkat, akkor azok célja, hogy megkülönböztessék a szintetikust az igazitól. Ezáltal a gépi tanulás rátanul a rejtett mintázatokra, hiszen az egyikben ezek léteznek a másikban pedig nem. Ez lényegében egy felügyelettel történő tanítás, ebben a megvalósításban.

A random forest algoritmus megvalósítása a TensorFlow könyvtár felhasználásával történt, így meglehetősen megkönnyebbítve a megvalósítást. A függvények segítségével beállítjuk, hogy az adathalmazban szereplő „Szintetikus” és „Normál” címkékre végezze a tanítást. Az erdőben természetesen megszabhatjuk, hogy hány fánk legyen és ezzel a pontosságot tudjuk növelni. Természetesen ez a pontosság számítási igény megnövekedésével jár, ezért figyelembe kell vennünk ilyenkor a számunkra elérhető teljesítményt is, nem csak a végeredményt.

Ezután a korábban kettéválasztott adathalmaz második felét használjuk fel a tesztelésre, amiben csak „Normál” címkével ellátott elemek vannak. A tesztelés eredménye azt fogja megadni, hogy az eredeti adatunk szerkezetében mennyire nagy összefüggések vannak. Minél nagyobb a pontosság, amit elértünk a modellel, annál nagyobb volt az eredeti adathalmazban is az összefüggés. Tehát ez egy jó visszajelzés arra, hogy eddig hogyan állunk. Az eredmény egy jó visszacsatolása annak, hogy az adathalmaz, amit a legelején kiválasztottunk és a bemenetek, amiket kitaláltunk és megvalósítottunk mennyire jók. A tesztelés eredményét felhasználva tudunk továbblépni a következő fázisba.

A hasonlósági mátrix felhasználása

Az erdő teszteléséből származó melléktermékként kitudjuk nyerni a hasonlósági mátrixot. Az 5.3. ábrán látható, egy ilyen hasonlóság mátrix az ábrán lévő értékeknél tizedespontok vannak, mivel a fejlesztő környezet angol. A mátrix a modell egyik teszteléséből származik. Ahogy az ábrán is jól észrevehető a mátrix átlója egyesekből áll és átlóra szimmetrikus értékeket tartamaz.

```
The shape of proximity is (3006, 3006)
[[1.    0.042 0.037 ... 0.037 0.001 0.003]
 [0.042 1.    0.668 ... 0.736 0.001 0.17 ]
 [0.037 0.668 1.    ... 0.622 0.001 0.194]
 ...
 [0.037 0.736 0.622 ... 1.    0.001 0.137]
 [0.001 0.001 0.001 ... 0.001 1.    0.004]
 [0.003 0.17  0.194 ... 0.137 0.004 1.   ]]
```

5.3. ábra. Hasonlóság mátrix.

Az értékek úgy keletkeznek, hogy a különböző fákat felhasználjuk az erdőben. Ha két elem hasonlít egymáshoz, akkor ugyanazt az utat járják be a fában és végül ugyanabba a csomópontba kerülnek. Abban az esetben, ha ez megegyezik egy másik fánál lévő eredménnyel, olyankor a hasonlóság értékét 1-gyel megnöveljük. Ezt megvizsgáljuk minden mintapárra, hogy ugyanott végezte-e vagy sem a fákban, majd ezt a számot leosztjuk a fák számával. Így egy 0 és 1 közötti értéket kapunk, ahol az 1 azt jelenti, hogy a kettő elem teljesen megegyezik. Emiatt a mátrixunk n × n – es alakú lesz és ebből származnak a mátrix tulajdonságok is. Az átló egyértelműen egyesekből áll, hiszen egy elem önmagával teljesen megegyezik, az átlóra való szimmetria pedig abból származik, hogy „A” ugyanannyira hasonlít „B”-re, mint „B” az „A”-ra.

A mátrix számítási igénye sajnos elég magas és nagyban befolyásolja a tanítás sebességét ezáltal. A hasonlósági mátrixról több információval szolgál a [26].

