Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai kar

Anonim rendszer botnet forgalom felismerésére és szűrésére

Készítették:

Faragó Márton, V. VIK

farmar@freemail.hu

Mészáros Tamás, V. TTK

slovi@math.bme.hu

Konzulensek:

Szentgyörgyi Attila, BME-TMIT

Kenyeres Péter, BME-TMIT

TDK 2009

Tartalomjegyzék

1) Bevezető....4
2) Motiváció....6
3) Elméleti alapismeretek....8
3.1) Botnetek....8
3.1.1) A botok életciklusa....9
3.1.2) A botnetek terjedése....9
3.1.3) Támadási formák....10
Elosztott szolgáltatás-megtagadással járó támadás....10
Spamküldés....11
Adathalászat (phishing)....11
Hálózat-megfigyelés (sniffing)....12
Billentyűzet-megfigyelés (keylogging)....12
Internetes reklámok és rangsorolás....12
Elosztott számítási feladatok....13
3.1.4) Command & Control módszerek....13
Centralizált C&C....14
Láncolt C&C....15
Peer-to-peer C&C....15
Hibrid C&C....16
Véletlen C&C....17
3.1.5) Gyülekezési módszerek....17
Forráskódba írt („hard coded”) IP-cím....17
Dinamikus DNS domain-név....18
Elosztott DNS szolgáltatás....18
Dinamikus domain-név....18
3.2) NetFlow....18
3.3) Peer-to-peer hálózatok és a DHT....20
3.4) Secure Overlay Service....21
3.5) Klaszterezés....21
3.5.1) K-közép algoritmus....22
3.5.2) X-közép algoritmus....22
3.6) Outlier szűrés....24
4) A rendszer bemutatása....26
4.1) A rendszer megtervezése....26
4.2) A rendszer felépítése....28
4.2.1) A rendszer komponensei....28
Ügynökök....28
Honeypotok....29
Terjesztők....29
Adatfeldolgozó....29
4.3) Kiértékelés és mintakészítés....29
4.3.1) Aggregálás....29
4.3.2) Klaszterek azonosítása és mintakészítés....31
4.3.3) Több forrás esete....32
4.4) C&C felismerés....32

5) A rendszer tesztelése....33
5.1) A teszthálózat felépítése....34
5.1.1) Virtuális rendszert futtató számítógépek....34
5.1.2) Gateway....35
5.1.3) Algoritmus futtató....36
5.1.4) Terjesztő....36
5.2) A tesztelés folyamata....36
5.2.1) A tesztelt vírusok....37
Agobot....37
Beatrix....40
Sasser....42
5.2.2) Tesztsorozatok bemutatása....42
5.3) Mérési eredmények....43
5.3.1) Vírusok vizsgálata....43
5.3.2) Valós forgalmi teszt....44
5.4) További eredmények....46
5.4.1) Aggregálás....46
5.4.2) Feldolgozási idő....48
5.5) Konklúzió, továbbfejlesztési javaslatok....48
6) Összefoglalás....50
7) Irodalomjegyzék....51

1 Bevezető

Az elmúlt évtizedben a felhasználókat és szolgáltatókat is érintő, Interneten fellelhető fenyegetések nagy része átalakult az egyes számítógépeket célzó vírusokból olyan elosztott támadási formává, amely akár egész hálózatokat is használhatatlanná tehet. A botnet valamilyen vírussal megfertőzött ún. zombi gépek hálózata, amely általában egy ember vagy csoport irányítása alatt áll. A parancsok valamilyen vezérlő (C&C – Command&Control) csatornán keresztül jutnak el a többi, a rendszer irányításban résztvevő, többnyire szintén fertőzött géphez, melyek aztán továbbítják őket az egyes hosztokhoz, melyeket botoknak nevezünk.

A botneteket többnyire ártó szándékkal hozzák létre: leggyakrabban elosztott szolgáltatás-megtagadással járó támadásra (DDoS – Distributed Denial-of-Service), kéretlen reklámokat, adathalász programot vagy trójait tartalmazó e-mail küldésére, személyes adatok ellopására használják. A terjeszkedéshez általában az Internetre csatlakozott egyes számítógépek sebezhetőségeit használják ki, ezáltal megfertőzve azokat.

A fertőzött gépek tulajdonosai általában nincsenek tisztában azzal, hogy tagjai lennének egy ilyen típusú hálózatnak: a vírus a háttérben megbújva fut a gépükön, anélkül, hogy tudnának róla. Bár a vírusirtók képesek lehetnek ezen vírusok azonosítására és eltávolítására, ám sokan nem használnak megfelelő védelmet, így a botnet hálózat kiépülését nehéz megakadályozni. Bár a botnet támadások detektálása viszonylag egyszerű, magának a hálózatnak a teljes felszámolása vagy akár megbénítása komoly kihívásokkal jár.

A dolgozatunkban egy olyan architektúrát mutatunk be, amely különböző hálózatokból származó NetFlow alapú aggregált forgalmi adatok között képes felismerni a botnetek forgalmát. Ezt követően az összegyűjtött adatokat – az IP címekre és portokra vonatkozó információk figyelmen kívül hagyásával – anonimizálja és továbbítja más, ugyanezen rendszert használó hálózatok irányába, ezzel elősegítve az azokban található esetleges további fertőzött számítógépek felderítését. Az adatok anonimitása a rendszer egyik legfőbb erénye, hiszen a tagoknak többnyire nem érdeke saját forgalmi statisztikájuk nyilvánosságra hozatala. A skálázhatóság és globális elérhetőség érdekében az architektúra nagy mértékben épít a peer-to-peer (P2P) hálózatok elosztott hash tábláira (DHT).

A rendszer kipróbálására egy teszthálózatot terveztünk és hoztunk létre, amelyen különböző botnetek forgalmának vizsgálatával ellenőriztük a helyes működést. Az algoritmus tesztelésével

megmutattuk, hogy rendszerünk alkalmas botnetek forgalmának kiszűrésére, az adatok anonimizálására és megbízható továbbítására más, a rendszerben résztvevő felek irányába.

Dolgozatunk következő részében bemutatjuk azon motiváló tényezőket, mely a rendszer megalkotására ösztönöztek. Ezután áttekintést adunk az architektúra kidolgozásához felhasznált elméleti háttérről. A negyedik fejezet az általunk tervezett rendszer elemeit és szolgáltatásait mutatja be, majd a rendszer teszteléséhez felépített teszthálózat, a mérések során vizsgált vírusok és a mérési eredmények leírása következik. Dolgozatunkat a mérések vizsgálata alapján levont következtetések bemutatásával, végül az összegzéssel zárjuk.

2 Motiváció

A hacker csoportok elsődleges céljai az Internet széles körű elterjedése óta megváltoztak: míg régebben az öncélú rombolás illetve az önkifejezés volt a fő motiváló tényező, addig mára a számítógépek feltörésével és megfertőzésével céljuk az anyagi haszonszerzés [1]. Az egyéni felhasználók illetve vállalatok ellenében történő, növekvő mértékű Internetes támadások egyre összetettebbek, és többször hatalmas pénzügyi veszteséget vagy akár nagyobb üzleti zavarokat is okozhatnak a világban. Az FBI 2006-os tanulmánya szerint az amerikai vállalatoknak összesen évi 67 milliárd dollárjába kerül a vírusok, kémprogramok és más, számítógépes támadási formák elleni védekezés [2].

Napjainkban az ilyen összetettebb támadási formák hátterében leggyakrabban botnetek állnak. A jelenleg ismert legnagyobb ilyen a Conficker vírus és variánsai által megfertőzött, több mint tízmillió zombigépből álló hálózat [3]. Ez fertőzte meg többek között a Francia Tengerészgyalogság, az Angol Védelmi Minisztérium, és a Német Hadsereg számítógépes rendszerét is, bejutva katonai létesítményekbe, hivatali számítógépekbe, anyahajókra, tengeralattjárókra, valamint kórházakba [4][5][6].

A botnetek terjedése a megfelelő védekezés híján igencsak gyors lehet, az F-Secure elemzőinek mérései alapján 2009 februárjában a Conficker egyik variánsával, a Downadup vírussal fertőzött gépek száma négy nap alatt 2,8 millióról 8,9 millióra ugrott [7]. Az ilyen gyors terjedés okai között megtalálhatóak a nem megfelelően frissített – többnyire nem legális forrásból származó – operációs rendszerek, illetve a tűzfal vagy vírusírtó programok hiánya. Az ilyen felhasználói magatartás miatt nehéz megfékezni a botnetek terjedését, hiszen egymástól teljesen független számítógépek lesznek áldozatai a fertőzésnek.

A botnetek által indított támadások skálája igen széles, a hálózati forgalom megfigyelésével azonban ezek közül már több hatékonyan kivédhető [8][9][10][11]. A nehézséget a botnetek felszámolása vagy legalább a megbénítása okozza. Az ezzel foglalkozó kutatások során többféle módszert dolgoztak ki, melyek segítségével a hálózati forgalom megfigyelésével észlelni lehet a botneteket. Ezek a módszerek nagyrészt az IRC vagy HTTP alapú vezérlő csatornákat használó botnetek felderítésére alkalmasak. Sokszor azonban nincs lehetőség a hálózaton átmenő adatcsomagok teljes tartalmának vizsgálatára, ezen kívül pedig nagysebességű hálózatok analizálása esetén gondokat okozhat az adatok tárolása és feldolgozása. A jelenleg használt technológiák

további hátránya, hogy egy hálózatból érkező adatok feldolgozásával működnek. A botnetek elosztott jellege miatt azonban több hálózatra lenne szükséges kiterjeszteni a megfigyelést.

Az általunk bemutatandó módszer nemcsak megfigyeli a hálózat forgalmát, majd detektálja és megjelöli a fenyegetéseket, hanem feldolgozza a bejövő anonimizált adatokat, és az eredményeket szétosztja a rendszer felhasználói között. Ily módon rendszerünk képes gyors és hatékony módon reagálni az újonnan felbukkanó fertőzésekre.

3 Elméleti alapismeretek

Ebben a fejezetben az architektúránk megalkotásához szükséges alapismereteket mutatjuk be. Elsőként a botnetekről lesz szó egy bővebb leírás keretében melyet egy rövid áttekintéssel kezdjük a botnetekről és azok múltjáról, majd ezt követően felvázoljuk egy átlagos bot életciklusát, végül a botok működésének legfontosabb területeit mutatjuk be. Ezt követően a NetFlow, valamint az erre épülő monitorozás bemutatása következik, mely rendszerünk alapját képezi. A harmadik alfejezet a peer-to-peer rendszerekről valamint a velük szoros összefüggésben lévő elosztott hash táblákról szól, amit a Secure Overlay Service bemutatása követ. A fejezetet a klaszterezésről és az outlier szűrésről szóló matematikai összefoglaló zárja le.

3.1 Botnetek

A botnet elnevezés a robot és a network szavakból tevődik össze. Olyan fertőzött gépek hálózatát értjük alatta, mely egy hacker, vagy azok egy csoportjának irányítása alatt áll. Őket más néven botmasternek nevezzük. Bot vagy másnéven zombi alatt azon fertőzött számítógépet értjük, mely valamely botnethez tartozik.

Az első bot, a *PrettyPark* féreg 1999-ben jelent meg. A féreg támadási módjai nem haladták meg akkori férgek repertoárját – mely főképp különféle felhasználói- és rendszeradatok megszerzéséből állt –, ám ez volt az első, mely lehetővé tette megalkotójának, hogy egy *IRC (Internet Relay Chat)* [12] szerverhez csatlakozva távolról vezérelje a féreg által megfertőzött számítógépeket. Az IRC *C&C*, azaz utasítási és irányítási üzeneteket átvivő csatornaként való használatának forradalmi ötlete gyorsan elterjedt a víruskészítők körében, akik később a módszert tovább tökéletesítették. Az így készült botokat idővel több, fejlettebb funkcióval, összetettebb támadási formákkal bővítették, így a belőlük kiépült nagy kiterjedésű hálózatok egyre nagyobb fenyegetést jelentettek az Internet felhasználóira.

A botok kialakulásával párhuzamosan megjelent a profit iránti igény az addig főleg kihívás vagy szórakozás kedvéért tevékenykedő hackerek részéről. Egyre népszerűbbek lettek ezáltal az elosztott szolgáltatás-megtagadással járó támadások, a spamlevelek, illetve az adatlopások. Az Internetes bűnözők rájöttek, hogy a botnetek használatával az eddigieknél nagyságrendekkel nagyobb erő áll rendelkezésükre céljaik megvalósításához. A gyors meggazdagodás reményében a már elterjedt botok kódjaira építkezve új változatokat hoztak létre, mely a botok számának drasztikus

növekedéséhez vezetett: az egyik legelterjedtebb IRC botnak, az *SDBot*nak [13] csaknem négyezer variánsáról tudtak a szakértők 2004 augusztusában.

A botnetek újabb generációi összetettebb C&C mechanizmusokkal kommunikálnak és veszélyesebb támadásokra képesek mint elődeik, valamint a botok készítésekor immáron felhasználják a programozásban azóta elterjedt új módszereket és irányelveket, így a modularitást is, mely segítségével a vírusírók az újabb fajta támadási formákat pillanatok alatt beilleszthetik saját vírusuk kódjába. Az utóbbi időben történt fejlődéseknek köszönhetően a legújabb botnetek – például a Conficker [14], vagy a Storm [15] – terjedhetnek megosztott hálózatokon, fájlmegosztó rendszereken, *P2P (peer-to-peer)* hálózatokon, illetve más vírus által hagyott kiskapukon, vagy ismert sebezhetőségek kihasználásával is, a botok pedig nem csak IRC, hanem HTTP, P2P, vagy egyéb csatornán keresztül is képesek kommunikálni egymással.

3.1.1 A botok életciklusa

Általános esetben a zombi hálózatok tagjainak életútja hasonló módon alakul. Ez az életciklus általában hat fázisra bontható:

- **Terjesztés:** a botnet üzemeltetője a kártékony kódot különböző csatornákon keresztül terjeszti a botnet meglévő tagjainak segítségével.
- **Fertőzés:** az áldozat számítógépén futtatásra kerül a vírus, aminek hatására az megfertőződik, és így zombigéppé válik.
- **Várakozás:** a bot utasításra vár a C&C csatornán, esetleg periodikusan valamilyen tevékenységet végez (például kéretlen reklámokat tartalmazó üzeneteket terjeszt, vagy sebezhető számítógépet kereses).
- **Támadás:** a bot üzenetet kap a megfigyelt C&C csatornán, majd végrehajtja az utasítást, és újra várakozó állapotba kerül.
- **Megszűnés:** a bot által figyelt C&C csatorna vagy a célszerver elérhetetlenné válik vagy a vírust törlik a számítógépről.
- **Újraéledés:** a bot által figyelt C&C csatorna vagy a célszerver újra elérhetővé válik.

3.1.2 A botnetek terjedése

A botnetek terjedése újabb számítógépek megfertőzésével történik, amely ugyanúgy zajlik mint bármely más vírus esetén – ez nem meglepő, hiszen mint már korábban említésre került, a

botneteket a C&C csatorna használata különbözteti meg a többi rosszindulatú kódtól. A terjeszkedés alapvetően két módon történhet: valamilyen sebezhetőség vagy a felhasználók hiszékenységének kihasználásával.

Az első módszer lényege, hogy az egyes fertőzött hosztok – a botmastertől kapott utasításra reagálva vagy akár automatikusan – saját alhálózatukban más, valamilyen sebezhetőséggel rendelkező aktív számítógépet keresnek. Sebezhetőség alatt az operációs rendszer vagy a futtatott programok biztonsági réseit, valamint a tűzfal nem megfelelő konfigurációját értjük. Amennyiben találnak ilyet, annak kihasználásával bejutnak, majd megfertőzik a gépet. Új áldozat keresésekor a fertőzött gépek sok kis adatcsomagot küldenek a hálózatra, ami az azt figyelő programok számára könnyen felismerhető, így ilyenkor a bot könnyen azonosítható.

A terjedés másik módja a felhasználó bevonásával történik. Ebben az esetben a botok valamilyen vírusos mellékletet vagy fertőzött weblapra mutató linket tartalmazó e-maileket vagy üzeneteket terjesztenek, melyet szándékosan úgy fogalmaznak meg az alkotók, hogy azzal felkeltsék a felhasználó figyelmét és rávegyék a melléklet vagy a link megnyitására. Amennyiben a jóhiszemű felhasználó ezt megteszi, a vírus futtatásra kerül, az áldozat gépe pedig a zombi hálózat részévé válik.

3.1.3 Támadási formák

A terjedés alapvető fontosságú a botnetek számára, ám tulajdonosaik célja a pénzszerzés, amit valamilyen jellegű támadás segítségével igyekeznek elérni. Az alábbiakban ezen támadási és pénzszerzési formák bemutatása következik.

Elosztott szolgáltatás-megtagadással járó támadás

A DDoS (Distributed Denial of Service), magyarul elosztott szolgáltatás-megtagadással járó támadás az egyik legrégebbi támadási forma, amivel a botnetek rendelkeznek. A több hosztról azonos időben induló DDoS támadás célja a célpont hálózati vagy számítási kapacitásának teljes lefoglalása, ezáltal lelassítva vagy megbénítva az áldozat által nyújtott szolgáltatást. Leggyakrabban TCP SYN, HTTP, és UDP elárasztást alkalmaznak, az összetettebb botok – például az Agobot [16] – többféle DDoS támadási formát is támogatnak. Ezt a támadási formát ritkán, szervezett formában szokták alkalmazni, sokszor valamilyen üzleti stratégia szerint. Jellemzően valamilyen versenytárs ellehetetlenítését célozzák meg ezek a támadások, amikre anyagi juttatás fejében felbérlik a botnetek gazdáit. 2008 telén például az AlertPay.com, egy online fizetéssel foglalkozó portál lett áldozata egy DDoS támadásnak [17], mely a karácsonyi vásárlás közepette több mint hét órán

keresztül akadályozta az ügyfeleket a szolgáltatás használatában, ezáltal hatalmas erkölcsi és anyagi kárt okozva a cégnek.

