Budapesti Műszaki és Gazdaságtudományi Egyetem
Hálózati Rendszerek és Szolgáltatások Tanszék

Fokozott kiberbiztonság és rosszindulatú kódok elleni védelem beágyazott IoT-eszközök számára

Tézisfüzet

Futóné Papp Dorottya

Konzulens: Buttyán Levente, habil. Ph.D.

Budapest, Magyarország
2020

Köszönetnyilvánítás

Az itt bemutatott kutatás a következő projektek, programok és ügynökségek pénzügyi támogatását élvezte:

- SETIT projekt (2018-1.2.1-NKP-2018-00004)[1]
- A H2020-ECSEL program a 692474 számú támogatási szerződésen keresztül (AMASS)
- SECREDAS projekt, ami az ECSEL Joint Undertakingtől kapott támogatást az Európai Unió Horizon2020 kutatási és innovációs programjának részeként
- EU ARTEMIS projekt EMC^2 része, ami az Osztrák Kutatásfejlesztési Ügynökség (FFG), valamint a H2020-ECSEL program finanszírozásával valósult meg.

Hálásan köszönöm továbbá az EFOP ösztöndíjból származó finanszírozást is (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Infocommunications), melynek társfinanszírozója az Európai Szociális Alap.

1. Bevezetés

A beágyazott eszközök jól meghatározott feladatok végrehajtására kifejlesztett speciális eszközök. Számos modern alkalmazási területük van, többek között az egészségügyben, a közlekedésben és a mezőgazdaságban. A beágyazott eszközök internetkapcsolata a közelmúlt egyik jelentős fejlődése, ami az ún. dolgok internetéhez (röviden IoT) vezetett. Az internetkapcsolat számos új és innovatív alkalmazást tett lehetővé, emiatt ma már általában beágyazott IoT-eszközökről beszélhetünk. Például, az okos mérőkkel felszerelt házak automatikusan jelenthetik a víz- és energiafelhasználást. Az okos városok jelzőlámpái érzékelik a forgalom sűrűségét, és ennek megfelelően irányíthatják a forgalmat a dugók csökkentésért. Az orvosok távolról is felügyelhetnek bizonyos beültetett egészségügyi eszközöket, pl. pacemakereket.

Az internetkapcsolat ugyanakkor a támadók számára is elérhetővé tette a beágyazott IoT-eszközöket. Ezen eszközök támadása pedig racionális tevékenység a támadók szempontjából. A beágyazott IoT-eszközök, különösen az egészségügyi és az okos otthon alkalmazási területeken, érzékeny és személyes adatokat kezelnek, amelyeket érdemes ellopni. Ezenkívül, bár egyenként a beágyazott IoT-eszközök számítási teljesítménye kicsi, ha ezeket az eszközöket együttesen vesszük figyelembe, a számítási kapacitásuk már nem elhanyagolható. Műszaki szempontból a beágyazott IoT-eszközök hardveres és szoftveres komponensei nem különböznek nagyon a hagyományos számítógépekétől. Hiányos konfigurációjuk (pl. megfelelő hitelesítés nélkül elérhető, nyitott portok), valamint az

[1] A 2018-1.2.1-NKP-2018-00004 számú projekt a Nemzeti Kutatási és Innovációs Alapból biztosított támogatással, a "Nemzeti Kiválósági Program: 2018-1.2.1-NKP" pályázati program finanszírozásában valósult meg.

alapértelmezett vagy fix jelszavak megkönnyítik az eszköz elérését. Technikailag is lehetséges kihasználni az IoT-eszközökön futó szoftverkomponensek sérülékenységeit, beleértve azok firmware-jét és operációs rendszerét (OS), amely gyakran valamilyen beágyazott Linux-változatra épül. Nem meglepő tehát, hogy a biztonsági szakértők a beágyazott IoT-eszközöket célzó vírusok, férgek, trójaiak és más típusú kártékony kódok számának növekedését észlelték. Az egyik leghírhedtebb példa a Mirai [2], amely több százezer IoT-eszközt fertőzött meg, és 2016-ban az egyik legnagyobb elosztott szolgáltatásmegtagadásos támadást indította népszerű internetes szolgáltatások ellen. Az IoT-re leselkedő veszélyek között azonban más kártékony kódok is vannak, például a Gafgyt, a Tsunami és a Dnsamp [10].