A klaszterezés megvalósítása

A hasonlósági mátrixszal a kezünkben már könnyedén végezhetünk egy klaszterezést. A mátrix elemei távolságot adnak meg az egyes elemek között, ami egy térben elképzelve redukálható különálló térrészekre. Mivel 9 bemenet lett felhasználva ezáltal egy 9 dimenziós térről beszélünk, amit emberileg sajnos nehéz elképzelni.

Szerencsére a klaszterezés megoldásában is hasznosnak bizonyultak a Python könyvtárak. Az Sklearn könyvtár segítségével ez könnyen megvalósítható volt.

Klaszterezési modellnek a k-medoids módszert implementáltam. Ez a klaszterezési módszer nagyon hasonlít a k-means-re. Több lépésben próbálja az adathalmazt csoportokra bontani, ahogy próbálja köztük minimalizálni a távolságot. A k-medoids az adathalmazban lévő elemek közül választ klaszter központokat és ezekre próbál szűkíteni. Mivel ez a megoldás az elemek közötti különbségen alapszik, ezért a hasonlóság mátrixunk tökéletesen megfelel erre a célra.

Sajnálatos módon a k-medoids klaszterezés egy NP-nehéz probléma és az emiatt használt heurisztikus megoldások gyakran sok számítást igényelnek ezzel a tanítási folyamatot lassítva. A k-medoids megoldására egy lehetséges algoritmus a Partitioning Around Medoids (PAM) [28], melyet én is használtam a megvalósítás során.

A klaszterek száma nem kettő lesz annak ellenére, hogy az lenne a logikus, mert támadást és nem támadásokat keresünk. A klaszterezés folyamat végeredményeképpen a térben elkülönülő elemhalmazok különböző tulajdonságú forgalmakat reprezentálnak. Ebből kiindulva több klaszterre van szükségünk, hogy a valóságot lefedhessük. A klaszterek számát empirikus alapokon haladva 16-nak határoztam meg. Ez az érték tűnt a hatékonyságot és a számítási igényt is figyelembe véve a legelfogadhatóbbnak, mivel a magas klaszterszám nagyban megnöveli a számítási kapacitást, amire szükségünk lenne.

5.2.2 A modell kimenete

Az egész anomália detektor modell tesztelését ezen a klaszterben elhelyezés alapján lehet elvégezni. A klaszterek között lesznek olyanok, amikben támadások lesznek és lesznek olyanok, amikben normális forgalmakat találunk. Ezek után megnézzük, hogy egy adott időegység melyik klaszterbe kerül és ez alapján tudjuk majd, hogy támadás-e vagy sem.

Az anomália detektor egy támadás érzékelése esetén továbbítja az adatokat a második fokozatban található detektoroknak.

5.3 A második fokozatban található vektordetektorok

A második fokozat elemei azt a célt szolgálják, hogy ha már a támadást érzékelt az előző része a detektornak, akkor megtudjuk azt, hogy ebben a rosszindulatú tevékenységben mégis milyen különböző vektorokat használt fel a támadó.

Ezáltal nem csak a támadást ismerjük meg jobban, de a DDoS detektor pontosságát is növeljük, hiszen annak a valószínűsége, hogy tényleges támadás történt nagyobb, ha az első fokozaton kívül a másodikban valamelyik vektordetektor felismeri a saját vektorját a gyanús forgalomban.

Ezeknek a működése más elven működik, mint az anomália detektoré, mivel itt egy konkrét tudott dolgot keresünk. A más probléma okán más megoldási módszer is lett alkalmazva. Ahogy azt már korábban is említettem itt ANN típusú modellt használtam. A három különböző vektornak, amikre vannak a különböző detektorok, van egy-egy jellegzetes tulajdonsága és ezek alapján próbáltam őket megtalálni.