Spamküldés

Bizonyos botokat úgy programoztak, hogy azok elektronikus levelek küldésére és továbbítására is alkalmasak legyenek. Ez a funkció kettős előnyt jelent a botmasterek számára: a nagy számítási kapacitás és sávszélesség miatt az eddiginél sokkal nagyobb számú levelet terjeszthetnek, mivel azonban nem lehet visszakövetni a forrásig a levelek útját, így nem kell félniük a felelősségre vonástól sem.

A botnetek spamküldési képességének erejét jól mutatja a MessageLabs 2009 szeptemberi elemzése [18], mely alapján az összes levélszemét 87,9 százaléka, napi mintegy 151 milliárd kéretlen levél származik zombihálózatoktól. A legnagyobb spamküldő hálózat az 1,3-1,9 millió tagot számláló rustock [19], a legtöbb szemetet pedig a grum [20] hálózata küldi, napi 40 milliárdot. Ez az összes elküldött levélszemét 23 százaléka.

Az ily módon kiküldött levelek célja magának a botnetnek a terjesztése, illetve a felhasználókat interakciót igénybe vevő támadási formák közvetítése. A botok képesek lehetnek e-mail címek begyűjtésére is a fertőzött gépeken, ami által még sikeresebbek lehetnek a címzettek átverésében. A spamküldő botnetek leggyakoribb megbízói különféle reklámozó kémprogramok fejlesztői. Ezen programok a gépre települve folyamatosan saját hirdetésekre illetve honlapokra irányítják a felhasználót, valamint követik internetezési szokásaikat. Másik gyakori módszer, hogy nem reklámozási célra, hanem adathalász honlapok népszerűsítésére használják a botnetet. Ennek részleteit mutatjuk be a következő pontban.

Adathalászat (phishing)

A botnetek képesek nagy mennyiségű személyes vagy egyéb titkos adat megszerzésére. Általában jól ismert cégek – főleg bankok, pénzintézetek – nevében e-mail üzeneteket küldenek, melyekben azt kérik a felhasználótól, hogy lépjen be elektronikus úton fiókjába. A levél „kényelmi funkcióként” egy linket is tartalmaz, hogy az áldozat könnyebben eljuthasson a honlapra. A link azonban nem a cég weblapjára mutat, hanem egy ahhoz kísértetiesen hasonlító – esetleg kívülről nem is megkülönböztethető – ál-honlapra, mely többnyire a botnet valamely tagján fut. Ha a felhasználó bejelentkezik, felhasználói neve és jelszava az adathalász adatbázisába kerül. Ezek után bankok illetve hasonló, más pénzügyi tevékenységgel foglalkozó cégek esetén – ami az esetek döntő többségét jelenti – a felhasználói adatokat felhasználva a botnet gazdái saját vagy megbízójuk

részére utalják a megszerzett fiókban található összeget. Bár az UCSD tanulmánya szerint, melyben a Storm botnet egy részének viselkedését elemezték [21], a levélszemetekben található linkeket követve mindössze minden 12,5 milliomodik címzett adta meg adatait, a teljes rendszerre vetítve a botmastereknek ez így is napi hétezer dolláros (1,26 millió forintos) bevételt jelent. Az sem jelent megoldást a problémára, ha egy ilyen weblapról bebizonyosodik, hogy adathalászatra használják, és ezért letiltják a hozzáférést. A botnet más tagjai ugyanis a honlapot tükrözve pillanatok alatt átvehetik annak hosztingját.

Adathalászatra a fenti módszeren kívül használatos még a hálózat- illetve billentyűzet-megfigyelés, azaz a sniffing és a keylogging, melyekről a következőekben ejtünk szót.

Hálózat-megfigyelés (sniffing)

A botok felruházhatóak a hálózaton áthaladó adatcsomagok megfigyelésére alkalmas képességgel. A helyi hálózati interfészt lehallgató (promiscuous) módba állítva figyelik meg ilyenkor a hálózati forgalmat, felhasználói neveket és jelszavakat, illetve egyéb értékes információkat lopva. Ez a módszer manapság kevéssé jelentős veszélyforrás, mint régen, ugyanis az azonosításhoz szükséges adatok túlnyomó többségét titkosított formában küldik át a hálózaton.

Billentyűzet-megfigyelés (keylogging)

A keylogging nem más, mint a számítógép előtt ülő felhasználó által lenyomott billentyűk rögzítése és továbbítása a botmasternek. Nem véd ellene a titkosított csatornán történő adatátvitel, hiszen a lehallgatás még a titkosítás előtt megtörténik. A fejlettebb programok képesek bizonyos feltételek teljesülése esetén szűrni, például csak akkor rögzíteni a billentyűlenyomásokat, ha azok egy bizonyos honlap címének megadása után történnek, s így valószínűleg a felhasználói név illetve jelszó karaktereit jelentik. Ezzel a felhasználók bizonyos megcélzott honlapokra – tipikusan bankok portáljaira – vonatkozó hozzáférési adatait lehet ellopni.

Internetes reklámok és rangsorolás

A pénzszerzés manapság egyik igen egyszerű módja az online reklámokkal és a keresőoldalakon való minél előrébb való helyezéssel kapcsolatos. Felállítható például egy ál-honlap, aminek operátora pay-per-click, azaz kattintásonként fizető reklámszerződést köthet egy legális vállalkozással. Ezután több módszerrel növelheti a kattintásokat, s ezzel egy időben a bevételeit is: spam leveleket küldhet szét a honlap népszerűsítésére, a fertőzött gépeknek utasítást adhat, hogy lépjenek az oldalra és kattintsanak a reklámra, vagy akár a böngésző nyitólapját is beállíthatják oly

módon, hogy indításkor automatikusan klikkeljenek a reklámra. A botok képesek átirányításra is, azaz módosítani tudják a felhasználó által a böngészőbe írt url-t, hogy az áldozat a saját honlapjukra jusson.

Elosztott számítási feladatok

A botnetek tagjai együttesen hatalmas számítási kapacitással rendelkeznek, így képesek erős titkosítással védett és ezért egyébként nehezen megfejthető kódok feltörésére is. A kódtörésnek kétféle módja létezik: az online és az offline.

Előbbi történik akkor, mikor valaki „élesben", például egy webportál bejelentkező mezőjén keresztül úgy próbálja megfejteni a jelszót, hogy végigpróbálja a lehetséges eseteket. Ez ellen általában a bejelentkezési kísérletek maximalizálásával – és a határ túllépése esetén annak letiltásával – szoktak védekezni, így veszélyt csak a nagyon nyilvánvaló jelszavak használata esetén jelent a felhasználók számára. Az ilyen jellegű próbálkozások azonban nem reménytelenek: egy tízezer nyilvánosságra hozott Hotmail jelszó alapján készült felmérésből az derült ki, hogy az emberek 42 százaléka gyenge jelszót használ [22]. Egy olyan bot, amely az *123456* jelszóval próbálkozik, 64-szer járt volna sikerrel.

Az offline kódtörés az előbbi módszerrel szemben valamilyen elkapott, kódolt jelszó megfejtését jelenti. Ebben az esetben a próbálkozások számára nincs korlát: a kódtörő végignézi az összes lehetséges karaktersorozat kódolásának eredményét, és összehasonlítja az elfogott kóddal míg azok egyezni nem fognak. Míg egy mai erősnek mondható asztali számítógéppel több évre lenne szükség egy hét karakteres kód feltörésére, addig egy ezer tagból álló botnet 1 nap alatt végez a feladattal. A grafikus meghajtók általános számítási célokra történő felhasználásával ez az idő várhatóan tovább fog csökkenni a jövőben. A botnetek ezen felhasználási módjának éppen ezért hatalmas fejlődési lehetőségei vannak.

3.1.4 Command & Control módszerek

A botneteket alapvetően meghatározza, hogy milyen C&C csatornát használnak. A botok ezen keresztül kapják az utasításokat, és ezen múlik, hogy milyen gyorsan terjednek el a parancsok. A kommunikációs csatorna igen fontos a botnetek analizálása szempontjából, mivel egyrészt egy bot és variánsai legtöbbször azonos módszert használnak, másrészt ha sikerül lefülelni egy ilyen csatornát, hatékonyan fel lehet deríteni a teljes hálózatot, vagy akár meg is lehet bénítani a botnet irányítását.

A valóságban a C&C rendszerek többféle architektúrát alkalmaznak és többféle módon

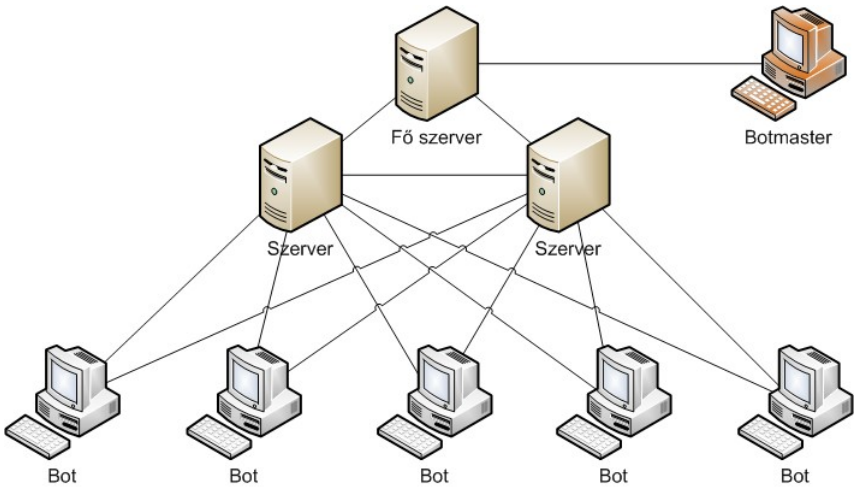
működnek. A leggyakoribbak a centralizált és a peer-to-peer elven működő rendszerek. Az alább bemutatásra kerülő módszerek összevetése az 1. táblázatban látható.

Típus	Tervezés	Detektálás	Késleltetés	Skálázhatóság	Túlélés	Vezérlő detektálás
Centralizált	Könnyű	Könnyű	Kicsi	Jó	Rossz	Könnyű
Láncolt	Közepes	Közepes	Nagy	Rossz	Közepes	Könnyű
Peer-to-peer	Nehéz	Nehéz	Közepes	Kiváló	Kiváló	Nehéz
Hibrid	Nehéz	Nehéz	Közepes	Kiváló	Kiváló	Nehéz
Véletlen	Könnyű	Nagyon nehéz	Nagy	Közepes	Kiváló	Nehéz

1. táblázat: C&C módszerek összehasonlítása

Centralizált C&C

A központosított architektúra volt az első, amit botnetek használtak. Több jól ismert bot – például az AgoBot [16], SDBot [13], Rbot [23] – is ebbe a családba tartozik.



1. ábra: Centralizált C&C csatornát használó botnet felépítése

A centralizált botnetek esetén a botmaster egy vagy több, megfelelő erőforrással rendelkező – többnyire fertőzött – gépet választ ki (1. ábra), amelyről majd irányítása alatt akarja tartani a hálózatot. Ezen – a hálózatban szerver feladatot ellátó – gépek IP-címét általában a botok kódjában helyezi el a hálózat létrehozója. A szerverek valamilyen IRC vagy HTTP szolgáltatást futtatnak, és mikor egy új bot kerül a hálózatba, az ehhez a szolgáltatáshoz csatlakozik. A megfelelő csatornához való csatlakozás után a bot a botmaster ugyanazon csatornán keresztül érkező utasítására vár.

Hogy ne lehessen lehallgatni a kommunikációt, a botnetek védelmi mechanizmusokkal lehetnek

ellátva. Tipikusan egy IRC botnet esetén ez a botok és a botmaster által használt csatorna jelszóval történő levédését jelenti.

A centralizált botnetek elterjedésének számos oka van. Először is a szoftveres eszközök (különféle IRC szkriptek) széleskörű elterjedtsége miatt könnyen és gyorsan lehet implementálni és személyre szabni a botokat, az elérhető minták segítségével olyanok is képesek lehetnek létrehozni botneteket, akik egyébként nincsenek birtokában ehhez szükséges programozói tudásnak. A botmaster ezzel a módszerrel közvetlenül irányíthat több ezer számítógépet, ráadásul mindegyik egyazon pillanatban értesül az utasításokról, így azonnal teljesíteni képesek, amire a hálózat gazdája parancsot ad, ezáltal könnyebb a hálózat menedzselése.

A központosítás bizonyos hátrányokkal is jár azonban. A szerverek ugyanis nemcsak legfontosabb, hanem egyben leggyengébb pontjai is a hálózatnak. Ha a botmaster nem tudja elérni őket, nem képes irányítani a botnetet. A botok szerver irányába történő kommunikációjának megfigyelése elegendő lehet annak megtalálásához, de elég egyetlen bot elfogása is, hiszen amennyiben sikerül visszafejteni a forráskódját, a benne található szerverlistából kiolvasható a szükséges adatok.

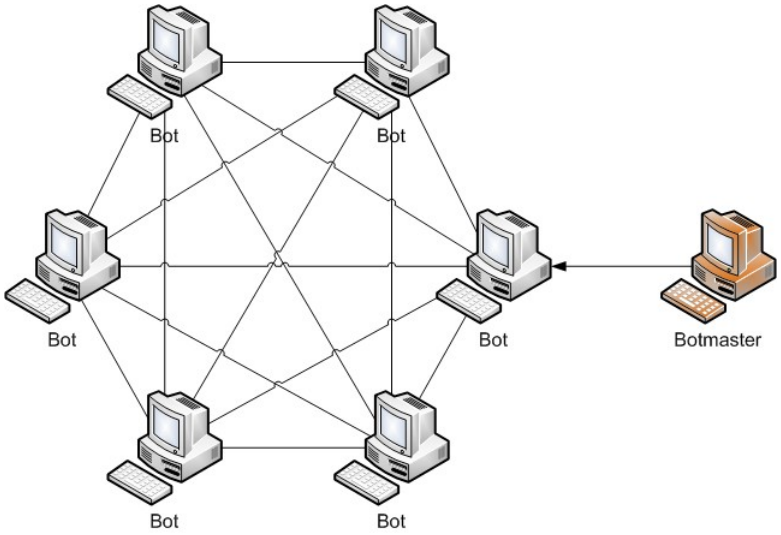
Láncolt C&C

A láncolt struktúrát használó botnetek esetén a botoknak létezik egy bizonyos sorrendje, és minden bot kizárólag az őt megelőző illetve követő társát ismeri és csak velük kommunikál. Bár az ilyen jellegű hálózatok kialakítása viszonylag könnyű, azonban egyetlen gép kiesése is a teljes hálózat működésképtelenné válását eredményezi. További hátránya, hogy az üzenetek terjedési ideje a botnet méretével lineárisan nő, így egy bizonyos méret felett célszerűtlen a használata. A gyakorlatban már ritkán használt módszer.

Peer-to-peer C&C

A központi szerverek lekapcsolása elleni védekezésként a botnetek alkotói újabb technológiák kifejlesztésébe kezdtek. Így alakult ki a P2P alapú kommunikáció a botok között. Ennek során a botok egy elosztott rendszert alkotnak, nincs kinevezett központ, aki a parancsokat továbbítaná (2. ábra), hanem a botok az általuk ismert társaiknak továbbítják a parancsokat peer-to-peer hálózatokból átvett megoldások alkalmazásával. A botmasternek elegendő egy tetszőlegesen kiválasztott zombigépnek kiadni az utasítást, egy bizonyos idő elteltével a hálózat minden tagja megkapja az üzenetet. Ez a módszer jóval nagyobb ellenállóképességgel ruházza fel a botnetet, hiszen az nem dől össze egy vagy több bot kiesése esetén sem, valamint nincs a botok által generált

forgalomnak jól kimutatható célpontja.

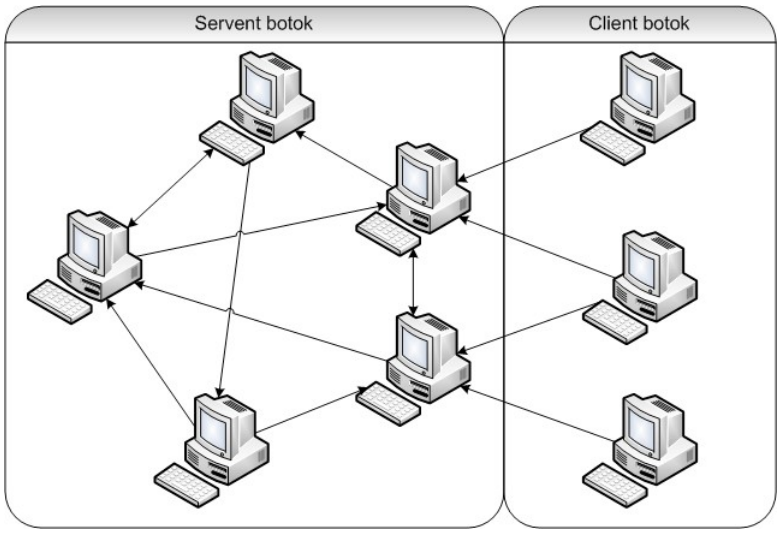


2. ábra: P2P C&C csatornát használó botnet felépítése

Az előbb említett okok miatt a P2P architektúrát a centralizáltnál jóval nehezebb detektálni és feltérképezni, ám ennek ára van. Először is a botok és a hálózat megtervezése bonyolultabb, gondos körültekintést igényel. Másodszor a jelenlegi P2P alapú kommunikációs technológiák tipikusan 10-50 résztvevő közötti kapcsolatot támogatnak. Ez a centralizált botnetekhez képest – ahol a botmaster jellemzően több ezer fertőzött számítógépet irányít – jelentősen kisebb. Harmadszor pedig a P2P technológia nem nyújt garanciát az üzenetek kézbesítésére és a terjedési késleltetésre. Éppen ezért peer-to-peer kommunikációs csatorna használata esetén nehezebb a botnet koordinálása, mint központosított esetben.

