Ransomware és egyéb kártékony kódok elleni védekezés

Tamás Csongor

Konzulenseim: Dr. Bencsáth Boldizsár (CrySyS Lab), Guba Viktor (NISZ Zrt.)

Ransomware-ek gyűjtése

Ahhoz, hogy vizsgálódni tudjunk a témában, mindenekelőtt szükségünk van friss ransomware-ekre. Azért fontos, hogy frissek és aktívak legyenek, mert így a malware gyártók jelen technikáival ismerkedhetünk meg.

E feladat megoldásához először információt gyűjtöttem arról, hogy mások, akik hasonló témában dolgoztak, milyen módszerrel gyűjtöttek mintákat. Számos cikk elolvasása után végül arra jutottam, hogy legtöbben publikusan elérhető malware adatbázisokat használtak vagy mintákat kaptak egy anti-vírus üzemeltetőtől. A kigyűjtött adatbázisokat áttekintve megállapítottam, hogy számos közülük már sajnos nem üzemel. A továbbra is aktív státuszban levők között voltak ingyenesek és fizetősök is, amik lehetővé tették fájlok letöltését SHA hash-ük alapján.

Azonban mindegyiküknek volt egy hiányossága, ami alkalmatlanná tette őket számunkra: nem lehetett család vagy osztály szerint listázni, keresni bennük. Az egyetlen ilyen szolgáltatást a VirusTotal nyújtja, ahol YARA hunting szabályok alkalmazásával kereshetünk a folyamatosan érkező fájlok között, majd ezeket le is tölthetjük.

Első opcióként tehát ezzel próbálkoztam. Interneten talált és néhány minta alapján saját magam által írt hunting szabályokat fogalmaztam meg WannaCry, BadRabbit, Petya és ismeretlen kategóriájú zsarolóvírusok keresésére. Tudtam, hogy hosszú idő is eltelhet, míg ezek a szabályok találatot jeleznek, ezért folytattam a munkám más irányokban. Végül 3 hónap után sem jelzett egyik szabály sem, úgyhogy ez az út hosszú távon biztosan nem járható.

Ezek alapján új célt adtam a kutatásnak: készítsünk malware adatbázist, ahol családok szerint csoportosítva vannak a minták. Ehhez rendelkezésre állt számomra az Ukatemi cég malware adatbázisa, mely 300 000 000 mintát tartalmaz, napi ~40 000 újonnan érkező fájllal.

Klasszifikációs lehetőségek:

A csoportosítás számos módon történhet. Nekem egy olyan módszert kell találnom, ami skálázható (hiszen 300 000 000 malware-en kell működnie), ennek megfelelően gyors, viszonylag rövid időn belül implementálható és megfelelően pontos eredménnyel szolgál. További olvasás után három módszert találtam, mellyel alkalmas lehet malware klasszifikációra: dinamikus eljárás, statikus eljárás, ezen belül mesterséges intelligencia valamint Locality Sensitive Hash eljárások alkalmazása.

Dinamikus eljárás:

Dinamikus eljárásnak ezen feladat esetében azt nevezzük, amikor egy virtualizált környezetben futtatjuk a malware binárisát és a futás során bekövetkező változásokból gyűjtünk adatokat, ezek alapján végezzük el a csoportosítást. Ennek elvégzésére lehetőségünk nyílik például Cuckoo sandboxon keresztül.

Azonban a feladat megoldására ez az út nem járható. Egy program futtatása jó esetben egy percet vesz igénybe, mivel a futás után vissza kell állítanunk a virtuális környezetet az eredeti állapotába, hogy újabb mintát indíthassunk el. Ez ilyen nagy adatbázis esetén kivitelezhetetlen. Továbbá dinamikus eljárás alkalmazásához monitorozandó tulajdonságokat kell definiálnunk és megállapítanunk, hogy milyen tulajdonságcsoport esetén milyen családba tartozik a malware. Ez nem egy féléves feladat időkerete.