5.3.1 A TCP(ACK) elárasztás vektort detektáló ANN modell

A TCP(ACK) típusú támadás egy elterjedt opció a rosszakarók körében. A támadásnak az a lényege, hogy a támadó olyan hamis TCP üzeneteket küld, ahol az ACK flaget bekapcsolják, de ezek az üzenetek nem tartoznak semmilyen folyamatban lévő kommunikációhoz és ezáltal a tűzfalat és a szervert kényszerítve folyamatos keresésre, hogy mégis melyik meglévő folyamhoz tartozhatnak az üzenetek.

A tanításhoz és teszteléshez ennél a modellnél is valós internetes forgalomból származó adathalmazt használtam fel. A megvalósított gépi tanulási modell pedig a Keras könyvtárban található szekvenciális modell, amit a könyvtárnak köszönhetően könnyedén tudtam implementálni. A modell dolga ebben az esetben csak annyi, hogy eldöntse az adott adathalmazról beletartozik-e az osztályba vagy sem.

Az ilyen típusú támadásoknak van egy jellegzetessége, amit kitudunk használni a felderítésük során. Az ACK flagek arányának eltolódása a megszokottól jól jelzi ezt a fajta támadást. Természetesen, mint mindig ez nem feltétlenül elég, mivel egy magas felbontású videó stream-nél is az ACK arány elérheti a 0,98-at.

A felhasznált bementek és a modell megvalósítása

Itt már kevesebb bemenet is bőven elegendő lehet. Ennek oka abban keresendő, hogy egy kifejezetten specifikus dolgot keresünk. Egy garantált bemenet lesz az ACK flagek aránya. Emiatt szükségünk van arra, hogy megvizsgáljuk a TCP flagek közül melyikek vannak beállítva. Ahol az ACK flagek be vannak állítva azokat összeszámolom és ezután megvizsgálom, hogy melyek azok, amiket mi kaptunk. A két értéket elosztva megkapjuk az arányt. A másik bemenet, ami felhasználásra kerül az a kapott csomagok

száma. Ezek a bemenetek is az eddigiekben használt időegységes alapon működnek. A CSV fájlok beolvasása után az adatokból tensor készül és kerül a modell bemenetére. Mivel ez egy felügyelt tanítást alkalmazó metódus, ezért szükség van az adat címkézésére. A címkézés az anomália detektornál használtal megegyezik.