Hibrid C&C

Az amerikai UCF egyetem egy botnetekkel kapcsolatos tanulmányában bemutatásra került módszer a P2P architektúrára épülő zombihálózatok architektúrájára tesz fejlesztési javaslatot [24]. A botokat két csoportra osztják: a Client botok bizonyos időközönként a peer-listájukban szereplő Servent botokhoz csatlakoznak parancsok lekéréséhez (3. ábra). A Servent botok kizárólag statikus, és globálisan elérhető IP címmel rendelkező számítógépek lehetnek, egyszerre működnek kliensként és szerverként. Feladatuk, hogy a botmaster által kiadott *update* parancs esetén effektív módon lássák el peer-listával a hozzájuk rendelt botokat. A Servent botok titkosított csatornán várják a hozzájuk érkező kéréseket, ezáltal megnehezítve a hálózat felderítését. A botmaster bármelyik botnak kiadhatja parancsait: a bot továbbítja azt az összes, peer-listájában szereplő Servent botnak, akik továbbítják azt a saját listájuk tagjainak, és így tovább.



3. ábra: Hibrid C&C modell

Véletlen C&C

Ezt a módszert Evan Cooke mutatta be 2005-ben [25]. Bár jelenleg még nem tudni véletlen C&C rendszert használó botnetről, ám az ötlet igen vonzó lehet a magas túlélési valószínűséget megcélzó hálózatok alkotói számára. A módszer lényege, hogy a botok nem folytatnak aktív kommunikációt a botmasterrel, ehelyett beérkező kapcsolatra várnak tőle. Támadások indításakor a botmaster végigkutatja az Internetet a parancsára váró botok után kutatva, és ha talált egyet, kiadja neki az utasítást, majd tovább keres. A módszer egyszerűen implementálható és igen ellenálló a felderítéssel és felszámolási kísérletekkel szemben, viszont mivel nagy hálózat esetén hosszú ideig tart az összes tagot elérni, ezért nehezen skálázható, és nagy kiterjedésű, összehangolt támadásra nem alkalmas.

3.1.5 Gyülekezési módszerek

A gyülekezési mechanizmusok a botnetek terjeszkedéséhez szükségesek: új bot belépésekor annak meg kell találnia a szervert vagy a neki kiosztott gépeket. Erre háromféle módszer terjedt el.

Forráskódba írt („hard coded”) IP-cím

A legrégebbi és egyben leggyengébb módszer: a bot megalkotója a forráskódba írja be a szerver IP címét. Mikor a bot aktiválódik, a belé kódolt IP címen található számítógéphez csatlakozik. A módszer hátránya, hogy a központ egyszerűen azonosítható, valamint a kommunikációs csatorna is könnyen blokkolható. A szerver kiesése esetén a teljes botnet használhatatlanná válik, mivel a botok

nem képesek a beléjük kódolton kívül más helyre csatlakozni. Éppen ezért ezt a módszert mára csak elvétve alkalmazzák.

Dinamikus DNS domain-név

A mai botok nagy része forráskódjában domain-neveket tárol, melyek valamilyen dinamikus DNS szolgáltatóhoz vannak rendelve. A módszer előnye, hogy egy szerver letiltásakor a botnet megalkotója egy másik helyre költöztetheti azt, majd ezután elég a szolgáltatónál regisztrálnia az így bekövetkező IP-cím változást. A botok a szerver elérhetetlensége esetén nem válnak használhatatlanná: egy DNS kéréssel tájékozódnak az új címről. A botmaster szándékosan is váltogathatja a szerver helyzetét bizonyos időközönként, így nehezítve a botnet felderítését.

Elosztott DNS szolgáltatás

Hogy tovább nehezítsék a hatóságok dolgát, bizonyos botnetek saját elosztott DNS szolgáltatást is futtatnak, melyek kívül esnek a hivatalos szervek fennhatóságán. A botok ezen DNS szerverek címét is tartalmazzák, és őket célozzák meg, amennyiben a botnetet vezérlő szerverek nevének feloldására van szükségük. Sok esetben ezen DNS szolgáltatások magas számozású portokon érthetőek el hogy megkerüljék az internetes átjárók biztonsági védelmét.

Dinamikus domain-név

A fenti módszerek kiegészítésére a legújabb vírusok – köztük a Conficker is – pszeudo-véletlen generátor segítségével dinamikus módon változtatják a felkeresendő szerver domain nevét. Ennek előnye, hogy mire a szolgáltatói oldalon fény derül a botnetforgalom célpontjára, addigra az már másik domain-néven és IP címen működik. Ha ez mégsem sikerül időben, és a domaint lekapcsolják, rövid időn belül a generátor új címet sorsol, és a botnet újra működésképessé válik.

3.2 NetFlow

A NetFlow a Cisco Systems által 1996 óta fejlesztett rendszer, mely különféle információkat szolgáltat a hálózati forgalom viselkedéséről [26]. Hálózati forgalom analízis, DDoS támadások elleni védelem, valamint online szolgáltatások számlázása a technológia három fő alkalmazási területe. Használatához a rendszert támogató switchre vagy routerre van szükség. Az eszköz a rajta átmenő csomagok adatait képes eltárolni, majd ezeket flow-kba, azaz folyamokba gyűjteni. Egy folyamot öt adat határoz meg:

- A forrás IP címe és portja

•A cél IP címe és portja

•A használt harmadik rétegbeli protokoll

•A Type Of Service, azaz a szolgáltatás minőségét meghatározó byte

•A bemeneti logikai interface

Ezeken az adatokon kívül a router/switch további adatokat is számon tart. Ezek:

•A flowban küldött csomagok száma

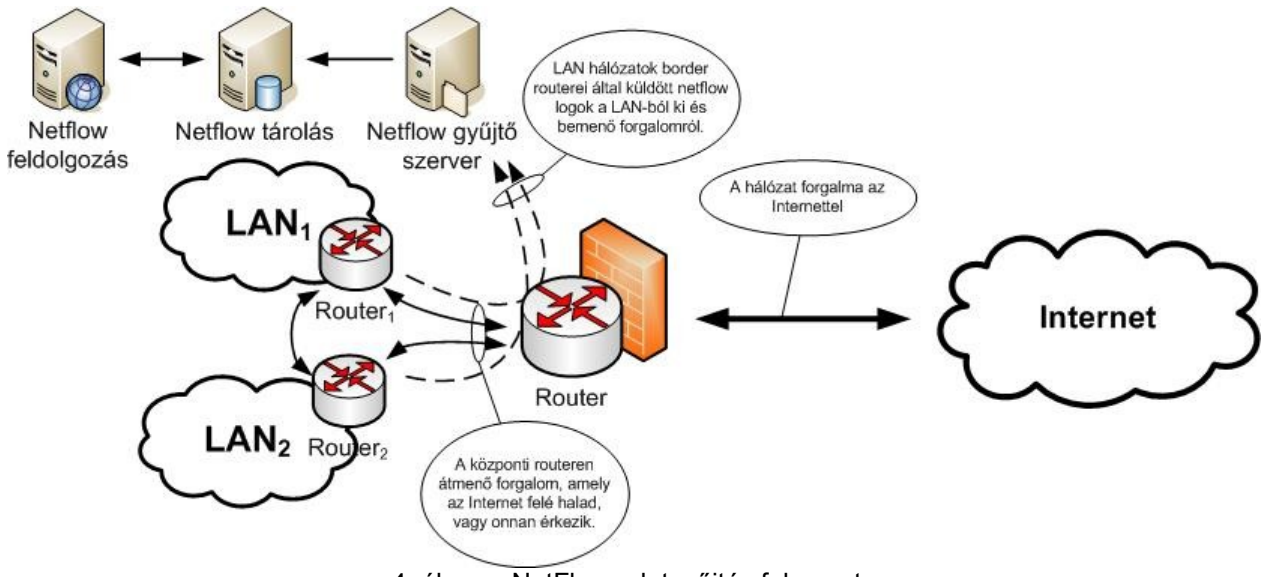
•A flow byte-ban mért mérete

•A flow kezdetének és végének időpontja

•A kimeneti logikai interface

•TCP flag-ek

•Routing információk (következő hop, forrás- és cél- IP maszk és AS azonosító)



4. ábra: a NetFlow adatgyűjtés folyamata

Ezen információkat az eszköz táblázat formájában egy beépített cache-ben tárolja. Ehhez kétféle módon lehet hozzáférni: egy Command Line Interface (CLI) segítségével, vagy egy úgynevezett NetFlow gyűjtő szerver igénybe vételével (4. ábra). Előbbit főleg hibajavítás esetén használják, mikor szükség van a hálózati forgalom azonnali megfigyelésére. A NetFlow gyűjtő feladata az exportált folyamok összerakása és értelmezése, valamint ezek összefűzése vagy aggregálása. A NetFlow exportálás periodikus időközönként történik a szerver felé, általában 30-50 flow-t tartalmazó UDP csomagok formájában. Egy flow a következő esetekben válik alkalmassá az exportálásra:

•Ha inaktívvá válik egy adott ideig (nem érkezik a flow-hoz tartozó új csomag)

•Ha a flow hosszú életű, és tovább tart a beépített „active” időzítőnél

•Ha a folyam lezárását jelző TCP flag (FIN, RST) érkezik

Az exportálás során az adat valamilyen előre meghatározott formátum alapján kerül feldolgozásra és továbbításra. A legelterjedtebb az Version 5, mely minden fent említett adatot továbbít, míg a szintén igen elterjedt Version 7 nem tartalmazza az autonóm rendszerekre, interfészekre, TPC flagekre és a TOS mezőre vonatkozó információkat. A NetFlow collector az adatok begyűjtése és a rajtuk végzett műveletek elvégzése után azokat valamilyen naplózó vagy feldolgozó programnak küldi tovább.

3.3 Peer-to-peer hálózatok és a DHT

A P2P hálózatok alapelve a decentralizáció: szemben a kliens-szerver architektúrával nincs központi elem, nincs hierarchia a résztvevők között, mindenki azonos szinten helyezkedik el. Az P2P hálózatok népszerűsége több okra vezethető vissza: olcsóbb a létrehozásuk és tervezésük mint egy nagy központi szerveré vagy szerverparké, a hálózatba kötött gépek számítási- és tárkapacitása mind kihasználható, valamint a rendszer elosztott jellege miatt jóval ellenállóbb a hibákkal és támadásokkal szemben. Az ilyen jellegű rendszerek legnagyobb kihívása adminisztrációs szempontból egymással semmilyen kapcsolatban nem lévő hálózatok között egy robusztus elosztott hálózat létrehozása. Hogyan lehet megtalálni egy adott adategységet egy hierarchiát és központi szervert nélkülöző hálózatban úgy, hogy a módszer skálázható is legyen? Erre a kérdésre dolgozták ki az elosztott hash tábla (DHT – Distributed Hash Table) módszerét [27].

A hash tábla *(érték, kulcs)* párokat tartalmaz, ahol a kulcs egy egyedi, a hozzá tartozó értékből egyirányú kódolással képzett karaktersorozat. A hash táblához bármelyik résztvevő fél hozzáférhet, és a kulcs ismeretében a hozzá tartozó értéket is ki tudja olvasni. Minden résztvevő néhány kiválasztott társáról („szomszédjáról”) információkat (például IP címet) tárol, ezáltal egy átfedő hálózatot alkotva, és ezen keresztül küldött üzenetek segítségével végzi a kulcsok tárolását és visszakeresését. A DHT implementálásához szükség van egy olyan algoritmusra, mely képes megállapítani, hogy a melyik tagja tartalmazza az adott *(érték, kulcs)* párost. A DHT egyetlen művelettel rendelkezik: a *keres(kulcs)* parancs meghívásakor a paraméterben kapott kulcsért felelős résztvevő azonosítóját (például IP-címét) kapjuk meg. Egy fájl tárolásához annak neve alapján a feltöltőnek csak elő kell állítani a kulcsot, majd a *keres* függvény meghívása után a kapott résztvevőnek kell küldenie azt. A fájl visszakeresése is hasonlóképpen működik: a névből kulcsot

állítunk elő, majd a *keres* függvény segítségével annak tárolási helyéről lekérjük az állományt. Természetesen a módszer nem csak fájlokra, hanem tetszőleges olyan adatra is működik, amelyet a módszerben implementált bemenetként képes feldolgozni.

3.4 Secure Overlay Service

Az *SOS (Secure Overlay Service – Biztonságos Átfedő Szolgáltatás)* egy elosztott tűzfalként működő átfedő hálózat, mely a résztvevők védelmét szolgálja DDoS támadások ellen. A rendszer két fő részből áll. Az egyik a védett hálózat peremén történő összetett szűrés, mely meggátolja az illetéktelen forgalom bejutását, és a támadást a gerinchálózat irányába szorítja vissza, ahol a nagy sebességű routerek már képesek kezelni a megnövekedett csomagmennyiséget. A másik egy többlépcsős hierarchia, mely a rendszer tagjait és a köztük folyó kommunikációt igyekszik elrejteni a támadó elől. Kívülről nem látható az egyes résztvevők rendeltetése, így a támadó kénytelen „vaktában” célpontot választani. A védett hálózat valamely tagja ellen történő támadás esetén annak feladatát egy másik tag veszi át, így a rendszer továbbra is működőképes marad. Amennyiben megszűnik a támadás, a kiesett tag visszakerül a rendszerbe, és újra képes feladata ellátására.

Az SOS egyik nagy előnye, hogy az általa védett hálózat elleni sikeres támadáshoz majdnem minden résztvevőt ki kell iktatni [28]. Ennek következménye, hogy az SOS hatékonysága függ az overlay csomópontjainak számától: minél többen használják, annál nehezebb támadást intézni ellene.

3.5 Klaszterezés

A klaszterezés során feladatunk egy adott adathalmaz diszjunkt osztályokra bontása, particionálása oly módon, hogy egy osztályon belül az adatok hasonlóak, míg különböző osztályokban a lehető legeltérőbbek legyenek. Mindehhez csak az adathalmaz elemei közti hasonlóságot használjuk. Ennek a hasonlóságnak a mérésére minden klaszterező eljárás egy, a halmazbeli elempárokon értelmezett $d(x_i, x_j)$ távolságfüggvényt használ. Ezzel megfogalmazva a probléma a következő: egy olyan osztályozását keressük a halmaznak mely minimalizálja az osztályokon belüli távolság összegeket. Az általánosság megszorítása nélkül ebben a dolgozatban mi az Euklideszi-távolságot használjuk:

$$d(x_i, x_j) = [\sum_{l=1}^{k} (x_{il} - x_{jl})]^{\frac{1}{2}} \quad .$$

Ezzel jelöléssel a feladatot most már pontosan megfogalmazhatjuk. Keressük azt az $S=\{S_{1,}S_{2,}....,S_m\}$ partíciót, amelyre

$$\sum_{i=1}^{m}\sum_{x_j\in S_i}[d(x_j,\mu_i)]^2$$

minimális, ahol μ_i az i-edik osztály középpontja, jelen esetben az i-edik osztálybeli elemek átlaga. Ezt szokták négyzetes hibának is nevezni.

A feladat megoldására több standard algoritmus ismert. Az alábbiakban a két legelterjedtebb algoritmus, a K-közép (K-means) [29], valamint annak egy javított változata, az X-közép (X-means) algoritmus [30] bemutatása következik.

3.5.1 K-közép algoritmus

A K-közép algoritmus használata esetén az adathalmazt fix, *K* darab osztályba particionáljuk (*K*-t előre rögzítjük) úgy, hogy a fenti négyzetes hiba minimális legyen. Inicializálásként választunk egy tetszőleges kiinduló partíciót, (például véletlenszerűen besoroljuk az elemeket a *K* klaszterbe) és kiszámoljuk a μ_i klaszterközepeket. Az algoritmus ezután iteratív lépéseket hajt végre. Minden körben besorolja az egyes elemeket abba a klaszterbe, amelynek a μ_i középpontja hozzá legközelebb esik, és az iteráció végén frissíti a μ_i értékeket. A folyamat akkor ér véget, amikor a klaszterek állandósultak. Az algoritmus futási ideje $O(lKN)$, ahol *l* az iterációk száma, *K* klaszterek száma míg *N* az adathalmaz mérete. A legtöbb alkalmazásban az algoritmus néhány iteráció alatt konvergál.

A K-közép algoritmusnak több előnye is van. Az egyik legfontosabb, hogy nagyon egyszerű, így könnyű implementálni. Ezen felül kicsi a számításigénye, valamint korábban is jó eredményeket értek el vele [31]. Ugyanakkor több hiányossága is van. A *K* klaszter számot nekünk kell megadni, továbbá érzékeny a kezdeti partíció megválasztására (adott kezdeti partíció mellet lehetséges, hogy az algoritmus egy lokális, de nem globális minimumhoz konvergál). A problémák részbeni kiküszöbölésére dolgoztál ki az X-közép algoritmust [30].

3.5.2 X-közép algoritmus

A K-közép algoritmushoz képest ezen algoritmus meghatároz egy *K** optimális klaszterszámot is, és erre ad vissza egy minimális négyzetes hibával rendelkező partíciót. Ha *K*-ra semmilyen megszorítást nem teszünk, és a feladat kitűzésén sem változtatunk, akkor az optimális megoldás az, amikor minden elem külön klasztert alkot, hiszen ekkor a négyzetes hiba a lehető legkisebb, vagyis

0. Hogy ezt elkerüljük, *K*-ra inputként megkövetelünk egy alsó és egy felső korlátot. Ez lényegesen nagyobb szabadságot jelent, mintha konkrét értéket adnánk *K*-nak, továbbá valós alkalmazásokban az ilyen alsó és felső határok természetesen adódnak. De ebben az esetben is igaz marad az, hogy nagyobb *K*-val kisebb négyzetes hiba érhető el. Mi viszont egy olyan partíciót keresünk, ami legjobban írja le a valóságot, és ebben nem feltétlenül a lehető legtöbb klaszter van. Ehhez bevezetjük az optimalitás egy új megfogalmazását. Az alapprobléma az, hogy partíciók egy családjából hogyan válasszuk ki a legjobbat. A leggyakoribb ilyen helyzetekben használt minősítési függvény a bayesi információ-kritérium függvény (BIC – Bayesian information criterion). Ezt részleteiben most nem tárgyaljuk, egy részletes leírás megtalálható [30]-ban. Ez alapján azt a partíciót fogjuk optimálisnak választani, amire a BIC függvény érték maximális.