Disszertációmban a beágyazott IoT-eszközök biztonságát vizsgálom. Áttekintem az IoT-eszközökkel kapcsolatos veszélyeket, javaslok egy új támadási taxonómiát és kiemelek több, biztonsági szempontból fontos területet. A javasolt taxonómiámat meglévő sebezhetőségi adatokon alkalmazom és megállapítom, hogy a kártékony kódok komoly veszélyt jelentenek a beágyazott IoT-eszközökre. Emiatt két szempontból is tanulmányozom a kártékony kódok kérdését. Először a kártékony kódok klaszterezésének problémájával foglalkozom. Ez a technika lehetővé teszi, hogy elemzés során csak azokra a mintákra összpontosítsunk, amelyeket korábban még nem elemeztek, és amelyek nem hasonlítanak egyetlen ismert mintához sem. Másodszor a rejtőzködő kártékony kódok problémájával foglalkozom. Létezik a kártékony kódoknak egy olyan fajtája, amely csak akkor hajt végre rosszindulatú műveleteket, ha speciális bemeneteket, pl. parancsokat, kap a támadótól. Ez a trigger-alapú viselkedés, ami nagy kihívást jelent elemzés során. A kihívás leküzdésére javasolok egy új elemzési módszert, amely segít a rosszindulatú viselkedést védő környezeti feltételeket automatikus kinyerni. Végezetül javasolok egy új működési módot, amit RoViM-nak hívok, és amely lehetővé teszi a beágyazott IoT-eszközök számára, hogy periodikusan visszaállítsák önmagukat egy ismert kompromittálás-mentes állapotban, miközben funkcionalitásuk/szolgáltatásaik folyamatosan elérhetőek maradnak.

2. Új eredmények

Ebben a fejezetben foglalom össze a disszertációban részletesen kifejtett eredményeimet.

2.1. A beágyazott IoT-eszközök fenyegetettsége

A támadók képességeinek áttekintése és megértése, vagyis az ellenség megismerése előfeltétele a beágyazott IoT-rendszerek biztonságtechnikájának. A biztonsági elemzésnek, valamint a biztonságos tervezésnek és fejlesztésnek is figyelembe kell vennie a fenyegetések teljes spektrumát a biztonsági követelményeket azonosításához, valamint ahhoz, hogy a követelmények által támasztott korlátok között fejleszthessünk és alkalmazzunk különböző biztonsági mechanizmusokat. A fenyegetések megértéséhez meg kell határozni a sikeres támadások fő okait, a támadások közös jellemzőit és a fő sebezhetőségeket.

1. TÉZIS: *Áttekintettem a beágyazott IoT-eszközökre leselkedő veszélyeket, és új támadási taxonómiát javasoltam [C1] a gyakori támadások azonosítására és osztályozására. A javasolt támadási taxonómiát a CVE adatbázis releváns rekordjain értékeltem ki [C2][2], és kiemeltem a beágyazott eszközök biztonságának számos fontos szempontját.*