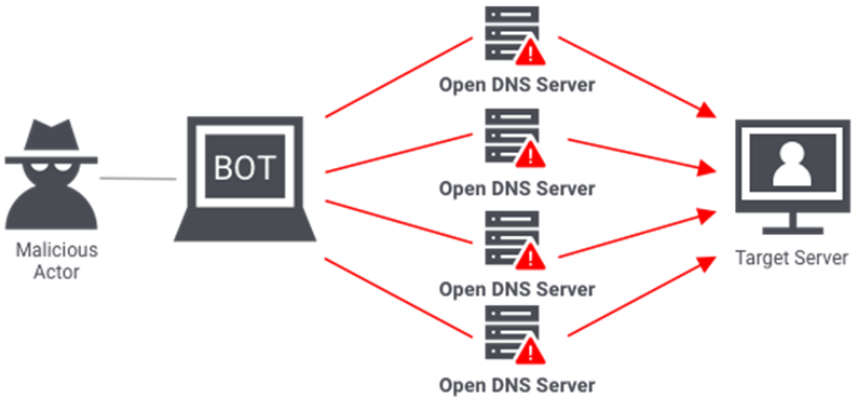
```
model = Sequential()
model.add(Dense(12, activation='relu', input_shape=(2,)))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

5.4. ábra. A TCP(ACK) támadás detektorának megvalósítása.

Az 5.4 ábrán látható a Python kód, amivel implementálva lett a Keras-os szekvenciális modell. Jól látható, hogy három réteg található a felhasznált modellben, van egy bemeneti, egy rejtett és egy kimeneti. A „dense” azt jelenti, hogy rétegenként az összes neuron össze van kötve a következő réteg neuronjaival. A függvényekben lévő számértékek: 12 – 8 – 1 a neuronok számát jelenti rétegenként. A bemenet méretét az „input_shape” határozza meg, amit már fentebb kifejtettem, hogy ezekből kettő lesz. A kimenetünk pedig egy sigmoid lesz, ami azt jelenti, hogy a kimeneti érték „0” vagy „1”. A kimenet annak feleltethető meg, hogy beletartozik-e az osztályba vagy sem.

5.3.2 A DNS Amplified vektort detektáló ANN modell

Az előző módszerhez nagyon hasonlóan épül fel ez a megoldás is. A kivételt az képzi, hogy más a tulajdonsága a támadásnak és ehhez más bemenetre van szükség. A DNS felerősítéses támadás esetén a támadó ismert DNS szervereknek küld DNS lekéréseket, viszont a célpont IP címét adja meg, ezzel elérve, hogy a szerverek neki válaszoljanak. Ilyen módon kis mennyiségű eszközzel is nagy támadást lehet végrehajtani. Az 5.5. ábrán ennek a támadásnak egy vizualizációja látható, ahogy a támadó egy botnet segítségével generált forgalom erejét felerősíti a szerverek felhasználásával és így egy sokkal nagyobb támadást tud véghez vinni.



5.5. ábra. DNS Amplified támadás. [27]

A támadás típust az jellemzi a legjobban, hogyha olyan DNS válaszokat kapunk, amire mi sosem küldtünk DNS kéréseket. Emiatt ez lesz az egyik felhasznált bemenet. A kérdéses értékek kiszámítását úgy végeztem, hogy megvizsgáltam hány olyan különböző IP cím van, ahonnan DNS válaszok érkeznek anélkül, hogy mi küldtünk volna előtte bármilyen kérést. A másik két bemenet a küldött és fogadott csomagok aránya és az összes csomag, ami érkezett. Az eddigiekhez hasonlóan, itt is ugyanúgy az időegységes felbontást alkalmaztam az adatok elkészítéséhez. A kiindulási adathalmaz ugyanúgy valós forgalomból származott és a címkézést az ezelőtti modellekhez megfelelő módon végeztem.

```
Initiation  Ratio   Sum
      13.0    1.0  15.0
```

5.6. ábra. DNS Amp. detektor modell bemenetei.

```
model = Sequential()
model.add(Dense(12, activation='relu', input_shape=(3,)))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

5.7. ábra. DNs Amp. támadás detektorának megvalósítása.

Az 5.6 és 5.7. ábrákon látható a modell egy bemenete és a felépítése az alkalmazott modellnek. Könnyen észrevehetjük, hogy az itt használt modell megegyezik az előzővel, azzal a kivétellel, hogy itt a bemenetek száma háromra van állítva. Az „Inititation” nevű bemenet a jellegzetes tulajdonságnak számító kéretlen DNS válaszokat küldő IP címek száma.

5.3.3 Az ICMP Echo Request vektort detektáló ANN modell

A három modell közül ez a legegyszerűbb. A támadásnál az a cél, hogy az áldozat gépe rengeteg pinggel legyen megterhelve. Ennek a támadásnak a felismerése nem igényel nagy erőfeszítést. Itt csak arra kell figyelnünk, hogy a megengedett viselkedés minél emberibb legyen. A modell implementációjának oka az, hogy rengeteg helyen ez ellen a támadás ellen úgy védekeznek, hogy ezt a fajta forgalmat teljesen letiltják, ami nem túl elegáns.

Két bemenetet választottam ennél a modellnél. Az egyikben az ICMP Echo Request üzenetek számát számolom a másikban pedig azt, hogy ezek hány helyről származnak. Ilyen módon mindkét támadási opciót kivédhetjük, azt is amikor kevés helyről sok üzenet érkezik és azt is amikor sok helyről kevesebb. A megvalósításban elegendő nagyobb időegységekben vizsgálódni, mert úgysem engedünk meg sok ilyen fajta üzenetet.