Az X-közép algoritmus során két lépés ismétlődik. Az elsőben egy K-közép algoritmus fut egy megfelelően választott *K* értékkel, míg a másik részben azt vizsgáljuk, hogy kell-e, és ha igen, akkor hol kell új klasztereket létrehozni. Az algoritmus pontos működését egy 2 dimenziós példán mutatjuk be. Tegyük fel, hogy a *K*-ra megadott alsó korlát *K=3*. Az algoritmus lépései a következőek:

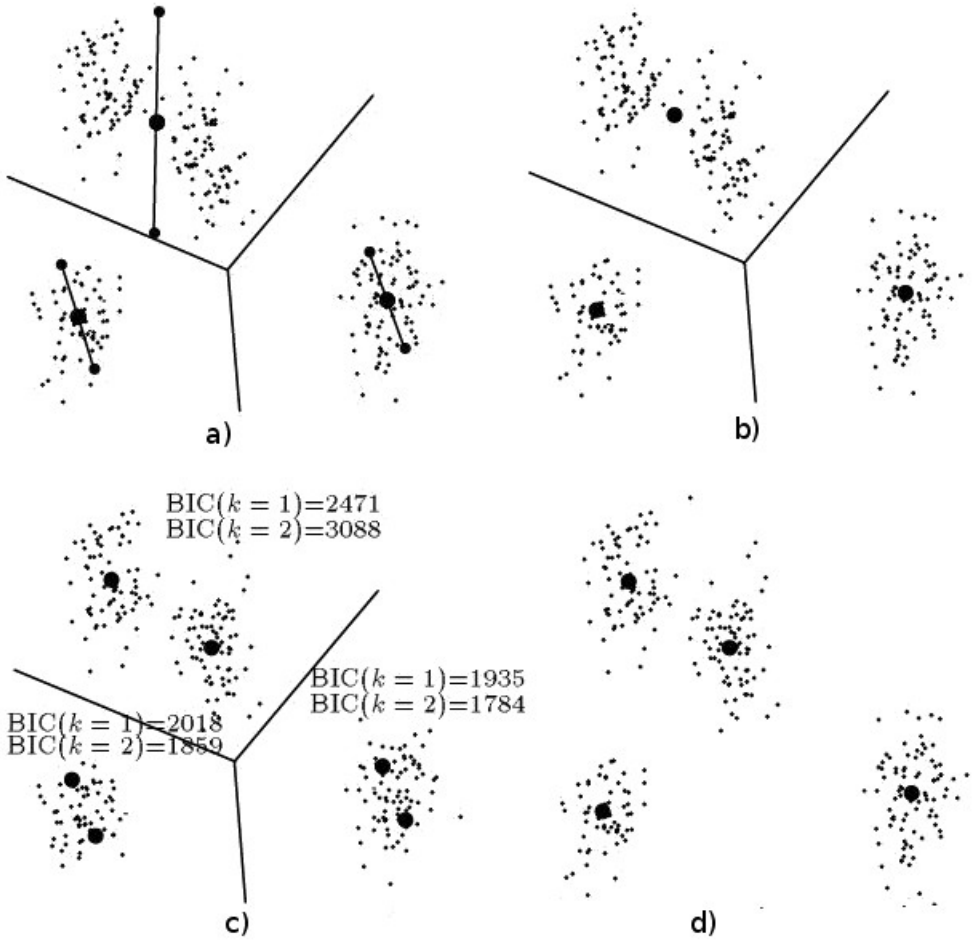
1)Végrehajtunk egy K-közép algoritmust *K=3*-al, és eltároljuk az így kapott partíciót. Ennek eredménye látható alább az 5/a. ábrán.

2)Minden klaszterközepet két gyermekre bontunk, és a klasztermérettel arányosan ellentétes irányban egy tetszőleges egyenes mentén elmozgatjuk őket, lásd 5/b.ábra.

3)Lokálisan minden klaszterben végrehajtunk egy 2-közép algoritmust az újonnan bevezetett klaszterközepekkel inicializálva, majd az egyes komponensekben kiszámoljuk az eredeti (vagyis az egy klaszterből álló) partíció és az újonnan kapott (vagyis a két klaszterből álló) partíció BIC értékét. Ennek eredményét az 5/c. ábra mutatja.

4)Attól függően, hogy melyik a nagyobb, vagy a szülőt hagyjuk el, vagy a gyerekeket. Az így kapott új partíciót az 5/d. ábra szemlélteti. Jelen esetben a végén 4 klasztert kaptunk. Ennek megfelelően vissza lépünk 1-hez, és *K=4*-el újrakezdjük.

Az algoritmus akkor ér véget, amikor *K* eléri a felső korlátot. (Nyilván abban az esetben is megállunk, amikor a *K* érték nem nőne.) Ekkor a különböző *K* értékekre a K-középpel kapott partíciók közül kiválasztjuk a legjobbat, vagyis azt, aminek a legnagyobb a BIC értéke. Ez a partíció lesz az X-közép algoritmus kimenete. Az algoritmus futási ideje megfelelő adattárolási módszerekkel jelentősen gyorsítható, lásd [30].



5. ábra: a) Az egyes klaszterközéppontok felbontása b) A K-közép algoritmus eredménye K=3-ra c) A lokális 2-közép algoritmusok eredményei, és a megfelelő BIC értékek d) Az X-közép első iterációjának eredménye

3.6 Outlier szűrés

Az outlier szűrés során feladatunk egy *x* mennyiség *n* elemű mintájából kiszűrni azokat, amik eltérnek a többitől. Két módszert vázolunk:

I. Első lépésben kiszámoljuk az $\overline{x}$ átlagot (mint empírikus várható értéket) és az σ^2 empírikus szórásnégyzetet:

1. $\overline{x}=\frac{\sum_{i=1}^{\infty} x_i}{n}$ és $\sigma^2=\frac{\sum_{i=1}^{\infty}(x_i-\overline{x})^2}{n}$

Második lépésben, ha a szórás relatíve nagy az átlaghoz képest, akkor az átlagtól

leginkább eltérő mintaelemet eldobjuk, és újra kezdjük a megmaradóakkal. Ha a szórás/átlag hányados megfelelően kicsi, akkor megállunk.

2. Ha $\frac{\sigma^2}{\overline{x}} < \varepsilon$, akkor STOP

Különben $j = argmax|x_i - \overline{x}|$, töröljük a *j*. flowt, és vissza 1.-hez

Megjegyezzük, hogy nagy csoportok esetében a szűrés gyorsítható, ha először a fentit csak $n_0 \ll n$ db flowra végezzük el, majd a többieknél mindig csak azt vizsgáljuk, hogy az adott elemet hozzávéve a $\sigma^2/\overline{x}$ hányados az ε küszöbindex alatt marad-e vagy sem, és ennek megfelelően tartjuk meg illetve dobjuk el az adott flowt.

II. Ennél a módszernél ugyancsak kiszámoljuk az $\overline{x}$ átlagot, valamint a *d* maximális eltérést:

$$1. \quad \overline{x} = \frac{\sum_{i=1}^{\infty} x_i}{n} \quad \text{és} \quad d = \max_{1 \le i \le n} |x_i - \overline{x}|$$

Második lépésben ugyanazt csináljuk, mint előbb, csak σ^2 helyett a *d* mennyiségre. Lényegében a mintaelemek átlaghoz viszonyított alsó és felső néhány százalékát dobjuk el.

Mint látható a két módszer nagyon hasonló, azonban a másodiknak kisebb a számításigénye.

4 A rendszer bemutatása

Az előző fejezetben az architektúránk elkészítéséhez szükséges ismeretanyagot mutattuk be. Ebben a fejezetben rátérünk az általunk tervezett rendszer bemutatására.

4.1 A rendszer megtervezése

Célunk egy olyan architektúra megalkotása, mely képes felismerni kártékony kódok forgalmát a vizsgált hálózatokban, mindezt robusztus és skálázható módon úgy, hogy az így szerzett anonimizált eseményeket az architektúra résztvevői képesek legyenek megosztani egymással.

A hálózatok forgalmának megfigyelésére a Cisco NetFlow rendszerét használjuk: a szolgáltatott információk tökéletesen alkalmasak a hálózat elemzésére. Nagyobb forgalom esetén azonban hatalmas méretű adatokkal kellene dolgozni, ezen adatokat először aggregáljuk, majd az aggregált forgalmi adatokat klaszterezzük. A klaszterekből a 4.4-es fejezetben ismertetett módszerrel tudjuk kiválasztani a botnethez tartozó forgalmat.

Mivel célunk egy elosztott és jól skálázható rendszer megalkotása, ezért egy P2P alapú rendszert terveztünk meg, melyben nincs fixen kijelölt központi szerver. A rendszer elosztott jellege miatt igen ellenálló a hibákkal és támadásokkal szemben. Egy szimplán P2P rendszerű hálózat azonban nem elegendő számunkra: a rendszert különböző feladatokat ellátó tagokból szeretnénk felépíteni, azonban ha a rendszer felhasználóin kívül mások számára is ismert lenne ezek elérhetősége, könnyen DDoS támadás áldozataivá válhatnának, ami a rendszert használhatatlanná, a védett hálózatokat védtelenné tenné. Az ilyen jellegű támadások kivédésével kapcsolatban több publikáció is készült [32][33][28], az általunk tervezett architektúra ezek közül a Secure Overlay Service (SOS) szolgáltatását használja.

A Secure Overlay Service bemutatásakor már szó esett egy számunkra igen előnyös tulajdonságáról: a rendszer hatékonysága arányos résztvevőinek számától – minél többen használják, annál nehezebb támadást intézni ellene. További előnye, hogy az útvonalválasztáshoz alkalmazott hash függvény elsődleges célján túl felhasználható bizonyos entitások kijelölésére és periodikus változtatására is. Ez lehetővé teszi a feladatok – és a velük járó terhelés – egyenletes szétosztását a rendszer résztvevői között. Mivel céljaink között igen fontos szerepet tölt be a globális felhasználhatóság támogatása és a stabilitás, valamint rendszerünkben az adatfeldolgozó szerepét a terhelés elosztása érdekében változtatni szeretnénk bizonyos időközönként, ezért a Secure Overlay System ideális választás számunkra.

Az egyes tagok közötti munkamegosztás, valamint a robosztusság további növelése érdekében a P2P hálózatot DHT szolgáltatás használatával egészítettük ki. Az elosztott hash tábla lényege – mint azt már a 3.3 fejezetben említettük –, hogy *(érték, kulcs)* párokat tartalmaz, és a rendszer minden résztvevőjének van egy kulcs tartománya, amiért ő a felelős. Amennyiben valaki kiesik a rendszerből, a rendszer többi résztvevője újraosztja annak feladatát. A kulcsérték meghatározásához a DHT egy egyenletesen szóró hash függvényt alkalmaz. Ennek a függvénynek a segítségével lehetőségünk van arra, hogy kijelöljük vele az éppen aktuális adatfeldolgozót. Ehhez venni kell egy számlálót, aminek értéke bizonyos megadott időközönként nő. Az új értékre kiszámoljuk a DHT hash függvényének segítségével annak kulcsát, majd megnézzük, melyik számítógép tartományába esik bele az így kapott kulcs. Egy kulcs érték pontosan egy számítógéphez tartozik – ez a rendszer lényege –, és ez a gép lesz az, ami átveszi az adatfeldolgozás feladatát. Mivel – mint azt említettük – a hash függvény egyenletes, így a feladat is egyenletesen fog eloszlani a számítógépek között.

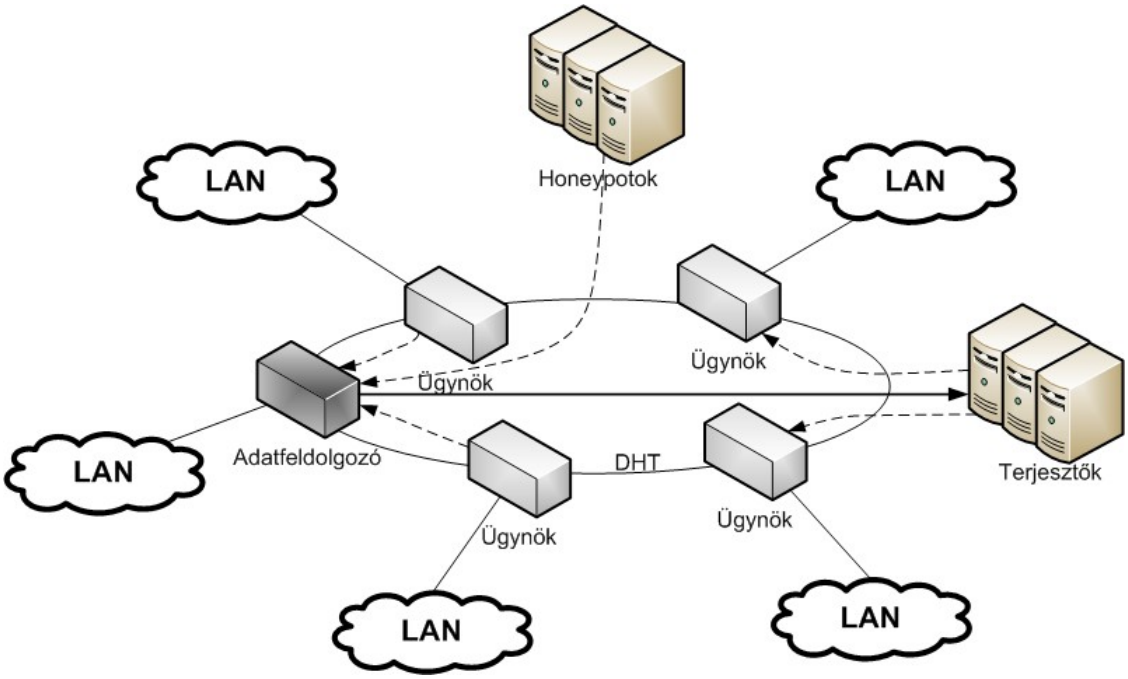
A rendszer feladatait több részre bontottuk:

- megfigyelés és a gyanús forgalom észlelése
- gyanús forgalom ellenőrzése
- kártékony forgalom adatainak tárolása és terjesztése

Ezen részfeladatokat a rendszer különböző tagjai között osztottuk el. Az ügynökök végzik a forgalom megfigyelését, anonimizálását, valamint amennyiben gyanús forgalmat észlelnek, annak adatait egy adatfeldolgozó entitásnak küldik, akinek feladata a gyanús forgalom ellenőrzése. Ehhez egy honeypot segítségét is igénybe veszi, aki az általa elfogott kártékony kódok forgalmi mintáinak lenyomatait küldi el igény szerint. Amennyiben sikerült azonosítani az illegális forgalmat, az arra jellemző adatokat egy terjesztésre kijelölt számítógépnek küldi el, aki szabályos időközönként értesíti az ügynököket az új információkról. Az egyes elemek részletesebb leírása a következő fejezetben történik meg.

Az adatok anonimitása a rendszer tervezésekor kiemelt szerepet kapott, hiszen a tagoknak többnyire nem érdeke saját forgalmi statisztikájuk nyilvánosságra hozatala: amennyiben egy cégről kiderül, hogy valamilyen károkozó található a hálózatában, valószínű, hogy presztízsveszteséggel kell, hogy számoljon, hiszen a befektetők illetve a cég szolgáltatását igénybe vevő emberek számára adataik biztonsága a legfontosabb. Éppen ezért a forgalmi adatokból már a feldolgozás első fázisában, az ügynököknél megtörténik az adatok anonimizálása.

4.2 A rendszer felépítése



6. ábra: a rendszer alapelemei

Rendszerünk négy alapelemből épül fel, ezek az ügynökök, a honeypotok, a terjesztők, valamint az adatfeldolgozó (6. ábra). A terjesztők kivételével a rendszer tagjait egy adatkereső és -szállító P2P hálózat köti össze. Ez biztosítja a skálázhatóságot és robosztusságot, ami elengedhetetlen feltétele a lehető legtöbb csomópont hálózathoz való csatlakoztatásának, s így egy globálisan elérhető elosztott detektáló rendszer kialakulásának.

4.2.1 A rendszer komponensei

Ügynökök

Az ügynökök céges vagy otthoni alhálózatba kötött számítógépek, melyek feladata a saját hálózatukat és az Internetet összekötő átjáróktól (gateway) kapott forgalmi információk begyűjtése és feldolgozása. Az ügynökök a rendszert átfogó P2P hálózathoz csatlakoznak, ezen keresztül képesek kommunikálni egymással. Az ügynökök a saját alhálózatuk irányából érkező gyanús forgalom észlelése esetén megkeresik az aktuálisan kijelölt adatfeldolgozót a P2P hálózat többi tagja között, és a forgalmi adatokat aggregálva és anonimizálva elküldik neki további elemzés céljából. Gyanúsnak akkor ítéli meg a forgalmat, ha az abból származó minták és a terjesztő entitásoktól periodikus időközönként kapott minták között egyezést talál.

Honeypotok

A honeypotok olyan számítógépek, melyeken direkt gyenge védelemmel ellátott operációs rendszer fut egy ellenőrzött virtuális környezetben. Saját aktív forgalmat nem generálnak a hálózaton. Ha mégis érzékelnek ilyet, akkor feltételezhető, hogy a számítógépet megfertőző vírus által jött létre, azaz rosszindulatú.

Rendszerünkben a honeypotok a gyanús forgalom megjelölésére szolgáló entitások. Amennyiben új típusú fenyegetést észlelnek, a károkozó forgalmáról lenyomatot készítenek, valamint megjelölik benne az általa használt C&C csatornát. Az így készített megjelölt és anonimizált lenyomatot az aktuális adatfeldolgozónak küldik tovább.