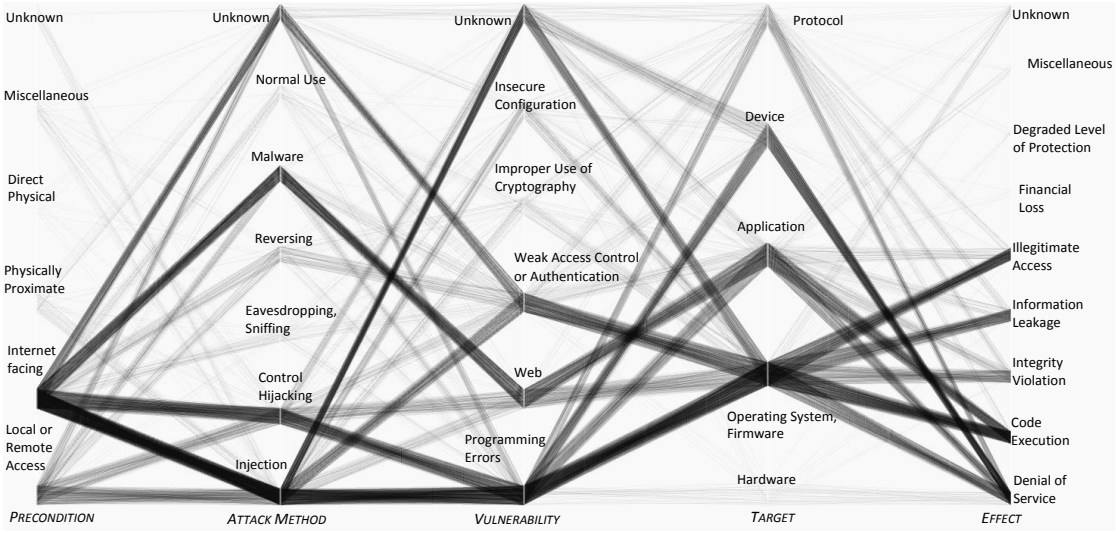
A kutatók személyes érdeklődései, a biztonsági tesztek költségei, valamint az eladó vagy eszköz tulajdonos által kikényszerített titoktartási nyilatkozatok miatt a nyilvánosságra kerülő támadások csak a teljes fenyegetettség töredékét tükrözik. Ahhoz, hogy átfogó képet kapjunk, a beágyazott rendszerekkel kapcsolatos sebezhetőségeket kell vizsgálnunk. A fő információforrás ezen a területen a Common Vulnerabilities and Exposures (CVE) adatbázis[3], amely a biztonsági rések legátfogóbb összesítése. A CVE adatbázis minden rekordjához egyedi azonosító tartozik, így megoszthatók a sebezhetőségi információk különböző szervezetek között. Kutatásom idején (2015) az adatbázis több mint 60 000 bejegyzést tartalmazott, amelyek nem mind kapcsolódtak a beágyazott eszközökhöz. Számos heurisztikát alkalmaztam, hogy kiszűrjem és kiválasszam a releváns rekordokat, majd manuálisan elemeztem ezeket. Az elemzés eredménye a támadási taxonómia alapjául szolgáló osztályozási kritériumrendszer volt.

A CVE rekordok és a támadási taxonómiákkal kapcsolatos korábbi munkák alapján 5 dimenziót határoztam meg, amelyek mentén a beágyazott rendszerek elleni támadások besorolhatók: (1) előfeltétel, (2) sebezhetőség, (3) cél, (4) támadási módszer, és (5) a támadás hatása. Az előfeltétel dimenzió olyan lehetséges feltételeket tartalmaz, amelyek teljesülése szükséges ahhoz, hogy a támadó képes legyen végrehajtani a támadást. A sérülékenységi dimenzió különböző típusú sebezhetőségeket tartalmaz, amelyeket a támadó kihasználhat. A cél dimenzió a rendszer-architektúra azon meghatározott rétegét vagy magát az eszközt adja meg, amit céloz a támadó. A támadási módszer dimenzió a sérülékenységek kihasználásához használatos technikákat tartalmaz. A hatás dimenzió a támadás lehetséges hatásait tartalmazza.

A javasolt támadási taxonómiát 3 826 darab beágyazott rendszerekkel kapcsolatos CVE rekordon értékeltem ki. Az 1. ábra párhuzamos koordináta diagramon mutatja be az eredményeket. A diagramon látható minden út egy egyedi CVE rekordból származó támadási forgatókönyv, az egyes dimenziókban érintett kategóriák pedig a támadás különböző aspektusait mutatják. Minél vastagabb egy-egy útvonal, annál több CVE rekord említi azt lehetséges forgatókönyvként. Tekintettel a gép-gép kommunikáció legújabb trendjeire és a beágyazott IoT-eszközök növekvő számára, arra számítok, hogy az internetkapcsolattal rendelkező eszközök fogják továbbra is a legtöbb támadást elszenvedni. Ezek a támadások különböző formákban fordulhatnak elő; az 1. ábra alapján azonban a kártékony kódok és a sérülékenységek kihasználásai adják a leggyakoribb megközelítéseket, amelyekkel foglalkozni kell.