Ennél a modellnél a többitől eltérő módon nem valós forgalmat használtam, hanem általam generált adatok alapján dolgoztam a feladat egyszerűségére tekintettel és amiatt, hogy ilyen támadási minták nem álltak rendelkezésemre megfelelő számban. A címkézést a másodpercenkénti 10 üzenetnél határoztam meg, vagyis az efölötti értékeket támadásnak vettem és az alattiakat pedig nem. A megvalósított modell megegyezik az 5.4. ábrán látható TCP(ACK) támadás detektorának modelljével, csak a bemenetekre más adatok kerültek betöltésre.

6. A modell validálása és optimalizálása, kiértékelési eredmények

A tesztelésekre és tanításokra szerencsére rengeteg adat állt rendelkezésemre, ezért ezt az általánoshoz közeli 70-30% arányban osztottam szét. A támadásos és normális forgalomból is egyaránt igaz ez az állítás. Emiatt a gyakran használt módon a feldogozott adatokból készült adathalmazt csak felosztottam valamilyen arány szerint, majd aztán ezzel dolgoztam. Viszonyítási alapként az anomália detektornál a kész időegységekből rendelkezésemre állt több mint 7000 hagyományos forgalom és 800 támadásos minta, ezzel közel 8000 minta állt rendelkezésemre, tehát a felosztások után is megfelelő mennyiség maradt. A modellek hiperparamétereit, ahogy azt már korábban is említettem, a tesztelések eredményét felhasználva empirikus módon optimalizáltam.

6.1 Az anomália detektor tesztelése

A tanítása során a felosztott rész duplája került a modellbe, hiszen a szintetikus adatok száma megegyezett az eredetiek számával. Az egész adathalmazt a szétválasztások előtt természetesen összekevertem, hogy véletlenszerű eloszlással kerüljön mindenféle adat a tanításba és a tesztelésbe is. Ezek után a random forest modellnek adtam a tanításhoz szükséges adatokat.

```
Out-of-bag accuracy: 0.9504
Test accuracy: 0.9787
```

6.1. ábra. A random forest tesztelési eredménye.

A 6.1. ábrán látható ennek a tesztelésnek az eredménye. Jól látható, hogy igen magas pontosságot tudtam elérni, ami megközelítette a 98%-ot. Korábban már említettem, hogy ez az eredmény megmutatja az eredeti adathalmazunkban mennyire erős összefüggések voltak és milyen sikerességgel tudja a valós és szintetikus forgalmat megkülönböztetni az erdő.

A felhasznált fák mennyiségét 1000-ben határoztam meg. Emellett a mennyiség mellett ezt a magas eredményt sikerült elérni és a számítási igény meglehetősen alacsony volt. A fák számának további emelése nem hozott lényegbeli változást az eredményekben.

Ezek után jött a hasonlósági mátrix készítése. Az adathalmazomat a tesztek során általában ennek megfelelően vágtam fel és arra ügyeltem, hogy 5000 elem fölé ne menjen. Az eredmény ilyenkor már egy 5000 × 5000 mátrix, aminek a számítási igénye igen magas volt. A korábban már bemutatott 5.3. ábrán látható volt egy ilyen tesztelés során létrehozott mátrix, itt a mátrix 3006 elem alapján készült.

Ezután következett a klaszterezés, aminek az eredménye a modell végleges eredményét is megadta. Próbálkoztam még a k-medoid klaszterezés mellett a k-means klaszterezéssel is. A két módszer között nem voltak nagy eltérések, de a k-medoids használatánál nagyobb pontosságot értem el. A modell megvalósításánál már említettem, hogy 16 klasztert használtam. Ennél magasabb klaszter szám nem hoz sokkal jobb eredményeket, viszont a számítási időt nagyban megnöveli.

```
[[  41    0  313    1   29  641   52   83    9  134  203 1016   38   26
    89    8]
 [  33    7    7   21   30    4   16   26   32   19    9    4   29   25
    16   45]]
Purity:
0.9218230206254159
```

6.2. ábra. A klaszterezés eredménye.