Statikus eljárás:

Statikus eljárás esetén a malware-eket, azok futtatása nélkül igyekszünk, csak a fájlok tartalma alapján kategorizálni. Ennek az eljárásnak a legközismertebb és elterjedtebb módja mesterséges intelligencia alkalmazása. Ennél a módszernél a binárisok alapján definiálunk egy kellően diverz tulajdonsághalmazt, mely alkalmas a családon belüli hasonlóság, valamint a családok közötti különbségek leírására. A tulajdonsághalmaz megállapítása után minden mintára ki kell számolni és eltárolni az egyes tulajdonságok értékeit, majd ezeket egy neurális hálónak vagy más, akár modell alapú osztályozónak nem ellenőrzött módon betanítani. Végül az így kapott eszköz alkalmas a csoportok felismerésére.

Sajnos ennek a módszernek is sok hátránya van. A hasonlóságokat és különbségeket megfelelően jellemző tulajdonsághalmaz meghatározása korántsem egyszerű feladat. Ahogy a feladat megoldására alkalmas osztályozó megtervezése, implementálása és tanítása sem. Bár ennek megoldásához felhasználhatnánk a Microsoft Malware Classification Challenge (2015)-ben elért eredményeket.

További kutatás során végül felfedeztem a Locality Sensitive Hash eljárásokat, melyek ígéretesnek bizonyultak a feladat megoldására. Munkám további része ehhez kapcsolódik.

Locality Sensitive Hashing (LSH):

Az LSH eljárások a standard, közismert (MD5, SHA stb.) eljárásokkal ellentétben a hash-ek közti ütközést igyekszik maximalizálni. Célja, hogy hasonló bemenethez hasonló hash értéket generáljon, különbözőhöz pedig különbözőt. A hash-ek közti hasonlóság nem feltétlen szemre értendő. Az LSH algoritmusok kér részből állnak: egy hash generáló és egy összehasonlító algoritmusból. A generáló algoritmus valamilyen módszerrel leképezi a bemeneti bájtfolyamot egy rögzített vagy változó hosszú hash értékre. Az összehasonlító algoritmus egyfajta távolságot vagy pontszámot rendel bármely két hash értékhez, mely jellemzi a hasonlóságuk mértékét. Természetesen az összehasonlító algoritmust a hash készítői igyekeznek a generáló algoritmus ismeretében kialakítani, hogy a hash-ek közti hasonlósági mérték a fájlok közti hasonlóságot jellemezze.

Az LSH algoritmusokat két csoportba rendezhetjük a bemeneti adat tekintetében: specifikus illetve általános. Specifikus esetben a generáló algoritmus bemenetének meghatározott szerkezetűnek kell lennie. Például csak előre kitűzött fájl formátumokon futtatható. Általános esetben a bemenetre bármilyen bájtfolyam kapcsolható. A specifikus módszer előnye, hogy pontosabb működés érhető el vele, értve ez alatt a hash-ek közti hasonlóság bemenetek közti hasonlóságra való leképezését. Azonban lassabbak, mint az általános algoritmusok és csak speciális bemeneteken futtathatók. Ezzel szemben az általános algoritmusok előnyösek, hiszen bármilyen bemeneti bájtfolyamon futtathatók valamint ebből kifolyólag gyorsabbak is. Viszont sokkal kevésbé bízhatunk a hasonlóság megállapított értékében.

Binárisokra jelenleg nincs specifikus algoritmus. Ennek elkészítése nagyon hasonló lenne a mesterséges intelligenciás megoldáshoz. A tulajdonsághalmaz egyes értékeiből képezzük a hash-t. Összehasonlító algoritmusként pedig használhatunk mind osztályozót, vagy logikus hasonlóság megállapíthatósága esetén egy determinisztikus algoritmust. Későbbi feladat lehet egy ilyen algoritmus elkészítése.

Specifikus algoritmus hiánya miatt a továbbiakban csak általános algoritmusokkal foglalkozom.

SSDEEP:

Az ssdeep az egyetlen elterjedt LSH algoritmus. Ahol elérhető LSH hash ott ez elérhető. Az SSDEEP egy spamsum-nak keresztelt algoritmusra épül, melyet Andrew Tridgell 2002-ben spam-detektálás

segítésére hozott létre. Ehhez szükség volt egy adatbázisra, mely tartalmazta ismert spam emailek ssdeep hash értékeit. Egy beérkező emailre ki kellett számolni a hash-t, végül ennek értékét az összehasonlító algoritmus segítségével összehasonlítani az adatbázisban található hashekkel. Ha a távolság (score) értéke elért egy küszöböt, akkor vélhetően a beérkező email is spam volt.