[2]Mindkét cikk esetében Zhendong Ma felügyelte az AIT Austrian Institute of Technology GmbH. számára végzett munkámat.

[3]https://cve.mitre.org/ (Utolsó látogatás: 2020.01.08.)



1. ábra. Beágyazott IoT-eszközökkel kapcsolatos gyakori támadási forgatókönyvek

2.2. Kártékony kódok klaszterezése bináris hasonlósági hash-ek használatával

A 2.1 fejezetben bemutatott eredmények alapján a beágyazott IoT-eszközök biztonsága jelentősen javulna, ha jobban védekeznénk a kártékony kódokkal, pl. Mirai [2], szemben. Az antivírus vállalatok gyakran alkalmaznak osztályozási módszereket [27, 26, 12] a kártékony kódok mintáinak elemzésekor. A kártékony kódokat családokba rendezik az alapján, hogy bár bináris szinten különbözhetnek egymástól, egyazon családnak a tagjai hasonló viselkedést mutatnak. Alapvetően a klaszterezés az elemzők terhelését csökkenti, lehetővé téve, hogy csak a semmihez sem hasonlító mintákra koncentráljanak.

2.1. TÉZIS: *Tanulmányoztam a kártékony kódok mintáinak klaszterezési lehetőségeit bináris hasonlóság alapon, kimondottan a TLSH [23] hasonlósági metrika alapján [C3][4]. Megmutattam, hogy a k-medoid és az OPTICS klaszterezési algoritmusok elfogadhatatlan teljesítményt nyújtanak az IoT-specifikus kártékony kódok nagy korpuszán.*

A k-medoid [15] egy particionáló algoritmus, amelyben a klaszterek központjai (a medoidok) csak érvényes adatpontok lehetnek. Az algoritmusnak van egy bemeneti paramétere, k, amely meghatározza, hogy hány klaszter lesz az algoritmus kimenetében. Az algoritmus először kiválaszt k darab medoidot, majd az összes adatpontot a legközelebbi medoidhoz tartozó klaszterbe illeszti. Egy optimális érték eléréséig ismételget ezt a két lépést. Az algoritmus jóságának mértéke $s(k)$, amely távolság alapján méri az egyes adatpontok adott klaszterekhez való hozzárendelésének nyereségét. Minél közelebb van

[4]Bak Márton implementálta az adatgyűjtést és a klaszterező algoritmust. Tamás Csongor adta meg a kártékony kód családok variánsainak detektálásához a tapasztalati TLSH küszöbértéket.

1. táblázat. Legjobb $s(k)$ értékek a különböző k-medoid klaszterkonfigurációkhoz

k	$s(k)$
318	0,962
625	0,959
232	0,955
925	0,950
396	0,948

2. táblázat. A $k = 17$ klaszterkonfiguráció statisztikái

s(k)	0,405
Maximális átmérő	1 038
Minimális átmérő	180
Átlagos átmérő	467,824
Legnagyobb klaszter mérete	1 110
Legkisebb klaszter mérete	35
Átlagos klaszter méret	573,294

a mutató 1-hez, annál jobb a klaszterezési konfiguráció.

Több szempontból is jó választásnak tűnik a k-medoid. Ez egy felügyelet nélküli algoritmus, azaz nincs szükség további adatok megadására, csak a minták közötti hasonlósági mérésekre. Ezenkívül az algoritmus csak a meglévő adatpontokat választja ki medoidként. Ez hasznos a kártékony kódok klaszterezésénél, mert így a medoidok más kártékony kód-mintákat is képviselhetnek ugyanabban a klaszterben. Ennek az algoritmusnak a hátránya, hogy a k bemenetet manuálisan kell megadni.