Terjesztők

A terjesztők a P2P hálózattól független részei a rendszernek. Feladatuk az adatfeldolgozótól érkező rendellenes minták begyűjtése, valamint azok rendszeres időközönként való megosztása minden, a hálózatban résztvevő ügynökökkel.

Adatfeldolgozó

A P2P hálózat egyik kijelölt tagja. A rendszerben egy időben egyetlen adatfeldolgozó számítógép van, azonban a feladat bizonyos időközönként másik tagra száll. Az adatfeldolgozó az ügynökök által veszélyesnek nyilvánított forgalomról készített tömörített, valamint a honeypotoktól lekért megjelölt adatokat gyűjti be. Feladata a küldők által már anonimizált adatokat klaszterezni és elemezni. Amennyiben az adatfeldolgozó kártékonynak ítél meg egy forgalmi mintát, továbbküldi a terjesztőknek, akik majd az így nyert információt az ügynökök rendelkezésére bocsátják.

4.3 Kiértékelés és mintakészítés

Az adatfeldolgozó feladata az ügynökök által beküldött aggregált adatok alapján egy, a botnet forgalom azonosítására szolgáló minta elkészítése. Ehhez először a teljes adathalmazt klaszterezzük az X-közép algoritmussal. Elvárásaink szerint a botnet kommunikációjához tartozó flow-k egy külön klasztert fognak alkotni. Ez alapján a klaszter alapján készítjük majd el a mintát. Több korábbi cikk [31],[34],[35] bemutatta, hogy a hálózati forgalom klaszterezése flow statisztikák segítségével alkalmas az azonos forgalomhoz tartozó flow-k csoportosítására.

4.3.1 Aggregálás

Valódi alkalmazásokban a Netflow által gyűjtött logok általában nagy méretűek, ezért nehéz velük

dolgozni. Hogy ezt a problémát kiküszöböljük, az ügynököknél egy előfeldolgozásnak vetjük alá ezeket a logokat. Most ennek az előfeldolgozásnak a mikéntjét mutatjuk be.

A logokban lévő flow-k azonosítására a forrás és cél IP-címeket, portokat illetve a megfelelő protokollt fogjuk használni. Ezeken kívül a 3.2. fejezetben szereplő statisztikák közül a következőket tekintjük:

- a flow-ban küldött csomagok száma
- a flow byte-ban mért mérete (az utóbbiakból könnyen számolható egy a flow-ra jellemző átlagos csomagméret)
- a flow kezdetének és végének időpontja (ezekből pedig számolható egy active time nevű időtartam ami ezek közt eltelt).
- Ennek megfelelően egy flow-t tekinthetünk egy 11 dimenziós vektornak.

Ha két flow-ra az IP-címek és a protokoll megegyeznek, tehát ugyanazok az IP-címek kommunikálnak ugyanazzal a protokollal, valamint legalább egyikük ugyanazon a porton keresztül, akkor ezeket a kapcsolatokat ugyanolyan típusúnak tekintjük, így nem szükséges őket külön kezelni. Ha egy A és egy B flow-ra $srcIP_A$=$destIP_B$, $destIP_A$=$srcIP_B$, $srcPort_A$=$destPort_B$ vagy $destPort_A$=$srcPort_B$[1], akkor ezeket a flow-kat is azonos típusúnak tekintjük. Ezeknél csak a kommunikáció iránya lesz más. A fel és le irányt tetszőlegesen megválaszthatjuk. Ennek a típusazonosításnak megfelelően a feldolgozás első lépésében átcsoportosítjuk a bejövő flow-kat, hogy az azonos típusúak egy csoportba kerüljenek. Célunk, hogy egy teljes csoportot egy flow-val reprezentáljunk.

A csoportosítás után minden csoporton belül kiszűrjük az outliereket, hogy egy rosszul besorolt flow ne tudja elrontani az egész csoport reprezentációját. Ezt a szűrést két dimenzióban végezzük el, az active time és az átlagos csomagméret mennyiségekre. Ezekre a mennyiségekre jellemző, hogy egy csoporton belül egy-egy flow nagyon eltér a többitől, ugyanakkor ezek a többi mennyiséget is reprezentálják valamilyen módon. A 3.6 fejezetben tárgyalt módszerek közül a kisebb számításigénye miatt a második módszert alkalmazzuk.

Az outlierek szűrését követően elkészítjük az egyes csoportok reprezentánsait. A csomagok számát valamint az active time-okat összeadjuk az csoportban szereplő flow-kra. A kezdeti időpontok közül kiválasztjuk a legkorábbit, a befejezési időpontok közül pedig a legkésőbbit, továbbá kiszámoljuk az alábbi származtatott mennyiségeket:

- az aggregált flow-k száma

1 src az angol source, vagyis forrás, dest az angol destination, vagyis cél szavak rövidítései

- átlagos csomagméret
- átlagos active time
- duration – a legkorábbi kezdeti időpont és a legkésőbbi befejezési időpont közt eltelt idő – ez fogja jelenteni az időtartamot, amíg a két IP-cím közti adott típusú kommunikáció aktív volt
- up/down+down/up – az adott csoportra tetszőlegesen megválasztjuk a fel és le irányokat, majd up jelöli a fel, down pedig a le irányú flow-kra a flow-k méretének összegét. Ez a mennyiség a két irány arányát méri. Ha ez kicsi (2 körüli), akkor a két irány nagyjából ugyanakkora forgalmat képvisel, ha pedig nagy, akkor jellemzően egyirányú forgalomról van szó.

Ez az 5 mennyiség könnyen számolható, és a továbbiakban ők reprezentálnak egy adott flow csoportot. A protokollok szerint az egészet szétválogatjuk, és mindegyikkel külön-külön foglalkozunk. Az IP címeket és portokat egyszerűen elhagyjuk, csak az ügynöknél lesznek feljegyezve, hogy adott esetben visszafejthető legyen a folyamat. Ezzel egyfajta anonimitást érünk el, a reprezentánsok nem tartalmaznak releváns információt az azonosítás szempontjából. Végső lépésként lenormáljuk az adatokat minden dimenzió mentén külön-külön 0 és 1 közé. Így mindegyik attribútum ugyanakkora súllyal jelenik meg a reprezentációban. A normáló faktorokat eltároljuk, mert a későbbiekben még szükségünk lesz rá. Az anonimizált és lenormált flow-kat az ügynökök küldik el az adatfeldolgozónak, ami ezeket kiértékeli.

4.3.2 Klaszterek azonosítása és mintakészítés

Amint az ügynökök elküldték a mintákat az adatfeldolgozónak, az klaszterezi azokat a botnet forgalom megtalálása érdekében. Az adatfeldolgozó ezt követően az X-közép algoritmust használja, hogy kiszámolja a K^* db klaszterközepet. Feladatunk a botnet kommunikáció klaszterének azonosítása. Ehhez a honeypotok által küldött adatokat használjuk. Alapvetően ők ugyanúgy és ugyanolyan formájú, azaz anonimizált adatokat küldenek az adatfeldolgozónak, mint a többi ügynök, azzal a különbséggel, hogy az ő általuk küldött adatok gyanús forgalomnak felelnek meg, és köztük meg vannak jelölve a C&C csatornának megfelelő flow-k.

Egy, a [36]-ben leírt valószínűségi modellhez hasonló módszert használunk. Jelölje p_i annak az eseménynek a valószínűségét, hogy az i-edik klaszter tartozik a botnet C&C kommunikációjához. Ezeket a p_i valószínűségeket az n_i/n maximum likelihood becsléssel becsüljük, ahol n_i a C&C kommunikációként megjelölt flow-k száma az i-edik klaszterben, n

pedig az összes jelölt flow száma. Ezek alapján a becsült értékek alapján az a klaszter tartozik legnagyobb valószínűséggel a C&C kommunikációhoz, ahol a legnagyobb ez a becsült valószínűség.

Miután azonosítottuk a megfelelő klasztert, a minta elkészítésére az egyik legkézenfekvőbb és talán legegyszerűbb lehetőség az, hogy tekintsük mintának egyszerűen a kiválasztott klaszter C centrumát, és tegyünk mellé egy d_0 kritikus távolságot. Ha egy flow távolsága C-től nagyobb, mint ez a d_0 érték, akkor ez a flow nagy valószínűséggel a botnet C&C-hez tartozik. Ezzel a kritikus távolsággal lehet finomhangolni a módszert. Ha d_0 nagy, akkor sok lesz az úgynevezett false positive eset – mikor kártékonyként azonosítunk valójában ártalmatlan forgalmat –, ugyanakkor nagy valószínűséggel a tényleges C&C csatornát is megtaláljuk. Ha d_0 -t csökkentjük, azzal csökkentjük a false positive esetek számát, de a megbízhatóság is csökken.

Amint elkészültek a minták, az adatfeldolgozó elküldi azokat a terjesztőnek, aki pedig szétosztja a felhasználók közt, és minden kezdődik elölről.

4.3.3 Több forrás esete

Most azzal az esettel foglalkozunk, amikor egyszerre több botnet van jelen a rendszerben. Ha ugyanolyan a C&C csatornájuk, akkor továbbra is egy darab kritikus klaszter jelenik meg. A módszer képes kezelni azt az esetet is, amikor a botnetek C&C csatornája különböző módon működik. Ilyenkor az alapvető feltevés értelmében, miszerint hasonló típusú forgalmakhoz tartozó flow-k hasonlóak a konfigurációs térben, ezek 2 vagy esetleg több külön-külön klasztert alkotnak majd, annak megfelelően, hogy mennyi botnet van jelen a rendszerben. A jelölt flow-k ilyenkor több felé oszlanak, ami azt eredményezi, hogy több klaszterre lesz nagy a p_i érték. Ez azt jelenti hogy több klasztert fogunk kiválasztani, és több mintát kell elkészíteni, minden kiválasztott klaszterhez egyet-egyet.

4.4 C&C felismerés

A C&C felismerés feladata a hálózatban két helyen történik meg: az ügynököknél, akik aggregált adataikat hasonlítják össze a terjesztőktől kapott mintákkal, valamint az adatfeldolgozónál, aki pedig az ügynököktől kapott gyanús minták és a honeypotokból kapott adatok között klaszterezéssel keresi meg az illegitim forgalmat és a hozzá kapcsolódó C&C csatornát.

A konkrét problémát úgy lehet megfogalmazni, hogy adott minták egy halmaza,

$$(C_1, d_1), \ldots, (C_M, d_M)$$

valamint egy *x* flow, amiről szeretnénk eldönteni, hogy botnet C&C-hez tartozik-e vagy sem. Ehhez kiszámoljuk minden mintára a $\|C_i - x\|$ távolságot, és kiválasztunk egy olyan *i* indexet,, amire ez kisebb, mint d_i , ha van egyáltalán ilyen. Ha nincs ilyen, akkor nagy valószínűséggel ez a flow nem botnet C&C-hez tartozik. Ha pedig találunk ilyet, akkor az a flow nagy valószínűséggel egy megfelelő botnet C&C-jéhez tartozik, mégpedig abba, ahol a $\|C_i - x\|$ távolság a legkisebb. Ezt a hipotézist arra alapozzuk, hogy ha ezekkel a klaszter közepekkel végrehajtanánk egy K-közép algoritmus egy olyan adathalmazra, amiben *x* is benne van, akkor a soron következő iterációs lépésben *x* épp a kiválasztott klaszterbe kerülne. Ha az egyes minták ugyanabból az eljárásból származnak, és a kritikus távolságokat érdemes a következőképp megválasztani:

Legyen az a maximális d_0 érték, amire még igaz, hogy az X-közép outputjában szereplő klaszter közepek köré ilyen sugarú gömböket írva, azok mind diszjunktak, vagyis

$$d_0 = \frac{1}{2} \min_{i \neq j} \|C_i - C_j\|$$

ahol C_i az i-edik klaszter középpontját jelöli.

Ha a kritikus távolságokat így választjuk, akkor ez azt eredményezi, hogy ha találunk megfelelő *i* indexet, akkor az egyértelmű.

5 A rendszer tesztelése

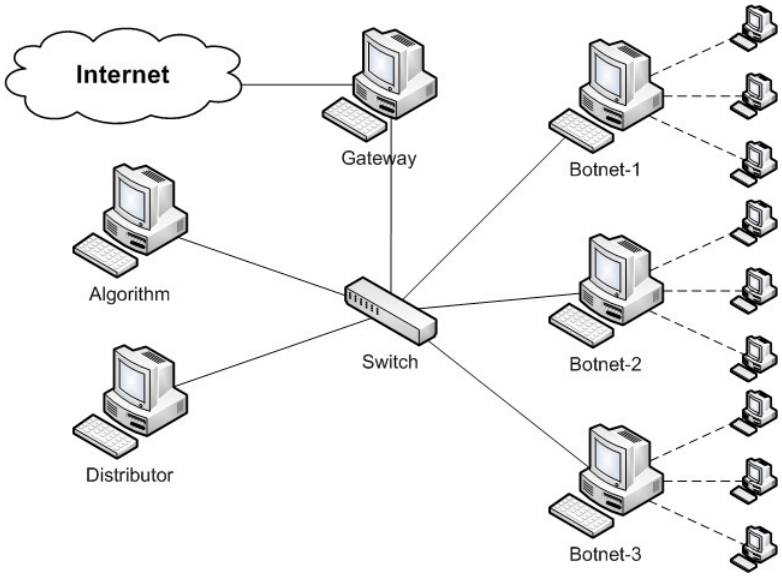
Tervezett architektúránk tesztelésére egy mérési környezetet alakítottunk ki, mellyel képesek lehetünk rendszerünk egyes elemeinek működését vizsgálni. A teszthálózat célja, hogy az éles rendszerhez hasonló módon működjön, azonban a külvilágtól el legyen zárva. Ez azért fontos, hogy a kártékony kódok forgalmának azonosításához használt algoritmus ellenőrzésekor felhasznált vírusok ne juthassanak ki a teszthálózaton kívülre.

Tesztrendszerünk biztonsága és lehető legkönnyebb konfigurálhatósága érdekében a felhasznált számítógépekre a Debian operációs rendszer stabil változatát telepítettük (Debian Lenny). Mindegyik számítógépen ugyanazokkal a lépésekkel, parancssorból lett telepítve a rendszer. Célunk ezzel az volt, hogy a telepített csomagok mennyiségének minimalizálásával a gépek stabilitását és biztonságát növeljük.

Elsődleges célunk a tesztrendszerrel az általunk megalkotott algoritmus tesztelése volt. A rendszer jelen dolgozatban nem részletezett részei – a peer-to-peer, DHT és az SOS szolgáltatások –

Python nyelven implementálva mind rendelkezésünkre állnak, így a későbbiek során a két modul – a forgalomanalízis és az architekturális szolgáltatások – egyesítése már nem okozhat gondot. Ahhoz azonban, hogy a teljes architektúra működőképes legyen, fontos hogy belássuk a detektálás hatékonyságát és megbízhatóságát.

5.1 A teszthálózat felépítése



7. ábra: a teszthálózat felépítése

Teszthálózatunk hat számítógépből épül fel (7. ábra). Ezek mind egy Cisco switch-en keresztül kapcsolódnak egymáshoz. .

5.1.1 Virtuális rendszert futtató számítógépek

A rendszerben található ügynökök és megfigyelt alhálózatok szimulációja céljából teszthálózatunkban három ilyen típusú számítógépet állítottunk fel (Botnet-1, Botnet-2 és Botnet-3 névvel). Ezen számítógépekre a VMware Workstation [37]virtualizációs program 6.5.2-es verziója segítségével 3-3 virtuális gépre Windows XP operációs rendszert telepítettünk. A virtuális rendszereket úgy állítottuk be, hogy megfelelő célpontot jelentsenek a vírusok számára: kikapcsoltuk a beépített tűzfalakat valamint az automatikus frissítés lehetőségét. A virtuális gépeket az azonosítás megkönnyítése érdekében a Botnet-*X*-*Y* névvel láttuk el, ahol az *X* a számítógép sorszáma, az *Y* pedig az azon futó virtuális gép sorszáma.

Az egy számítógépen futó három virtuális gép egymással van egy hálózatba kötve, valamint egy

a VMware Workstation által biztosított virtuális interfészen keresztül csatlakozik a tesztrendszer hálózatához, pontosabban az ott elhelyezett Cisco switch-hez.

A virtuális számítógépek célja az egyes alhálózatokban való adatforgalom generálása. Ezen forgalom mind áthalad a Gateway számítógépen, ami rögzíti a rajta – a belső hálózat vagy a külvilág irányában – átmenő forgalom adatait. A külvilág irányába kizárólag ICMP üzeneteket engedélyeztünk pingeléshez – a hálózat ellenőrzése céljából –, valamint http és DNS lekéréseket. Ezen beállítások elegendőek ahhoz, hogy a számítógépekkel képesek legyünk szimulálni egy átlagos felhasználó által generált Internetes forgalmat, viszont kellőképpen szigorúak, hogy az általunk vizsgált vírusok ne juthassanak ki a teszthálózatból. A maximális biztonság érdekében minden általunk lefuttatott vírus forráskódját leellenőriztük terjedési mechanizmusuk megismeréséhez és az általuk használt portok feltérképezéséhez.

A vírusok forráskódját a Botnet-1-1 virtuális gépen Visual Studio 6 alatt ellenőriztük és fordítottuk.

Mindegyik virtuális gépre telepítettük a mIRC 6.35 IRC klienst az IRC botok teszteléséhez. Ezek alapértelmezett módon a Gateway-re telepített szerverhez csatlakoznak a #Botnet szobába, felhasználói nevük a gép nevéhez hasonlatos. Azzal, hogy mindegyik gépről bejelentkezhetünk, lehetővé tettük hogy bármilyen kiindulási pontból bármely célpontnak parancsokat lehessen adni majd IRC botok tesztelésekor.