Az algoritmus működéséről nem találtam leírást, ezért az elérhető forráskód alapján fejtettem meg a mechanizmust. Az ssdeep egy Content Triggered Piecewise Hashing (CTPH) eljárás. Az elnevezésből a Content Triggered rész az ssdeepben implementált Adler-32 checksum-ra épülő rolling hash-re utal. Ez egy hét byte-os csúszó ablakkal halad végig a fájl bájtjain és a bájt hetesek bizonyos pszeudorandom értékeinél jelez (trigger). Az ekkor bekövetkező lépésre utal Piecewise Hashing rész. Minden bájt olvasásakor egy másik hash az úgy nevezett block hash is frissül, ez mindössze egy bájt hosszú. Ennek értéke minden rolling hash jelzéskor (trigger) elmentődik (ezek sorozata alkotja végül a hash-t) valamit visszaáll egy kezdeti értékre (reset). Gyakorlatilag a végső hash bájtjai a block hash értékeit mutatják egymást követő jelzések közötti bájtsorozatokra kiszámolva. A pszeudorandom jelzés természetesen determinisztikusan működik, csak a hetes bájtablak aktuális tartalmától függ.

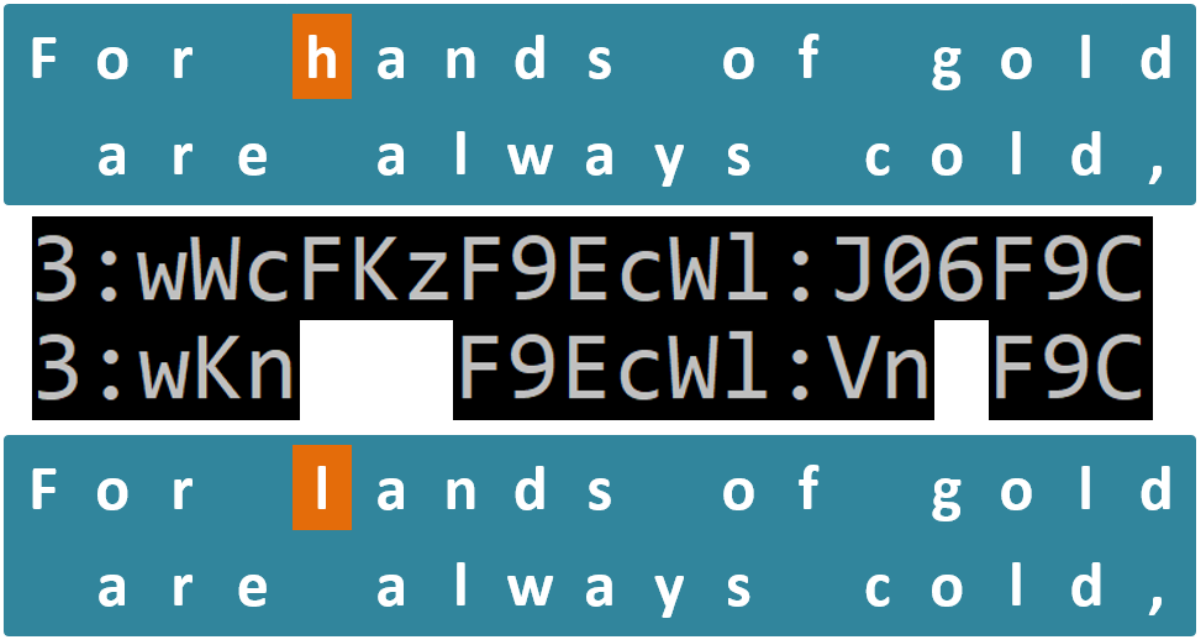
A hash generáló program egyszerre több ˝block size˝szerint növekvő sorba rendezett hash-t is számol egymással párhuzamosan. Ezek között a különbség a jelzések gyakoriságában van. Erre a legelső spamsum implementáció block size-t használt (tekintsük úgy, mintha minden block határon jelzés lenne). Az ssdeep-ben a block size-nak csak a hash-ek összehasonlításakor van szerepe, valójában nincs köze a jelzések közt található bájt sorozat hosszához. Minden egyes jelzésnél (megint egyedül a csúszó ablak aktuális értékének függvényében) a legkisebb csúszó ablakútól elkezdve több hash is frissülhet. Erre azért van szükség, mert egy nagy fájl esetén a legkisebb block size-ú hash nagyon hosszú lenne, az ssdeep hash pedig maximum 64+32+2+5 hosszú. Így végül a generáló algoritmus kimenete a legnagyobb befejezett block size-ú hashből kerül ki.