Mivel nem tudom, hány variáns van az adathalmazban (a gyűjtési módszertant a disszertáció 3.2. fejezeti részletezi), ezért kiszámoltam a klaszterkonfigurációkat az összes lehetséges k értékre. Mindegyik klaszterkonfigurációhoz kiszámoltam $s(k)$ metrikáját, hogy rangsorolhassam a különböző beállításokat. Amint az az 1. táblázatban látható, a kiszámított klaszterkonfigurációk legjobb $s(k)$ értékei alig különböznek, azonban a k értékek széles tartományban mozognak, így nem egyértelmű, hogy melyik konfigurációt érdemes választani. Megfigyeltem azt is, hogy több klaszterközéppontnak kicsi a többi klaszterközépponttól vett TLSH különbsége, vagyis egyes klasztereket össze lehetne vonni. Mivel a variánsok TLSH-különbsége alacsonyabb, mint 48 (ennek a levezetése a disszertációm 3.2.3. fejezetében található), a különböző klaszterek klaszterközéppontjainak a TLSH-különbsége meg kellene, hogy haladja a 48-at. Ezt a követelményt szem előtt tartva megvizsgáltam a klaszterkonfigurációkat, és megállapítottam, hogy a $k = 17$ konfiguráció adja a legjobb $s(k)$ értéket.

A $k = 17$ klaszterkonfiguráció statisztikái a 2 táblázatban láthatók. Ebben a konfigurációban a számított klaszterátmérők 180 és 1 038 között mozognak, az átlag 468. A TLSH hasonlósági pontszámon alapuló kártékony kódok klaszterezése esetében a klaszterátmérő a legnagyobb TLSH-különbségként értelmezendő ugyanazon klaszter bármely

két mintája között. Figyelembe véve, hogy a variánsok TLSH-különbségküszöbe 48, arra a következtetésre juthatunk, hogy az ebben a beállításban szereplő klaszterek nagyon különböző mintákat tartalmaznak; a klasztereket érdemes lenne feldarabolni.

Az OPTICS [1] ritka és sűrű régiókat azonosít egy adathalmazban. Két bemeneti paramétere van: ϵ_{max} és $minPts$. Az ϵ azon terület sugarát írja le, amiben legalább $minPts$ darab adatpontnak kell lennie. Az algoritmus dinamikusan kiszámítja azt az $\epsilon \leq \epsilon_{max}$ értékeket minden adatpontra, amivel legalább $minPts - 1$ szomszédos mintát lehet lefedni ϵ sugarú körben. Az algoritmusnak szintén bemenete egy előzőleg kiszámított távolságmátrix. Az OPTICS rendelkezik beépített klaszterezési algoritmussal, ez a ξ, amely az ϵ-értékek hirtelen változásai alapján klaszterezi az adatpontokat. Ez a fajta sűrűség-alapú klaszterezés kedvező választásnak tűnik az én esetemben, mivel az adathalmazomban vannak kártékony kód családok csak néhány mintával, de vannak több ezer mintát tartalmazó családok is.

Az ϵ_{max} felső határát az adathalmaz maximális TLSH-különbsége adja. A $minPts$ paraméter megválasztása azonban kihívás az adathalmaz belső szerkezetének ismerete nélkül. Ezen ismeretek megszerzése érdekében az OPTICS-ot különböző paraméterbeállításokkal futtattam: az ϵ_{max} paramétert 40, 50, 60 és 70 értékekre állítottam, míg a $minPts$ paraméter 1, 2, 5, 10, 20, 40, 50, 70, 100, 150 és 200 között vett fel értékeket.