5.1.2 Gateway

A teszthálózat összes számítógépének alapértelmezett átjárója. Rajta keresztül történik a hálózati elemek (a Botnet, a Distributor és a Algorithm számítógépek) kommunikációja. A Gateway egy több feladatot ellátó entitás. Egyfelől tűzfal funkciókat lát el: a külvilág felől befele, illetve a belső hálózat felől kifele irányuló forgalmat szűri meg – az iptables segítségével –, hogy se illetéktelen felhasználók ne legyenek képesek hozzáférni a hálózat tagjaihoz, se a vírusok ne tudjanak kijutni a világhálóra. A belső hálózatban történő forgalmat nem korlátoztuk.

A Gateway a csomagszűrésen kívül a hálózati forgalom megfigyelésében vesz részt, melyhez több programot is telepítettünk a számítógépre. Ezek elindításának megkönnyítése és automatizálhatósága érdekében készítettünk egy shell szkriptet, mely elvégzi azok paraméterezett indítását és leállítását. A script paraméter nélküli futtatás esetén a 8. ábrán látható információkkal látja el a felhasználót.

```
botnet@Botnet-Gateway:~$ ./botnet.sh

Usage: botnet {start|stop} {all|unreal|fprobe|fcapture|logging}
First parameter sets whether to start or stop a program
Second parameter selects the program to start/stop
YOU NEED ROOT ACCESS TO USE THIS SCRIPT
Available programs:
   unreal   = Unrealircd IRC server, required for IRC bot testing
   fprobe   = fprobe, required for netflow collecting
   fcapture = flow-capture, required for flow logging
   logging  = fprobe+fcapture
```

8. ábra: a botnet.sh script paraméter nélküli indításakor megjelenített információk

A forgalmi adatokat az fprobe [38] program segítségével gyűjtjük össze, ami továbbítja azokat a megadott collector felé. Az általunk készített shell szkript az *-i eth0 -n 7 localhost:555* paraméterekkel hívja meg a programot: a belső hálózat felé irányuló interfészt figyeli meg, NetFlow Version 7 formátumban exportálja az adatokat a localhost 555-ös portjára. A végleges napló fájl a flow-tools [39] programcsomagban található flow-capture programmal készül el. A programot a szkript a *-V 7 -n 288 -w /var/log/flow/ -N -1 -E 8G 0/0/555* paraméterezéssel indítja el: NetFlow Version 7 formátumban várja az adatokat az 555-ös porton, melyeket 5 perces felbontásban a /var/log/flow/ könyvtáron belül az aktuális mérés dátuma szerinti alkönyvtárba ment el. Az így készített naplófájlokat használja fel algoritmusunk a forgalom elemzésére.

A Gateway gépre az említett programokon kívül feltelepítettük az UnrealIRCD [40] nyílt forráskódú IRC szervert az IRC botok vizsgálatához. A tesztelés során a vizsgált botok a szerver #Botnet szobájához csatlakoznak automatikusan, és ott várják a parancsokat.

5.1.3 Algoritmus futtató

Ennek a számítógépnek a feladata az algoritmus futtatása. Ezen a gépen végeztük a Gateway által gyűjtött forgalmi minták elemzését az erre készített programunkkal, mely a

5.1.4 Terjesztő

A terjesztő számítógépet (Distributor) a tesztelés jelenlegi fázisában nem használtuk. Feladata az adatfeldolgozó számítógéptől kapott adatok fogadása és szétosztása lenne.

5.2 A tesztelés folyamata

Az algoritmus teszteléséhez elsőként működő víruskódokat gyűjtöttünk. A 64 darab Internetről

letöltött vírus forráskódjaiból célunk azok kiválasztása volt, melyek Windows XP rendszer alatt futnak, valamilyen jellegű hálózati forgalmat generálnak, és melyeket C vagy C++ nyelven írtak. A 64 kódból 25 felelt meg ennek a három feltételnek. A 25 kódból 3 volt, amit sikeresen le tudtunk fordítani. Ezek az Agobot, a Beatrix, illetve a Sasser, melyek részletes bemutatása alább következik:

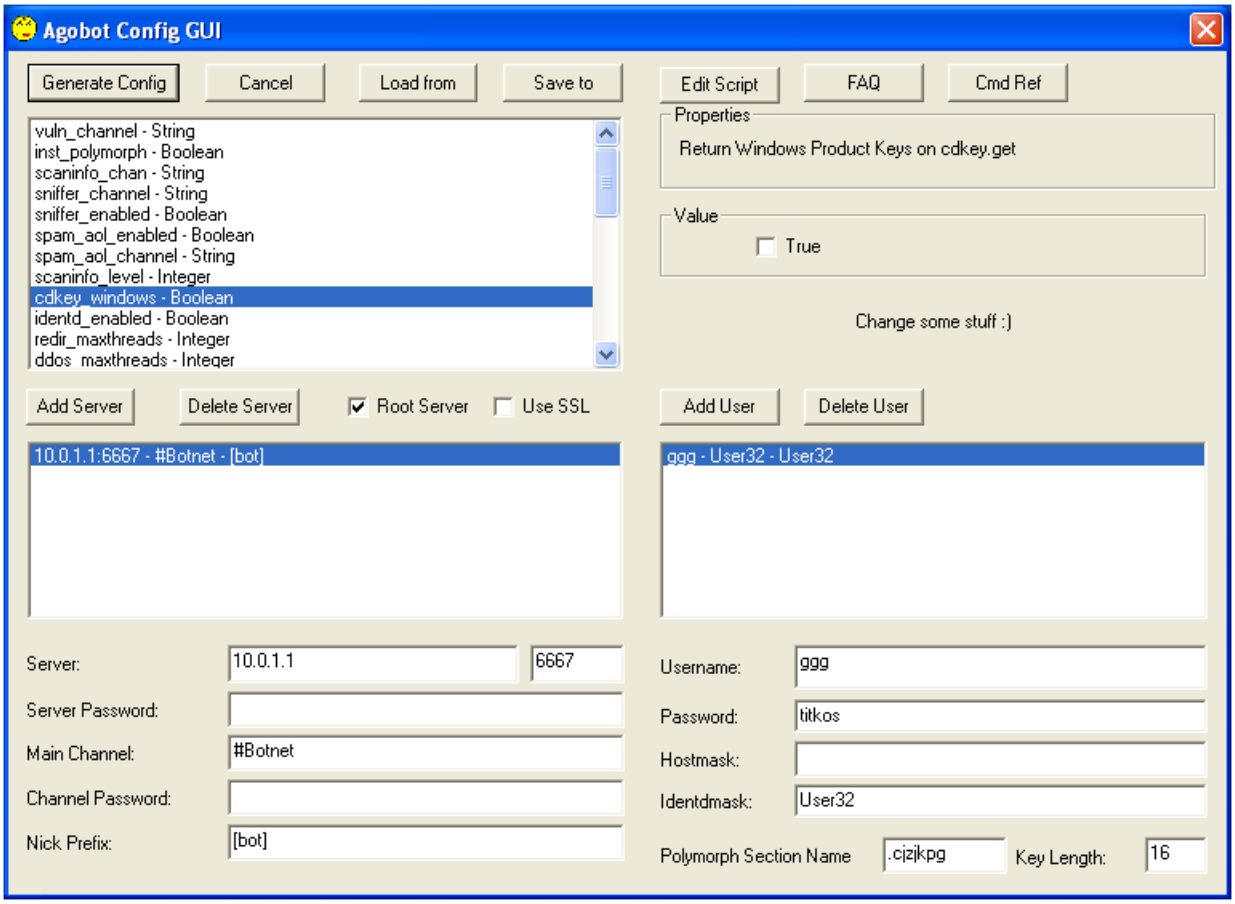
5.2.1 A tesztelt vírusok

Agobot

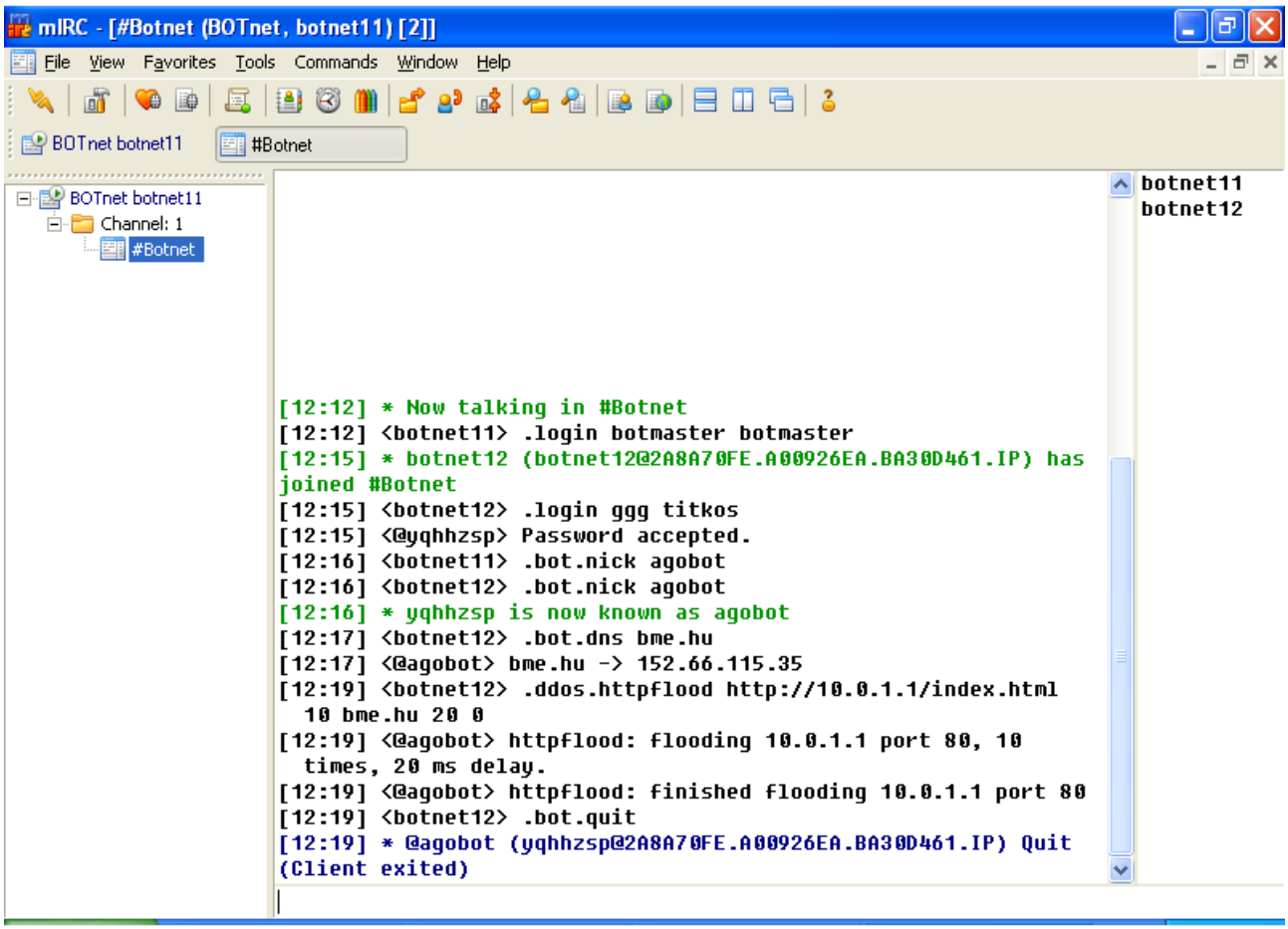
Az Agobot – mely Gaobot, Phatbot és Polybot néven is ismert – egy moduláris IRC bot és egyben a belőle kialakult víruscsalád neve is. Első verzióját Axel Gemble írta 2002-ben. Az Agobot egy több szálon futó, objektum-orientált vírus, melyet C++ nyelven írtak, némi assembly kóddal kiegészítve. Készítője GNU GPL licensz alatt tette közzé a vírust, ami azt jelenti, hogy az bármilyen jellegű módosítás esetén is a kód szabadon elérhető kell, hogy legyen [41]. A modularizált felépítés hatására a vírusba épített modulok könnyedén átültethetőek egy másik, ugyanúgy Agobot alapú vírusba, akár komolyabb programozói tudás nélkül, ráadásul a forráskód jól kommentelt, így könnyen megérthető a vírus működése. A modularitás másik előnye a víruskészítők számára, hogy igen gyorsan ültethetőek a kódba új fajta támadási módszerek vagy védekezési eljárások. Ennek hatására a vírusnak jelenleg több ezer variánsa létezik.

Az Agobotot a Windows operációs rendszereken való működésre tervezték. Forráskódjának lefordításához Visual Studio valamint a hozzá tartozó SDK (Software Developement Kit – Szoftverfejlesztési Eszköz) és Processor Pack szükséges. Készítője mellékelt egy segédprogramot is a vírushoz, melynek segítségével személyre lehet szabni annak beállításait (9. ábra). Ily módon megadható például hogy milyen szerverhez csatlakozzon a vírus, milyen botmaster nevet és jelszót fogadjon el, vagy hogy elinduljon-e automatikusan a fertőzött számítógépek újraindulását követően.

Az Agobot többféle módon terjedhet. Eleinte fájlmegosztó hálózatokon és megosztott hálózati meghajtókon terjedt, későbbi variánsai már féregszerű módon a Windows operációs rendszerekben jelen lévő távoli buffer túlcsordulási sebezhetőségek kihasználásával fertőztek [42]. A vírus a fertőzés után a rendszerben automatikus folyamatként indul, és csatlakozik egy előre megadott IRC szerverre. Ez esetünkben saját UnrealIRCD szerverünket jelenti. A szintén előzetesen megadott csatornához csatlakozva vár a botmaster utasításaira.



9. ábra: az Agobothoz mellékelt konfigurációs alkalmazás felhasználói felülete



10. ábra: az Agobot használata

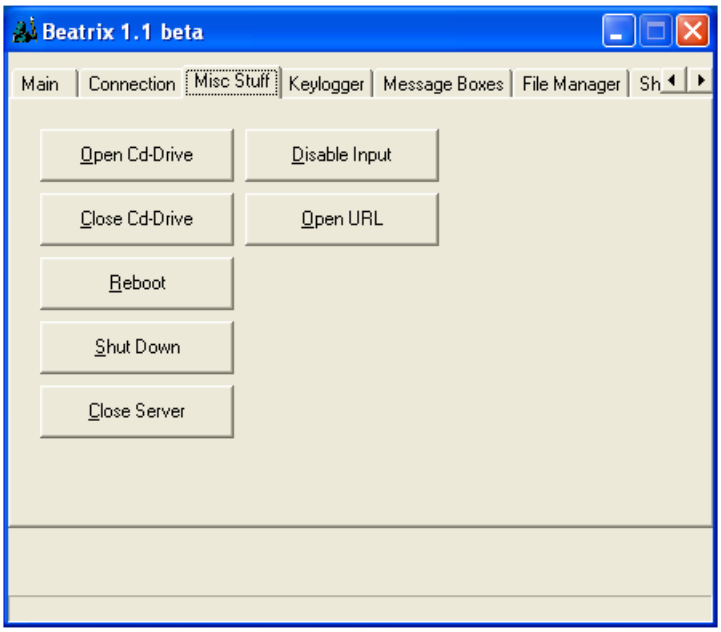
A botmasternek a csatornához csatlakozva azonosítania kell magát egy *.login név jelszó* paranccsal. Ezt a csatornán lévő botok összevetik a saját futtatási paramétereikben található névvel és jelszóval. Amennyiben megegyeznek, a felhasználó hitelesítette magát, s ezután kiadhatja utasításait a botok számára. A 10. ábrán látható, amint botnet11 felhasználó rossz paraméterekkel próbál meg bejelentkezni, így a bot nem reagál rá sem ekkor, sem mikor hitelesítés nélkül próbál parancsot adni. Botnet12 felhasználó azonban sikeresen bejelentkezik, és parancsaira a bot minden esetben reagál. Az ábrán látható parancsok sorrendben: névváltoztatás, domain név feloldás DNS segítségével, egy http flood, valamint a bot kikapcsolása. Az alább látható 2. táblázat tartalmazza az Agobot által elfogadott parancsok teljes listáját.

Parancs	Hatás
bot.about	Alapinformációk kiíratása a botról.
bot.die	Leállítja a botot.
bot.dns <hostname/ip>	IP vagy hosztnév feloldása DNS segítségével.
bot.execute <visibility> "<command>"	A bot (a *visibility* paraméter 0 értéke esetén rejtve) végrehajtja a *command* utasítást
bot.id	A bot azonosítóját adja vissza, melyet újabb verzió frissítésénél használ
bot.nick <nickname>	A bot IRC-n használt nevét változtatja meg a paraméterben kapott értékre.
bot.open <filename>	A paraméterben megadott fájlt nyitja meg a botot futtató számítógépen.
bot.remove	Törli a botot a számítógépről.
bot.removeallbut <id>	A megadott *id*-jű botokon kívül törli az összeset az azokat futtató számítógépekről.
bot.rndnick	Új, véletlenszerű IRC-n használt névvel látja el a botot.
bot.status	A bot állapotát jeleníti meg (például: készelét, futási idő, stb).
bot.sysinfo	A botot futtató rendszerről ad információt.
bot.quit	A parancs hatására a bot kilép az IRC szerverről és leáll.
bot.repeat <num> <cmd>	A bot a *cmd* parancsot hajtja végre *num* alkalommal.
commands.list	Felsorolja a botnak adható parancsokat
cvar.list	Felsorolja a bot futtatási paramétereit.
cvar.get <cvarname>	Kiírja a bot megadott paraméterének értékét.
cvar.get <cvarname> "<value>"	Beállítja a bot megadott paraméterét a megadott értékre.
irc.disconnect / irc.reconnect	A bot kijelentkezik / bejelentkezik az IRC csatornára
irc.action <target> "<action>"	A bot a *target* paraméterben kapott célpontnak egy *action* IRC parancsot küld végrehajtásra.
irc.getedu	Ha a hosztnév *.edu* sztringet tartalmaz, végrehajtja rá az *irc.netinfo* parancsot
irc.gethost <hostpart>	Ha a hosztnév a paraméterben megadott sztringet tartalmaz, végrehajtja rá az *irc.netinfo* parancsot
irc.join <channel> <pwd> / irc.part <channel>	A bot csatlakozik / elhagyja a megadott csatornát
irc.netinfo	Hálózati információkat irat ki a bottal.
irc.privmsg <target> "<text>"	A bot privát üzenetet küld a célpontnak.
irc.quit	A parancs hatására a bot kilép az IRC-ről.
irc.raw "<string>"	A bot raw sztringet küld a szervernek.
irc.reconnect	A bot újracsatlakozik a szerverhez.
irc.server <server> <port> <serverpass>	A bot futtatási paramétereiben szereplő IRC szerver beállításait módosítja.
irc.mode <modestr>	A bot IRC módot vált (például operátor rangot kap, stb).
login <user> <password>	A botmaster bejelentkezéséhez használt parancs.
mac.logout	A botmaster kijelentkezéséhez használt parancs.
redirect.tcp <localport> <remotehost> <remoteport>	Az adott TCP portot átirányítja a megadott hoszt és port irányába
redirect.gre <server> <client> [localip]	Átirányítja a GRE forgalmat *server* felől *client* irányába *localip*-n keresztül.
redirect.stop	Leállítja az összes átirányítást.
ftp.download <user> <pass> <host> <path> <target>	Fájl letöltése megadott FTP hosztról a számítógépre.
ftp.execute <user> <pass> <host> <path> <target>	Fájl letöltése megadott FTP hosztról a számítógépre, majd annak elindítása.
ftp.update <user> <pass> <host> <path> <target> <id>	Fájl letöltése megadott FTP hosztról a számítógépre, majd ha az id nem egyezik, a bot frissítése a letöltött fájl futtatásával.
http.download <host> <path> <target>	Fájl letöltése megadott http hosztról a számítógépre.
http.execute <host> <path> <target>	Fájl letöltése megadott http hosztról a számítógépre, majd annak elindítása.
http.update <host> <path> <target> <id>	Fájl letöltése megadott http hosztról a számítógépre, majd ha az id nem egyezik, a bot frissítése a letöltött fájl futtatásával.