Az algoritmus működéséről grafikus magyarázó ábrákat is készítettem, hogy a jövőben megkönnyítsem az ezzel foglalkozni vágyók munkáját.

Ahogy fentebb említettem, egy LSH algoritmus a hash-ek összehasonlításához használandó algoritmust is tartalmazza. Az ssdeep esetén ez röviden string edit distance névvel illethető. Értéke megadja, hogy milyen sokat kell változtatnunk az egyik hash-en, hogy a másikat kapjuk. Mivel a hash a fájlnak egy leképzése, ezért ez vélhetően a fájlok közötti hasonlóság mértékét is leírja.

Az összehasonlítás kimenete végül egy nullától százig terjedő pontszám. A nulla teljes eltérést, a száz tökéletes hasonlóságot jelent.

A működés alapján megállapítottam, hogy az algoritmus a fájlt maximum 64, általában eltérő hosszú részre osztja. Mindegyik részt leképez egy bájt-ra végül ebből készül a hash. Az alábbi ábra szemlélteti ennek hasznosságát egy nagyon egyszerű példán. Itt a két szöveg csak a narancssárgával kiemelt karakterben különbözik. A hash-ek közötti hasonlóság már szemre is megállapítható.



A 64 részre osztás megfelelő módszer lehet spam emailek összehasonlítására, hiszen ezek gyártói egy email sablont kisebb változtatásokkal többször is felhasználnak. Például egy nevet kicserélnek benne vagy egy szót egy szinonimájára, de akár egy írásjel lecserélése is elegendő a standard hash alapú védekezés kijátszására. Azonban ezek a módszerek nem verik át az LSH-kat.

Binárisok esetében azonban más a helyzet. Esetükben egyetlen fordító flag megváltoztatásával drasztikus különbség érhető el a lefordított állományban. Tegyük fel például, hogy a mind a 64 részbe valamilyen módon beillesztünk egy-egy NOP (No OPeration) utasítást. Ez nem változtatja meg a program működését, azonban a 64 részből képzett minden block hash bájt értéke megváltozik és a két fájlra a kimeneti hash-ek értéke teljesen különböző lesz. Ez csupán egyetlen példa, amivel ki lehet játszani az ssdeep-et, ezen gondolatmenet alapján rengeteget találhatunk még.

Azonban az imént említett támadás főként akkor jelent problémát, ha a támadók azt a célt is kitűzik maguk elé, hogy átverjék az ssdeep alapú összehasonlítást és szándékosan ennek megfelelően módosítják programjaikat.

Az algoritmus működését tovább lehetne elemezni, újabb támadásokat implementálva, azonban jelen feladat esetén ez csupán egy kitekintés. Számunkra az a fontos, hogy a valóságban alkalmas-e malware-ek hasonlóságának megállapítására. Ezzel a dokumentum későbbi részében foglalkozom.

TLSH:

A TLSH a TrendMicro Locality Sensitive Hash rövidítése. Ez egy újabb, 2013-ban publikált eljárás. Lényege, hogy statisztikát készít a fájlban előforduló bájt n-esekről. Ehhez egy 5 bájtos csúszó ablakkal halad végig a bemenet bájtjain. A csúszó ablakban található bájt 3-asokat vizsgál, ezeket leképezi egy bájtra majd eggyel növeli annak a vödörnek az értékét, ami a képzett bájt előfordulásait tartalmazza. Végül ebből az eloszlásból képzi a hash-t, mely rögzítetten 64 bájt hosszú.