A 6.2. ábrán látható egy klaszterezés végeredménye, ami egyben az egész anomália detektor modell végeredménye. Az ábrán található két tömbben felsorolt elemek aszerint vannak, hogy a 16-ból éppen melyik klaszterbe kerültek. A felső tömb a normális forgalom és az alsó pedig a támadásos forgalom elemei. Jól látható, hogy nem minden klaszter tisztasága megegyező. Vannak klaszterek, amik sajnos elég pontatlan eredményt értek el. A végeredmény a kapott klasztereknek a tisztasága. Ebben a tesztben 92% értem el. Ennek számítása úgy történik, hogy minden klaszterben megszámoljuk a leggyakoribb elem számát majd ezeket összeadjuk és elosztjuk az összes elem számával. Ezzel az egész modellünk pontosságát kapjuk meg, ettől függetlenül lehetnek klaszterek, amik ennél jobb vagy rosszabb eredményt értek el önmagukban.

6.2 A vektordetektorok tesztelése

Ezeknek a modelleknek a tesztelése meglehetősen hasonló módon zajlott. A rendelkezésre álló adathalmazt két részre bontottam, hogy az egyikkel tanítsak a másikkal teszteljek. A rendelkezésre álló adat nagyrésze a támadás típusból állt, hogy a detektor arra a kifejezett vektorra pontosan tudjon rátanulni.

A TCP(ACK) vektor detektor tesztelésénél viszont extra teszteket is végeztem. Ehhez a támadáshoz nagyon hasonló normális hálózati forgalmat könnyedén lehet rögzíteni. A magas felbontású videótartalmak erre a célra pont megfeleltek. A Wireshark segítségével több 4K felbontású videó forgalmát rögzítettem. Az ilyen típusú forgalomnál az ACK arány eléri a 0,98-at is, ezáltal pont a támadás legjellegzetesebb tulajdonságát imitálva. A forgalom sok csomagot generál, így ez még egy érv a hasonlóságra. Az új adatokat feldolgoztam ugyanazzal a módszerrel ahogy ezelőtt, majd ezekkel is teszteltem.

```
[5079 rows x 2 columns]
159/159 [==============================] - 0s 939us/step - loss: 0.8583 - accuracy: 0.9474
[0.8582741022109985, 0.9474306106567383]
```

6.3. ábra. A videó forgalmon végzett teszt eredménye.

A 6.3. ábrán látható a videóforgalmon végzett tesztek eredménye, amik közel 95% pontosságot értek el 5000 elemen. A tesztelés szélsőséges körülményéhez képest igen magas pontosságot tudtam elérni. A többi vektort detektáló modell eredmény is hasonlóan magas értékeket produkált. De ennél a fokozatnál ez volt az elvárt is, mivel itt egy adott specifikus dolgot keresünk és ezért erre pontosan rátudjuk fókuszálni a modellünket.

A neuronok és rétegek számának növelése nem hozott nagy változásokat, ezért a számítási igényre tekintettel nem növeltem azokat tovább. Emellett a tapasztalatom az, hogy a tanító adat sokkal többet számít, mint a modell nagysága, ezért jobban megéri az adatunk tisztítására és feldolgozására több időt fordítani.

7. Felhasználási és továbblépési lehetőségek

A DDoS detektort bármilyen környezetben lehet használni. Az egyetlen tényező amire ügyelni kell, hogy a tanításnál az implementáció helyének megfelelő forgalommal kell tanítani, különben a pontosság nem feltétlen lesz megegyező az elvártakkal.

A jelenlegi detektor modellektől számos ponton tovább lehet lépni. Természetesen a pontosságot az örökkévalóságig lehet finomítani. Ezen kívül még két kiegészítés lehet fontosabb.

Jelenleg a második fokozatot nekünk kell feltölteni kézzel, ha új vektorokat felismerő elemeket szeretnénk implementálni. Ez nem túl hatékony a vektorok számára nézve és arra, hogy mindig keletkezhetnek új vektorok, ahogy a támadók kreativitása növekszik. Az effektivitás növelés érdekében egy automatizált módszert lehet kitalálni, amely az újonnan talált támadásmintázatokra vektordetektorokat generál. Ezzel az új támadási típusokat felfedezhetjük és jobb védekezést találhatunk ki ellenük.