2. táblázat: az Agobotnak adható parancsok listája

Beatrix

A Beatrix Windows operációs rendszereken működő távoli elérést biztosító alkalmazás. Kliens-szerver alapon működik: a számítógépen futó szerver az 1000-es porton várja a kliens csatlakozását. A kliens a szerverhez való csatlakozás után egy kezelőfelületen keresztül (11. ábra) az azt futtató számítógépen különféle tevékenységeket hajthat végre: programot futtathat, képernyőképet készíthet az annak monitorán zajló eseményekről, megfigyelheti a billentyűzetet, vagy épp meggátolhatja annak használatát.



11. ábra: a Beatrix kezelőfelülete

A Beatrix azért szerepel fenyegetésként több víruskereső program listáján, mivel a szerver a háttérben megbújva, a felhasználó tudta nélkül is képes futni, valamint a billentyűzet-megfigyelés, a fájlok a célszámítógépre való másolásának és a programok futtatásának lehetősége veszélyesnek bizonyulhat. A távoli számítógépen elvégezhető műveletek teljes listája a következő:

- CD meghajtó kinyitása vagy becsukása, a számítógép újraindítása vagy kikapcsolása, a szerver leállítása, a beviteli eszközök kikapcsolása (majd engedélyezése), megadott URL megnyitása böngészővel
- billentyűzet-megfigyelés
- felugró ablakok megjelenítése (négyféle típus tetszőleges szöveggel)
- fájl menedzsment (letöltés, feltöltés, törlés, átnevezés)
- parancs futtatása
- „Mátrix” képernyő lejátszása
- képernyő- és webkamerakép készítése
- folyamatkezelő (folyamatok leállításának lehetőségével)

Sasser

A Sasser a Windows operációs rendszerek egy ismert – és a vírus 2004-es megjelenésére már kijavított – sebezhetőségét kihasználó féreg. Hatékonysága a nem megfelelően frissített operációs rendszereknek volt köszönhető.

A vírus különböző IP címeket vizsgál meg, és amennyiben nyitva találja a 445-ös TCP portot, azon keresztül az operációs rendszer LSASS – a Windows biztonsági szabályait felügyelő – komponensében buffer túlcsordulást okozva megfertőzi a számítógépet. A vírust úgy tervezték, hogy működése során ne legyen a felhasználó számára megfigyelhető a jelenléte, ám a kódban felejtett hiba miatt a fertőzött számítógépen az lsass.exe instabillá válik, és gyakran a rendszer összeomlását eredményezi.

Az általunk lefordított vírus a Sasser távolról irányítható verziója, mely oktatási célokra készült. Ez a változat nem automatikusan működik, hanem a *sasser.exe <rendszer> <IP> <port> [-t]* parancs hatására megpróbál a célpont megadott portjához csatlakozni. A *rendszer* paraméter a 0, 1, 2 értékeket vehet fel aszerint, hogy a megtámadott rendszer Windows XP Professional, Windows 2000 Professional, vagy Windows 2000 Advanced Server. Az opcionális *-t* kapcsoló a célpont operációs rendszerét azonosítja. A program egy lefutásának eredménye látható a 12. ábrán.

```
C:\sasser.exe 0 192.168.1.10 4444 -t
MS04011 Lsasrv.dll RPC buffer overflow remote exploit v0.1
--- Coded by .::[ houseofdabus ]::. ---

[*] Target: IP: 192.168.1.10: OS: WinXP Professional    [universal]
lsass.exe
[*] Connecting to 192.168.1.10:445 ... OK
[*] Detecting remote OS: Windows 5.0
```

12. ábra: a Sasser egy futásának eredménye

5.2.2 Tesztsorozatok bemutatása

Tesztjeink során négyféle forgalmi mintát állítottunk elő az egyes vírusok tesztelésekor. Elsőként a rendszer „üresjáratáról” készítettünk mintát, amikor semmilyen aktív tevékenységet nem végeztünk az ügynökökön. Ezt egy olyan tesztfázis követte, mikor a virtuális gépekről netes forgalmat generáltunk. A harmadik típusú forgalmi minta esetén a netforgalmat az aktuálisan kiválasztott vírus forgalmával ötvöztük. Ez felel meg az ügynökök által gyanúsnak ítélt forgalomnak, amit az adatfeldolgozónak küld ellenőrzésre. A negyedik mintatípus kizárólag a vírus által realizált

forgalmat tartalmazta. Ezt tekintettük úgy, mint a honeypot által az adatfeldolgozónak küldött megjelölt minta.

Az Agobot esetén az egyik virtuális gépről a Gateway-en található IRC szerverhez csatlakozó vírusnak adtunk ki egy másik virtuális gépről utasításokat. A Beatrix program esetén annak szerver- és a kliensprogramját külön-külön virtuális gépen futtattuk, és a klienssel a szerverhez csatlakozva adtunk ki utasításokat a „fertőzött” gép számára. A Sasser vírust egy kiválasztott virtuális gépen futtattuk, paraméterként a teszthálózat egy másik tagját megjelölve.

5.3 Mérési eredmények

Mérési eredményeinket két csoportba sorolhatjuk: elsőként az egyes vírusokkal a teszthálózaton végzett mérések eredményeit mutatjuk be, ezek után pedig az Agobot forgalmának valós környezetben való felismerését célzó tesztünk eredménye következik.

5.3.1Vírusok vizsgálata

	Agobot	Agobot	Sasser	Beatrix	Beatrix	Beatrix
Összes Flow	933	37	117	788	182	48
TCP Folyamok	760 (81,4%)	34 (91,9%)	85 (72,7%)	676 (85,8%)	146 (80,2%)	30 (62,5%)
TCP Byte-ok	2804524 (95,2%)	26184 (18,3%)	3394884 (99,8%)	12795979 (99,8%)	958251 (99,4%)	166936 (97,7%)
TCP Csomagok	8990 (96,5%)	214 (58,2%)	4584 (99,3%)	18838 (99,1%)	2278 (98,0%)	580 (93,8%)
UDP Folyamok	172 (18,4%)	2 (5,4%)	28 (23,9%)	108 (13,7%)	32 (17,6%)	16 (33,3%)
UDP Byte-ok	24840 (0,8%)	211 (0,1%)	4022 (0,1%)	15085 (0,1%)	4302 (0,4%)	1875 (1,1%)
UDP Csomagok	172 (1,8%)	2 (0,5%)	28 (0,6%)	108 (0,6%)	32 (1,2%)	16 (2,6%)
ICMP Folyamok	1 (0,2%)	1 (2,7%)	4 (3,4%)	4 (0,5%)	4 (2,2%)	2 (4,2%)
ICMP Byte-ok	116736 (4,0%)	116736 (81,6%)	1113 (0,1%)	4400 (0,1%)	1661 (0,2%)	1999 (1,2%)
ICMP Csomagok	152 (1,7%)	152 (41,3%)	6 (0,1%)	48 (0,3%)	18 (0,8%)	22 (3,6%)
Aggregált folyamok száma	51	3	22	65	23	11
Botnettel kapcsolatos aggregált folyamok száma	2	2	1	1	1	1
Klaszterek száma	4	2	4	8	3	4
Botnetet tartalmazó klaszter azonosítója	0	1	0	1	1	1
Hatékonyság	**>98%**	**100,00%**	**~100%**	**100,00%**	**78,00%**	**100,00%**

3. táblázat: vírusok teszteredményei

Kezdeti méréseinket a tesztrendszerünk által biztosított kis méretű hálózatok forgalmának elemzésével végeztük, hogy az eredmények könnyen validálhatóak legyenek. Bár a kis hálózattal járó kevés minta több esetben is negatívan befolyásolta a tesztek eredményét, a 3. táblázatból kiolvasható, hogy alacsony flow érték esetén is képes igen hatékonyan működni rendszerünk.

A táblázatban az egyes oszlopokban a forgalmi mintában szereplő vírus neve alatt az adott mérésekben szereplő eredmények találhatóak. A sorokban a mérés alatt vizsgált flow-k teljes száma, a TCP, UDP és ICMP típusú csomagokra vonatkozó flow-, byte- és csomagszám adatok valamint ezek egymáshoz való viszonya, alattuk az aggregált flow-k száma, a botnettel összefüggő flow-k száma, a klaszterek száma, az illegális forgalmat tartalmazó klaszter azonosítója, végül pedig a vizsgálat hatékonysága látható.

A táblázatból látható, hogy a fenti mintákon az általunk készített algoritmus 78%-nál nagyobb pontossággal képes detektálni a kártékony kódokkal kapcsolatos forgalmat, négy esetben pedig 100%-os hatékonyságot sikerült elérni.

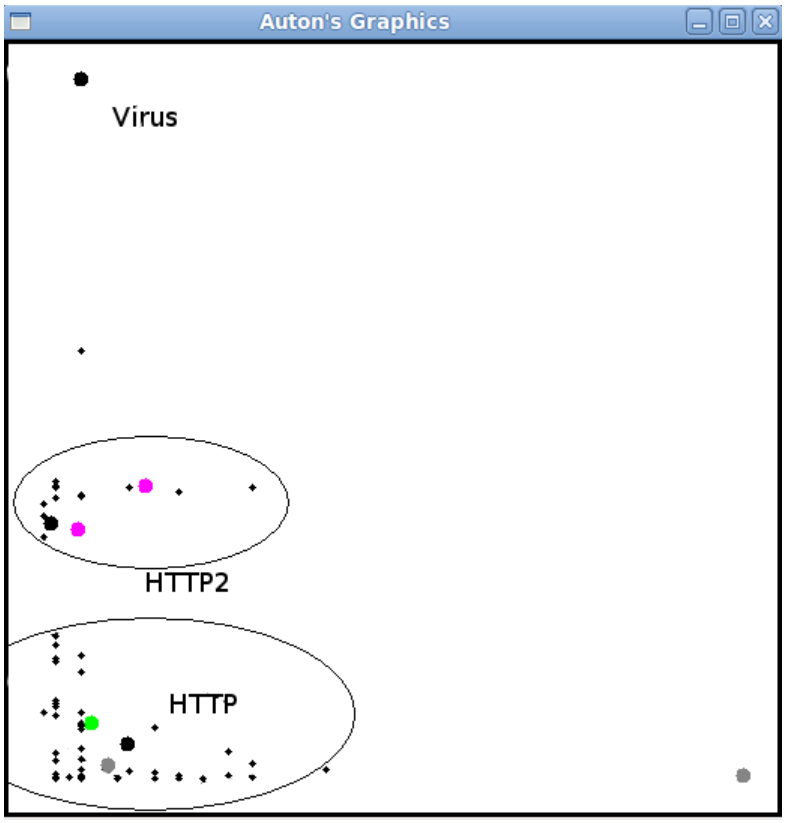
5.3.2 Valós forgalmi teszt

A teszthálózatunkon belül végzett mérések egyik hátulütője, hogy egy valós rendszerhez képest elenyésző forgalom mellett vizsgáltuk algoritmusunk hatékonyságát. Éppen ezért az Agobot forgalmának mérése során nyert lenyomatból két flow-t beültettük az egyetemi hálózatból nyert, nem aggregált mintába, majd az adatfeldolgozó gépen lefuttattuk rá botnet felismerő programunkat.

A program futása után megvizsgáltuk, hogy a botnetek melyik klaszterbe esnek. A 490 000 flow-t tartalmazó teljes vizsgált naplóból mindkét minta egy összesen 81 flow-t tartalmazó, a netes forgalomtól az 13. ábrán látható módon élesen elkülönülő klaszterbe került. Ez azt jelenti, hogy mindkét minta a teljes hálózati forgalom 0,49%-ára szűkített halmazba került, tehát gyakorlatilag detektáltuk a botnet forgalmát egy félmilliós mintában.

A 4.3.2 fejezetben bemutattuk, hogy a klaszterközéptől való távolságnak milyen hatása van elméletben a false positive eredmények számára. Ezt a gyakorlatban is megvizsgáltuk, mégpedig az Agobot forgalmát tartalmazó klaszteren. Az X-közép algoritmus lefuttatása után kiszámoltuk az egyes flow-k klaszterközéptől való távolságát, és azt tapasztaltuk, hogy az általunk választott paraméterekkel az Agobot flow-jánál mindössze egyetlen flow volt közelebb hozzá.

A mérés után további BME-s hálózatról szerzett forgalmi mintákba ültettük az előbbi Agobot mintákat, és ezeket is lemértük.



13. ábra: a klaszterek grafikus reprezentációja

d_0	False Positive flow-k átlaga (db)	A találatok és a tartalmazó klaszterméret aránya	A találatok és az összes vizsgált flow aránya
0,01	0,25	0,10%	0,000050%
0,02	6,5	2,63%	0,001300%
0,03	34,25	13,87%	0,006850%
0,04	70,5	28,54%	0,014100%
0,05	97	39,27%	0,019400%
0,06	115,28	46,67%	0,023056%
0,07	124,22	50,29%	0,024844%
0,08	133,78	54,16%	0,026756%
0,09	140	56,68%	0,028000%
0,1	146,22	59,20%	0,029244%
0,15	157	63,56%	0,031400%
0,2	160,72	65,07%	0,032144%
0,3	163,34	66,13%	0,032668%

4. táblázat: A klaszterközéptől való távolság hatása a false positive találatok mennyiségére

Az 4. táblázatban az általunk készített mérések eredményeinek átlaga látható. A táblázat azt mutatja meg, hogy bizonyos d_0 értékek esetén mennyi false positive eredményt kaptunk, és ez mekkora arány a tartalmazó klaszter illetve a teljes vizsgált forgalom méretéhez viszonyítva.

5.4 További eredmények

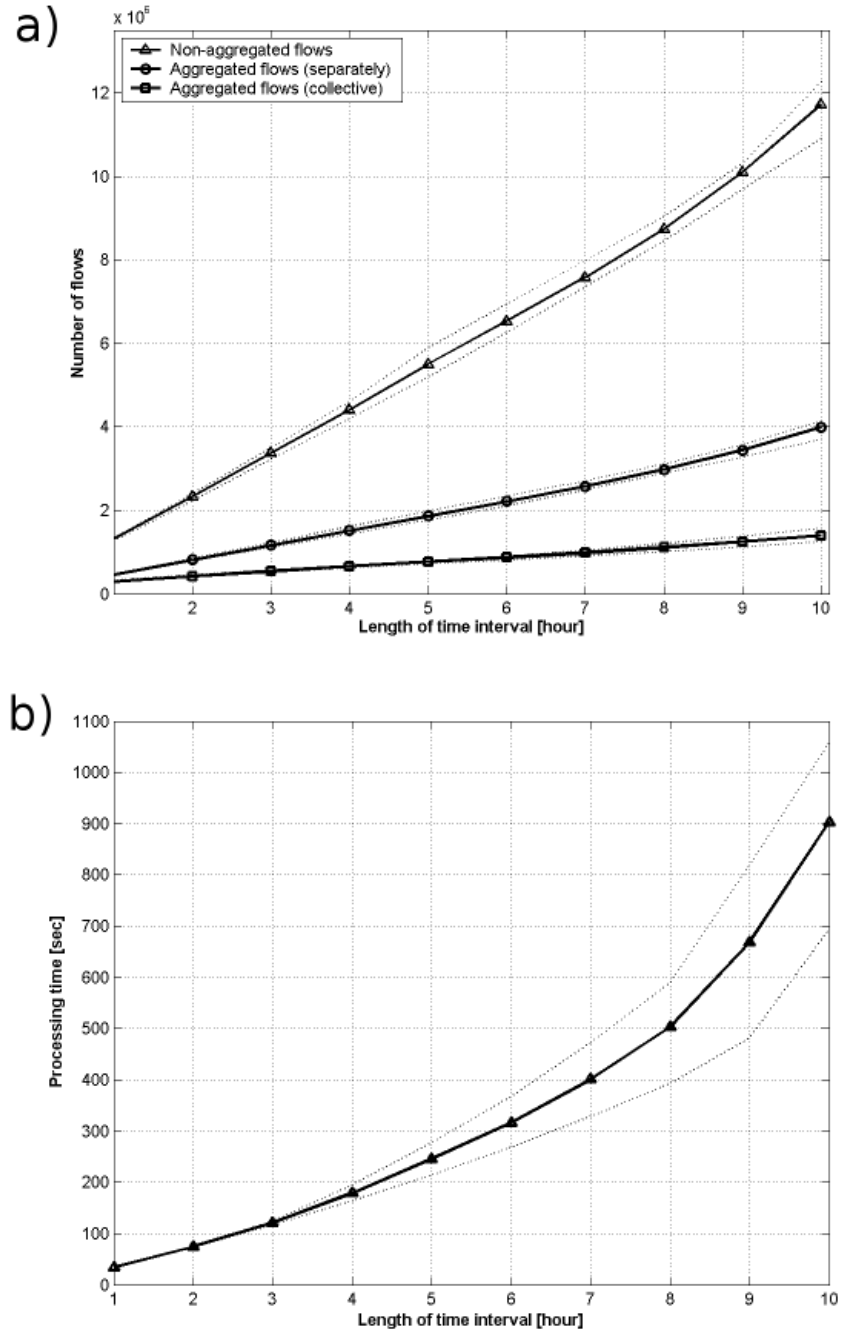
A vírusok tesztelésén kívül további vizsgálatokat is végeztünk, melyek eredményeit az alábbiakban mutatjuk be.