Az összehasonlítás leginkább Hamming távolság számítására hasonlít a két hash között. A kimeneti érték nullától egészen ezres nagyságrendig felmehet, bár az ezret ritkán haladja meg. A nulla tökéletes hasonlóságot jelent, ellentétben az ssdeep-pel.

A publikáló cikkben szerepel az algoritmus vizsgálata és értékelése is. Kivételesen binárisokat, sőt, malware-eket is használtak a módszer eredményességének megállapítására, azonban ez csupán 169 malware-t tartalmazott. Véleményem szerint ilyen kicsi adathalmaz esetén csupán felületes eredményt kapunk az algoritmus malware-eken értelmezett pontosságáról.

SDHASH:

Az sdhash-t 2009-ben implementálták. Alapvető ötlete, hogy statisztikailag valószínűtlen tulajdonságokat keresünk a bemenet bájtjaiban, ezekből képződik a hash érték. Egy valószínűtlen tulajdonság jelenléte egyedien jellemez egy fájlt vagy fájl csoportot.

A valószínűtlen tulajdonságok megállapításához a bementet 64 bájtos egységenként tekinti, ezekre számol entrópiát. Végül az entrópia eloszlásából állapít meg valószínűtlen tulajdonságokat.

Sajnos azt sdhash-t bemutató cikkben sem végezték el az algoritmus tesztelését malware-eken.

Hash-ek tesztelése

A továbbiakban a fent említett LSH-k valamint ezekre épülő csoportosító algoritmusok eredményességét vizsgálom.

A munkát a Microsoft Malware Classification Challenge adatbázisán kezdtem, mivel ez volt az egyetlen, számomra ingyenesen elérhető csoportosított malware gyűjtemény. Ebben a halmazban kb 400GB fájl található, összesen 9 családból. Az egyes fájlokhoz rendelkezésünkre állt, hogy melyik családba tartozik.

A vizsgálathoz először családonként mappákba rendeztem a fájlokat, mindegyikhez generáltam annak hash értékét mindhárom hash-hez. Ezt követően, mivel az adathalmaz hatalmas volt, ezért csak annak egy részén, 2 vagy 3 család összevonásával keletkezett halmazon dolgoztam. Erre azért volt szükség, mert a csoportosításhoz bármely két fájl között ki kell számolni a távolságot minden hash szerint, aminek időigénye a mintaszám négyzetével arányosan nő.

Második lépésként implementáltam három csoportosító algoritmust, melyek a fájlok közötti távolságokat kapták meg bemenetként és az általuk képzett csoportokat, valamint a hibás osztályozások számát adták a kimenetre.

Az első algoritmus akkor tette egy már létező csoportba a vizsgált fájlt, ha a csoportnak volt olyan eleme, amihez a fájl ˝közel˝ volt. A közel alatt egy küszöbön belüli érték értendő. Ez a módszer nem bizonyult hatékonynak. Minden alkalommal egy nagy csoport jött létre és számos 1-2 elemet tartalmazó. A nagy csoporton belül vegyesen voltak fájlok mindhárom osztályból. Azonban ez mégis alkalmas volt annak megállapítására, hogy ilyen mértékben nem bízhatunk az algoritmusokban. Hiszen ekkor mindenképp volt egy olyan páros, akik különböző családból származtak, mégis közelinek lettek ítélve. Az algoritmusok közül a TLSH és az SSDEEP teljesítettek a legjobban, az SDHASH ezeknél sokkal rosszabbul.

A második algoritmus a k-means egy testvére a k-medoids volt. A kettő abban különbözik, hogy míg a k-means-nél a csoport várható értéke adja a következő iteráció új középpontját, addig a k-medoidsnál minden középpontnak egy fájlnak (hash-nek) kell lennie, hiszen nincsenek ˝koordinátáink˝ amikre várható értéket (átlagot) tudnánk számolni. Ennek az algoritmusnak a távolságokon kívül még egy k értéket is meg kell adnunk, ennyi csoportot fog létrehozni. Azt tapasztaltam, hogy 3 család esetén kb. 30 egy jó választandó k érték. Elég magas, hogy legyen szabadsága az osztályozónak, mégis minden létrejött osztály népes.