A másik problémakör, aminek megoldása nagyban növelné a detektor hatékonyságát az a folyamatos tanítás megoldása. Mivel a forgalmi mintázatok gyorsan változnak, ezért gyakran újra kell tanítanunk a modelleket. Ez leginkább az anomália detektorra igaz. Egy tényleges megvalósításnál nem engedhetjük meg magunknak, hogy tanítási célból kikapcsoljuk a rendszerünket, emiatt ezt úgy kell megoldani, hogy az menet közben is tanítható legyen.

8. Összefoglalás

Az internetes szolgáltatások növekedésével és terjeszkedésével új fenyegetések jelentek meg. A DDoS támadások komoly problémát jelentenek ezeknek a cégeknek, hiszen ezek mindennaposak és hosszú kiesést okozhatnak a szolgáltatásban. Az elmúlt időszakban ezek nagysága és gyakorisága is sokat emelkedett. Megjelentek több multivektort tartalmazó támadások, ami a védekezést tovább nehezíti. Sajnos az elhárításuk nehéz és sok munkát igényel, mert a támadások bonyolultságának változásával más és más módszerek szükségesek, hogy megakadályozzuk a rosszindulatú tevékenységek sikerét. A detekció nehézségét az okozza leginkább, hogy a hétköznapi felhasználók forgalmát megkülönböztessük a támadóétól. A növekvő szolgáltatások mellett az internetképes eszközök száma is drasztikusan növekszik, ezáltal a botnetek potenciális növekedését előidézve és a támadások erejét növelve.

A dolgozat során a multivektor DDoS támadások azonosításának problémájára adtam egy megoldást a manapság nagy népszerűségnek örvendő gépi tanuláson alapuló módszerek felhasználásával. A mesterséges neurális hálózatokon alapuló modellek könnyebben tudják követni a támadások fejlődését, változását.

A megvalósított többfokozatú detektort Pythonban implementáltam, teszteltem és validáltam. A tanítások és tesztelések valós forgalomból származó adatokkal történtek. A detektorban felhasznált modellek a megoldandó problémákon alapulóan más és más megvalósítást igényeltek. A detektor fejlesztésére még bőven van lehetőség a jövőben, hiszen nemcsak a pontosságot lehet növelni, de új megközelítésekkel és kiegészítésekkel az egész detektor megbízhatósága és hatékonysága is növelhető.

Irodalomjegyzék

[1] Pei, J., Chen, Y., & Ji, W. (2019). A DDoS Attack Detection Method Based on Machine Learning. Journal of Physics: Conference Series, 1237, 032040. doi:10.1088/1742-6596/1237/3/032040

[2] Singh, K. J., & De, T. (2015). An Approach of DDOS Attack Detection Using Classifiers. Emerging Research in Computing, Information, Communication and Applications, 429–437. doi:10.1007/978-81-322-2550-8_41

[3] Das, S., Mahfouz, A. M., Venugopal, D., & Shiva, S. (2019). DDoS Intrusion Detection Through Machine Learning Ensemble. 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C). doi:10.1109/qrs-c.2019.00090

[4] Fouladi, R., Kayatas, C., & Anarim, E. (2018). Statistical Measures: Promising Features for Time Series Based DDoS Attack Detection. Proceedings, 2(2), 96. doi:10.3390/proceedings2020096

[5] Liang, X., & Znati, T. (2019). An empirical study of intelligent approaches to DDoS detection in large scale networks. 2019 International Conference on Computing, Networking and Communications (ICNC). doi:10.1109/iccnc.2019.8685519

[6] DDoS in a Time of Pandemic, https://www.netscout.com/sites/default/files/2021-04/ThreatReport_2H2020_FINAL_0.pdf Utolsó letöltés: 2021-11-25.

[7] The rise of multivector DDoS attacks, https://blog.cloudflare.com/the-rise-of-multivector-amplifications/ Utolsó letöltés: 2021-11-25.

[8] Marek Majkowski (2018) Multivector Attacks, https://blog.cloudflare.com/the-rise-of-multivector-amplifications/ Utolsó letöltés: 2022-10-30.