5.4.1 Aggregálás

Az aggregáló algoritmus hatékonyságát külön is lemértük a rendszer többi elemétől függetlenül. Az algoritmust C nyelven implementáltuk. A kód megírását nagy részben Kenyeres Péter (BME-TMIT) végezte. A teszt eredményeit azért mutatjuk be, mert ebből jól látszik az aggregálás tömörítési képessége, ami jelentősen hozzájárult a rendszer eredményességéhez.

Az algoritmust a BME-TMIT hálózatából 2008. április 17. és 19. között gyűjtött NetFlow logokon teszteltük. Mindegyik log 10 perces volt. Az adathalmaz teljes mérete 5,059 GB, körülbelül 100 millió flowt tartalmazott. Az outlier szűrésnél a tapasztalatok alapján az $\varepsilon = 1/2$ jó paraméterértéknek bizonyult. Először az algoritmust külön-külön futtattuk minden egyes 10 perces logra. Ekkor mind a 387 log esetében a tömörítés mértéke 0,3 és 0,35 közt volt, vagyis nagyjából ⅔-ával csökkent az adathalmaz mérete. Az átlagos futási idő egy logra körülbelül 10 másodperc volt.

Ezután az aggregáló algoritmust hosszabb időintervallumokra futtattuk. Ez nyilván javította a tömörítési arányt, hiszen nagyobb intervallumon több flow került egy-egy csoportba. Ugyanakkor ezzel nyilván nőtt a futási idő is. A 14/a. ábrán láthatjuk a flow-k számát , míg a 14/b. ábrán a futási idő alakulását a vizsgált időintervallum méretének függvényében.

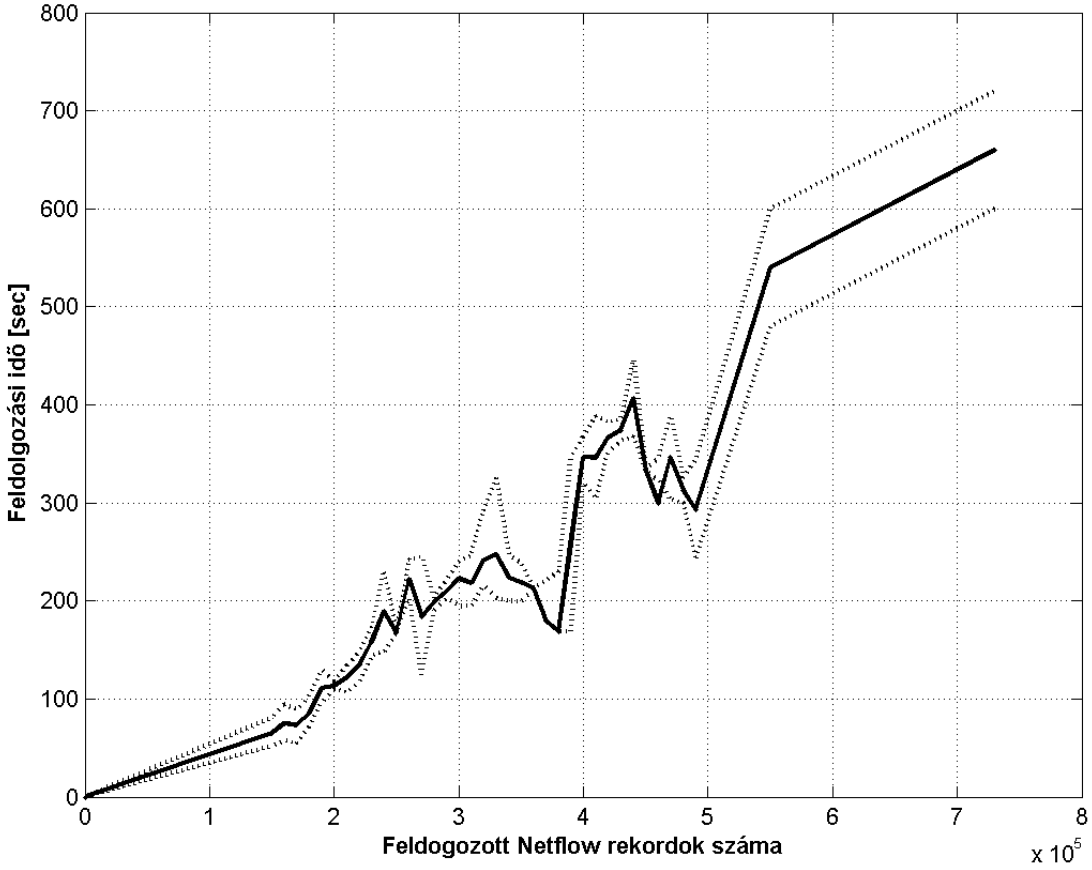


14. ábra: a) az aggregálás után a flow szám az időintervallum függvényében
b) a futási idő az időintervallum függvényében

A két grafikont összevetve azt mondhatjuk, hogy a legoptimálisabb, ha 5-6 óránként végezzük az aggregálást. Ezen értékek fölött a lineárisnál nagyobb mértékű a futási idő növekedése, amit már nem tudunk ellensúlyozni a tömörítési arány javításával. A pontos értékek természetesen erőforrásfüggők, viszont minden számítógépen létezik egy határ, mely fölött nem érdemes dolgozni.

5.4.2 Feldolgozási idő

Elsőként egy, az egyetemi hálózatról készített naplófájl segítségével figyeltük meg a feldolgozási idő változását a vizsgált flow-k függvényében.



15. ábra: programunk feldolgozási ideje az bejövő folyamok arányában

A 15. ábráról leolvasható, hogy egy 500 000 flow méretű napló esetén nagyjából 5 perc alatt képes programunk egy mai átlagos teljesítményű számítógépen feldolgozni az adatokat. Mint azt már bemutattuk a 4.3.1 fejezetben, azzal, hogy az egyes tagok a rajtuk átmenő forgalmat aggregálják, az adatfeldolgozó gép hatékonyabban képes erőforrásait felhasználni, ezzel jelentős mértékben csökkentve a számításra fordított időt.

5.5 Konklúzió, továbbfejlesztési javaslatok

A mérések során arra jutottunk, hogy teszthálózatunk működőképes, azaz képes az alhálózatokban történő forgalom naplózására, valamint algoritmusunk már kis forgalmi mintákon is képes igen

hatékonyan működni, valamint a valós, nagyméretű forgalomba beágyazott tesztünk során is képes volt detektálni a botnethez köthető forgalmat. Az ügynökökben történő aggregációnak köszönhetően az adatfeldolgozóban futó klaszterező algoritmusunk – mely programunk egyik legerőforrásigényesebb része – valós, nagyméretű forgalom esetén is kezelhető idő alatt képes elvégezni feladatát.

Azonban a kártékony forgalom felismerése még nem tökéletes, rendszerünk hatékonysága bizonyos esetekben a dolgozatban közölteknél rosszabb volt, valamint az egyes találatokat befogadó klaszterek méretezése – azaz a helyes klaszterközéptől való távolság megválasztása – is optimalizációra szorul. Ebből kifolyólag a jövőben az ilyen irányú kutatásokat kell majd végeznünk.

Ezen kívül hátra van még rendszerünk egészének egyben való tesztelése is, azaz a kártékony kódhoz tartozó forgalom vizsgálatának, valamint a hálózat különböző elemei közti kommunikációt és a forgalmi adatok megosztását biztosító szolgáltatások egyesítése. Mint azt már említettük, a feladat ez utóbbi része – Klettner Tamás által – már szintén implementálva van, az együttműködés tesztelésére mindössze a már meglévő teszthálózatunk kibővítése szükséges.

6 Összefoglalás

Dolgozatunkban bemutattuk a botnetet, mint a jelen és a jövő potenciálisan legfenyegetőbb veszélyforrását az Interneten. Bemutattuk a zombihálózatok képességeit, valamint különböző változatait, ezek erősségeit és gyenge pontjait. Az általuk képviselt fenyegetés ellenszereként felvázoltunk egy olyan architektúrát, melynek tagjai képesek detektálni a botneteket, valamint a detekció során nyert adatokat képesek egymás rendelkezésére bocsátani.

Bemutattuk a rendszer tervezéséhez szükséges elméleti tudás alapjait: a hálózati forgalom megfigyelését segítő NetFlow szolgáltatást, a peer-to-peer hálózatokat és az őket a rosszindulatú támadóktól védő SOS működését, majd a klaszterezés és az outlier szűrés matematikai alapjait vizsgáltuk meg, melyek a kártékony kódok kiszűréséhez voltak szükségesek.

Ezek után az architektúra tervezésekor meghozott döntéseinket taglaltuk, majd bemutattuk magát a megtervezett rendszer felépítését, és azt az általunk kidolgozott módszert, mely a tagok által a feldolgozónak küldött minták kiértékeléséhez és a botnetek által használt C&C csatorna detektálásához használható.

A következő részben felvázoltuk a rendszer forgalomanalizáló részének tesztelésére megalkotott hálózat és azok egyes tagjainak szerepét és megvalósítását. Ezt a tesztek során felhasznált vírusok leírása követte.

Dolgozatunk utolsó részében bemutattuk az általunk elvégzett teszteket, ezek eredményeit, majd az ezekből levont következtetéseinket és a rendszerre vonatkozó fejlesztési javaslatokat, valamint a további elvégzendő feladatokat.

7 Irodalomjegyzék

[1] Trend Micro: Taxonomy of botnet threats, 2006
http://us.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibrary/botnettaxonomywhitepapernovember2006.pdf

[2] Joris Evers: Computer crime costs $67 billion, FBI says; CNET News, 2006. január 19.
http://news.cnet.com/Computer-crime-costs-67-billion,-FBI-says/2100-7349_3-6028946.html

[3] Virus strikes 15 million PCs, UPI.com, 2009. január 26.
http://www.upi.com/Top_News/2009/01/26/Virus-strikes-15-million-PCs/UPI-19421232924206/

[4] Kim Willsher: French fighter planes grounded by computer virus, The Telegraph, 2009. február 7.
http://www.telegraph.co.uk/news/worldnews/europe/france/4547649/French-fighter-planes-grounded-by-computer-virus.html

[5] Lewis Page: MoD networks still malware-plagued after two weeks, The Register, 2009. január 20.
http://www.theregister.co.uk/2009/01/20/mod_malware_still_going_strong/

[6] Conficker-Wurm infiziert hunderte Bundeswehr-Rechner, PC Professionell, 2009. február 16.
http://www.pc-professionell.de/news/2009/02/16/conficker_wurm_infiziert_hunderte_bundeswehr_rechner

[7] Calculating the Size of the Downadup Outbreak, F-Secure Weblog, 2009. február 16.
http://www.f-secure.com/weblog/archives/00001584.html

[8] Karasaridis, A., Rexroad, B., Hoeflin, D.: Wide-scale botnet detection and characterization, In HotBots'07, 1st conference on 1st Workshop on Hot Topics in Understanding Botnets, pp. 7-7. USENIX Association, Cambridge, USA (2007)

[9] Sekar, V., Duffield, N., Spatscheck, O., Van Der Merwe, J., Zhang, H.: LADS: Large-scale Automated DDoS detection System, In USENIX ATC, pp. 171-184. (2006)

[10] Gu, G., Zhang, J., Lee., W.: BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic, In 15th Annual Network and Distributed System Security Symposium (NDSS) (2008)

[11] Choi, H., Lee, H., Lee, H, Kim H.: Botnet Detection by Monitoring Group Activities in DNS Traffic, In 7th IEEE International Conference on Computer and Information Technology (CIT), pp. 715-720., Aizu-Wakamatsu, Japan (2007)

[12] Nicolas Pioch: A Short IRC Primer, 1997. január 1.
http://irchelp.org/irchelp/ircprimer.html

[13] Sdbot [Hivatkozva: 2009. október]
http://threatinfo.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=WORM_SDBOT.AZ&VSect=T

[14] Phillip Porras, Hassen Saidi, Vinod Yegneswaran: An Analysis of Conficker's Logic and Rendezvous Points, 2009.
http://mtc.sri.com/Conficker/

[15] Jun Zhang: Storm Worm & Botnet Analysis, Websense Security Labs, 2008. június http://securitylabs.websense.com/content/Assets/Storm_Worm_Botnet_Analysis_-_June_2008.pdf

[16] F-Secure Virus Descriptions: Agobot [Hivatkozva: 2009. október] http://www.f-secure.com/v-descs/agobot.shtml

[17] Dancho Danchev: AlertPay hit by a large scale DDoS attack, ZDnet, 2008. december 1. http://blogs.zdnet.com/security/?p=2240

[18] Messagelabs Intelligence Q3/September 2009 http://www.messagelabs.com/mlireport/MLI_2009.09_Sept_SHSFINAL_EN.pdf

[19] Ken Chiang, Levi Lloyd: A Case Study of the Rustock Rootkit and Spam Bot, Sandia National Laboratories, 2007. április 3. http://www.usenix.org/event/hotbots07/tech/full_papers/chiang/chiang_html/

[20] Grum, M86 Security, 2009. március 19. http://www.m86security.com/trace/i/Grum,spambot.898~.asp

[21] Study shows how spammers cash in, BBC News, 2008. november 10. http://news.bbc.co.uk/2/hi/technology/7719281.stm

[22] Bogdan Calin: Statistics from 10,000 leaked Hotmail passwords, Acunetix, 2009. október 6. http://www.acunetix.com/blog/websecuritynews/statistics-from-10000-leaked-hotmail-passwords/

[23] Rbot [Hivatkozva: 2009. október] http://www.ca.com/us/securityadvisor/virusinfo/virus.aspx?id=39437

[24] Ping Wang, Sherri Sparks, Cliff C. Zou: An Advanced Hybrid Peer-to-Peer Botnet, 2007. http://www.usenix.org/event/hotbots07/tech/full_papers/wang/wang.pdf

[25] Evan Cooke, Farnam Jahanian, and Danny McPherson, The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets, Proc. of Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI '05), Boston, 2005.

[26] Product Literature: Cisco IOS NetFlow [Hivatkozva: 2009. október] http://www.cisco.com/en/US/products/ps6601/prod_literature.html

[27] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica: Looking up data in P2P systems, MIT Laboratory for Computer Science http://www.cs.berkeley.edu/~istoica/papers/2003/cacm03.pdf

[28] Keromytis, A. D., Misra, V., Rubenstein, D.: SOS: Secure Overlay Services, In: ACM SIGCOMM, pp. 61-72., Pittsburgh, USA (2002)

[29] R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification. Wiley, second edition, 2001

[30] Dan Pelleg , Andrew W. Moore, X-means: Extending K-means with Efficient Estimation of the Number of Clusters, Proceedings of the Seventeenth International Conference on Machine Learning, p.727-734, June 29-July 02, 2000

[31] J. Erman, M. Arlitt, and A. Mahanti. Traffic Classification using Clustering Algorithms.In SIGCOMM'06 MineNet Workshop, Pisa, Italy, September 2006.

[32] Peng, T., Leckie, C., Ramamohanarao, K.: Protection from distributed denial of service attacks

using history-based IP ltering, Communications, In: ICC '03. IEEE International Conference on, vol.1, pp. 482-486. (2003)

[33] Kargl, F., Maier, J., Weber, M.: Protecting web servers from distributed denial of service attacks, International World Wide Web Conference, pp. 514-524. ACM, Hong Kong (2001)

[34] A. McGregor, M. Hall, P. Lorier, and J. Brunskill: Flow Clustering Using Machine Learning Techniques. In PAM 2004, Antibes Juan-les-Pins, France, April 2004.

[35] S. Zander, T. Nguyen, and G. Armitage: Automated Traffic Classification and Application Identification using Machine Learning. In LCN'05, Sydney, Australia, November 2005.

[36] J. Erman, A. Mahanti, M. Arlitt, I. Cohen and C. Williamson: Offline/Realtime Traffic Classification Using Semi-Supervised Learning, Performance Evaluation Vol. 64., No. 9-12 (October 2007), pp. 1194-1213

[37] VMware Workstation [Hivatkozva: 2009. október] http://www.vmware.com/products/workstation/

[38] fprobe honlap [Hivatkozva: 2009. október] http://sourceforge.net/projects/fprobe/

[39] Flow-tools honlap [Hivatkozva: 2009. október] http://www.splintered.net/sw/flow-tools/docs/flow-tools.html

[40] UnrealIRCD dokumentáció [Hivatkozva: 2009. október] http://www.unrealircd.com/files/docs/unreal32docs.html

[41] GNU General Public License Version 3, 2007. június 29. http://www.gnu.org/licenses/gpl.html

[42] Agobot and the "Kit"-chen Sink, infectionvectors.com, 2004. július http://www.infectionvectors.com/vectors/kitchensink.htm