Végül egy saját algoritmust is implementáltam. Ez akkor ad egy már létező csoporthoz egy új fájlt, ha az legalább a csoport néhány eleméhez közel van és a távolsága a nem csoportban levő mintáktól hasonló eloszlású, mint a csoport tagjainak a nem tagoktól való távolsága. Ez az algoritmus produkálta a legkevesebb hibás osztályozást, valamint szabadon rendelkezett a csoportok száma felől. Azonban a sok beágyazott iteráció miatt nagyon lassan dolgozott. Végül megvizsgáltam, hogy a két feltétel mennyire befolyásolja egy fájl csoportból való kitiltását és azt tapasztaltam, hogy a második feltétel csak nagyon kis hányadnál játszik szerepet, ezért ezt végül kivettem az algoritmusból. Így kicsit romlott a teljesítőképesség, azonban drasztikusan lecsökkent a futásidő.

Mindhárom csoportosító algoritmusnál a TLSH bizonyult a legjobbnak, nyomában az SSDEEP-pel és sokkal mögöttük lemaradva érkezett az SDHASH. Végül rájöttem az SDHASH gyenge teljesítményének okára. Az adatbázisban elérhető fájlok nem binárisok voltak, hanem azoknak object dump leképezéseik. Ez nagyon befolyásolhatta az SDHASH működését, hiszen egy valószínűtlen tulajdonságnak tekinthető, hogy minden fájlban hexadecimális karakterek vannak és kettesével elválasztja őket egy szóköz. Ezért úgy ítéltem ez az adathalmaz nem alkalmas eredményesebb vizsgálatra. Így áttértem az Ukatemi cég egy napi feed-jére.

Munka az Ukatemi adatbázisán

Az Ukatemi adatbázisa körülbelül 300 000 000 valódi malware binárist tartalmaz, azonban ezek nincsenek osztályokba rendezve. Én ennek csak egy részén, 34681 fájlon kísérleteztem. Sőt először ebből is leválasztottam egy kisebb, 468 fájlt tartalmazó halmazt, hogy a távolságok kiszámítása ne vegyen igénybe több órát.

Mivel az előző adatbázison futtatott tesztek alapján a harmadik eljárás lecsonkított változata volt a legeredményesebb, ezért a továbbiakban csak ezt használtam. Sajnos az eredményül kapott csoportok helyességének megállapítására nem volt jobb mód, mint a binárisokat áttekintve, decompile-olva, VirusTotal-ra, Hybrid-Analysis-ra feltöltve összehasonlítani a csoportok tagjait. Ezt néhány tíz csoportra megtettem és azt tapasztaltam, hogy SSDEEP és TLSH esetén, amik egy csoportba lettek sorolva, azok valóban is egy malware családba tartoztak, azonban SDHASH esetén voltak tévedések. Például androidos és x86-os malware-ek között nagyon nagy hasonlóságot állapított meg, amiben nem volt igaza. Mindegyik algoritmusnak hiányossága volt, hogy voltak különböző csoportok, amik legjobb tudásom szerint azonos családba tartozó fájlokat tartalmaztak.

Azonban itt meg kell, hogy említsem az SSDEEP egy további nagy hibáját. Ahogy korábban említettem, az összehasonlítás eredményeként 0-100 intervallumban kapunk egy értéket. Sajnos egy 1-es érték már nagy valószínűséggel jelent hasonló eredetet a két forrás fájl között, így nincs lehetőségünk a küszöböt állítva megválasztani a False Positive rate-et. Tapasztalatom szerint ezzel ellentétben a TLSH-nál egy 50 alatti értéknél lehetünk szinte biztosak a közös eredetben, de ezt a küszöböt binárisok esetén akár 150-ig is érdemes lehet kitolni, így akár nagyon eltérő variánsok közötti hasonlóságot is felfedezhetünk, természetesen néhány rossz osztályozás ellenében.