[9] Common DDoS Attack Types, https://www.corero.com/blog/glossary/ Utolsó letöltés: 2021-11-25.

[10] Kolias, C., Kambourakis, G., Stavrou, A., & Voas, J. (2017). DDoS in the IoT: Mirai and Other Botnets. Computer, 50(7), 80–84. doi:10.1109/mc.2017.201

[11] Greenstein, S. (2019). The Aftermath of the Dyn DDOS Attack. IEEE Micro, 39(4), 66–68. doi:10.1109/mm.2019.2919886

[12] What is DDoS blackhole routing?, https://www.cloudflare.com/learning/ddos/glossary/ddos-blackhole-routing/ Utolsó letöltés: 2022-06-16.

[13] DDoS How A WAF Works, https://www.cloudflare.com/learning/ddos/glossary/web-application-firewall-waf/ Utolsó letöltés: 2022-10-30

[14] What is a Web Application Firewall (WAF)?, https://www.cloudflare.com/learning/ddos/glossary/web-application-firewall-waf/ Utolsó letöltés: 2022-06-17.

[15] What is a DDoS attack?, https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/ Utolsó letöltés: 2021-11-25.

[16] Priya, S. S., Sivaram, M., Yuvaraj, D., & Jayanthiladevi, A. (2020). Random Forest Information, Machine Learning based DDOS Detection. 2020 International Conference on Emerging Smart Computing and Informatics (ESCI). doi:10.1109/esci48226.2020.9167642

[17] Ho, Tin Kam (1995). *Random Decision Forests*. Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282.

[18] Priya, S. S., Sivaram, M., Yuvaraj, D., & Jayanthiladevi, A. (2020). *Machine Learning based DDOS Detection. 2020 International Conference on Emerging Smart Computing and Informatics (ESCI).* doi:10.1109/esci48226.2020.9167642

[19] WANG, Sun-Chong. Artificial neural network. In: *Interdisciplinary computing in java programming*. Springer, Boston, MA, 2003. p. 81-100.

[20] ABIODUN, Oludare Isaac, et al. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 2018, 4.11: e00938.

[21] A sequential neural model (Keras Sequential Api, n.d) , Jayawardana, J. R., & Bandaranayake, T. S. (2021). Analysis of optimizing neural networks and artificial intelligent models for guidance, control, and navigation systems. International Research Journal of Modernization in Engineering, Technology and Science, 3(3), 743-759.

[22] Mehmood, T., & Rais, H. B. M. (2016). *Machine learning algorithms in context of intrusion detection. 2016 3rd International Conference on Computer and Information Sciences (ICCOINS).* doi:10.1109/iccoins.2016.7783243

[23] Priya, S. S., Sivaram, M., Yuvaraj, D., & Jayanthiladevi, A. (2020). *Machine Learning based DDOS Detection. 2020 International Conference on Emerging Smart Computing and Informatics (ESCI).* doi:10.1109/esci48226.2020.9167642

[24] Peneti, S., & E, H. (2021). *DDOS Attack Identification using Machine Learning Techniques. 2021 International Conference on Computer Communication and Informatics (ICCCI).* doi:10.1109/iccci50826.2021.9402441

[25] Afanador, N. L., Smolinska, A., Tran, T. N., & Blanchet, L. (2016). Unsupervised random forest: a tutorial with case studies. Journal of Chemometrics, 30(5), 232–241. doi:10.1002/cem.2790

[26] Englund, C., & Verikas, A. (2012). A novel approach to estimate proximity in a random forest: An exploratory study. Expert Systems with Applications, 39(17), 13046–13050. doi:10.1016/j.eswa.2012.05.094

[27] Volumetric attacks, What is a DDoS Attack?, https://www.onelogin.com/learn/ddos-attack Utolsó letöltés: 2022-10-30

[28] Partitioning Around Medoids (Program PAM). (n.d.). Wiley Series in Probability and Statistics, 68–125. doi:10.1002/9780470316801.ch2