Úgy tapasztaltam, hogy egy 70-es TLSH küszöb esetén nincsenek rossz osztályozások, azonban egy család több kisebb részre szakad variánsok szerint. Így voltak olyan hasonlóságok, amiket az SSDEEP felfedezett, de a TLSH nem, de mégy így is több olyan volt, amit a TLSH talált meg és az SSDEEP nem. A TLSH küszöbét följebb emelve 100-ra, már nullára csökken azon hasonlóságok száma, amiket az SSDEEP megtalált, de a TLSH nem, azonban a csoportokban megjelennek nem oda tartozó fájlok is, ugyan csak néhány csoportban egy-két helytelen fájl.

Létrehoztam további néhány scriptet, melyek automatizálják az elvégzett feladatokat. Egy mappában tárolt malware-ekre kiszámolják a hash-eket, összehasonlítják őket, ezek alapján csoportokat képeznek

majd kiírják az algoritmusok által talált csoportok eltéréseit. Innen már csak a csoportok helyességének értékelését kellene elvégezni kézzel, de gyakorlatilag ezzel egy csoportosított malware adatbázist hoznánk létre.

Kiterjesztés a teljes adatbázisra

Munkám legnagyobb eredménye az lenne, ha ez alapján klasszifikálni tudnánk a teljes adatbázis összes fájlját. Ehhez ugyanazokat a lépéseket kell elvégezni, mint amit automatizáltam a kisebb teszt halmazon.

Első lépés a hash-ek generálása. Ezt egy map-reduce feladattal lehet megoldani, mely minden fájlon elvégzi a hash generálást majd eltárolja a kimenetet. Ehhez módosítottam az eredeti programokat, új kapcsolókat vezettem be, melyek a hash generálás és összehasonlítás interfészeit kompatibilissé teszik az adatbázissal. Ezek az alábbiakban láthatók:

- `--- generate hash ---`
- `tlsh -q <file>`
- `ssdeep -q <file>`
- `sdhash <file>`
- `--- compare 2 hashes ---`
- `tlsh -c <hash> -d <hash>`
- `ssdeep -C <hash> <hash>`
- `echo "<hash>\n<hash>" | sdhash -C`

Ezek lefuttatására azonban technikai okok miatt még nem került sor.

Második lépésként ki kell számolni bármely két hash között a távolságot. Kis számolás után kiderül, hogy ez ekkora adattömegen több százezer évbe telne, ezért nem kivitelezhető. A jövőben megoldás lehet, ha nem hasonlítjuk össze bármely két fájlt, hanem minden, már létrejött csoportból kiemelünk pár, a csoportot jó jellemző mintát és csak ezekkel végezzük el a távolság számítást. Ennek nyeresége attól függ, hogy valójában mennyi csoportba tartoznak az adatbázisban tárolt fájlok, hiszen ennek négyzetével lenne arányos a teljes lefutás időigénye. Véleményem szerint így akár fél év alatt is elvégezhető a teljes adatbázis csoportosítása, azonban ez csak egy erős becslés, nincs mögötte mérnöki számítás.

Mivel a hash-ek hiányában a folyamat további lépései csupán elméletiek, ezért a csoportosítás helyett akár egy új feladatot is megfogalmazhatunk: lehessen hasonlóság alapján keresni az adatbázisban. Ez valójában a csoportosító feladat egyetlen fájl csoportosítását végző folyamatának felel meg, mégis egy önmagában értékes funkció. Ennek megvalósítását el is végeztem a kicsi, 34681 mintát tartalmazó halmazon, így ebben már néhány másodperc alatt lehet keresni.

Összegzés:

Féléves munkám során mélyrehatóan elemeztem az iparban kizárólagosan elterjedt LSH algoritmust, az SSDEEP-et. Megállapítottam hiányosságait és hibáit malware-ek hasonlóságának vizsgálatában. Ennek alternatíváiként megvizsgáltam a TLSH-t és az SDHASH-t. A három eljárás működőképességét valós mintákon vizsgáltam és rájuk építve csoportosító algoritmusokat hoztam létre. Az így létrehozott

rendszer működését teszteltem és kiértékeltem, alkalmasnak találtam malware-ek csoportosítására. Megterveztem és előkészítettem az eljárás 300 000 000 mintán történő lefuttatását. Végül javaslatot tettem a munka jövőbeli folytatására.