Az eredményül kapott klaszterkonfigurációk azonban nem voltak kielégítőek, mivel minden konfigurációban magas volt a besorolatlan minták száma. Az eredmények alapján a különböző ϵ_{max} értékek nem befolyásolják ezt a tulajdonságot: a $minPts$ fixen 2-re állítása mellett $\epsilon_{max} = 40$-nél 1 800 darab osztályozatlan mintát kaptam, míg $\epsilon_{max} = 70$ esetén 1 721-et. Minél nagyobbra emeltem a $minPts$ értéket, annál több lett az osztályozatlan minta. A $\epsilon_{max} = 70$, $minPts = 200$ konfiguráció 6 934 osztályozatlan mintát eredményezett, ami az adathalmaz 68%-ja. Ilyen nagy számú osztályozatlan minta hátrányos a kártékony kódok klaszterezésénél, mert sok minta további elemzést igényelne.

2.2. TÉZIS: *A k-medoid és az OPTICS elfogadhatatlan teljesítménye miatt új kártékony kód klaszterező algoritmust javasoltam [C3], amiről megmutattam, hogy jobb teljesítményt ér el mind a k-medoidnál, mind az OPTICS-nál.*

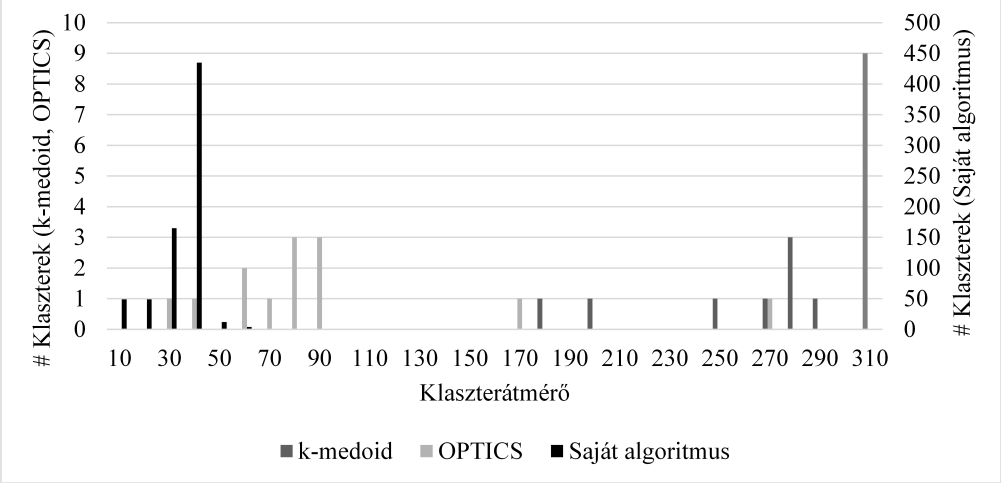
A javasolt klaszterezési algoritmusomat a következő követelményeket szem előtt tartva fejlesztettem ki. Először is, a mintákat bináris hasonlóságuk alapján kell klaszterezni, ami hasonlóságot TLSH-különbségük fejez ki. Másodszor, még a legkisebb klasztereket is meg kell találni egy változó sűrűségű adathalmazban. A bemeneti adatsor tartalmazhat egyelemű klasztereket, azaz minden más mintától különböző mintákat; ezeket azonban nem szabad zajként kezelni, mert ezek a legérdekesebb minták a kártékony kódok elemzése során. A kapott algoritmust a disszertációm 3.4. fejezete tárgyalja.

A javasolt klaszterezési algoritmus hatékonyságának értékeléséhez összehasonlítottam azt mind a k-medoid, mind az OPTICS eredményeivel. Az értékelés során figyelembe vettem a klaszterátmérőket, a létrehozott egyelemű klaszterek számát, valamint két új jósági mértéket.

A létrehozott klaszterek és az egyelemű klaszterek száma a 3. táblázatban látható. A k-medoid és az OPTICS egyaránt lényegesen kevesebb klasztert hoz létre, mint az én

3. táblázat. Klaszterező módszerek összehasonlítása

	k-medoid	OPTICS	Saját algoritmus
Klaszterek száma	17	13	392
Egyelemű klaszterek száma	0	6 058	353



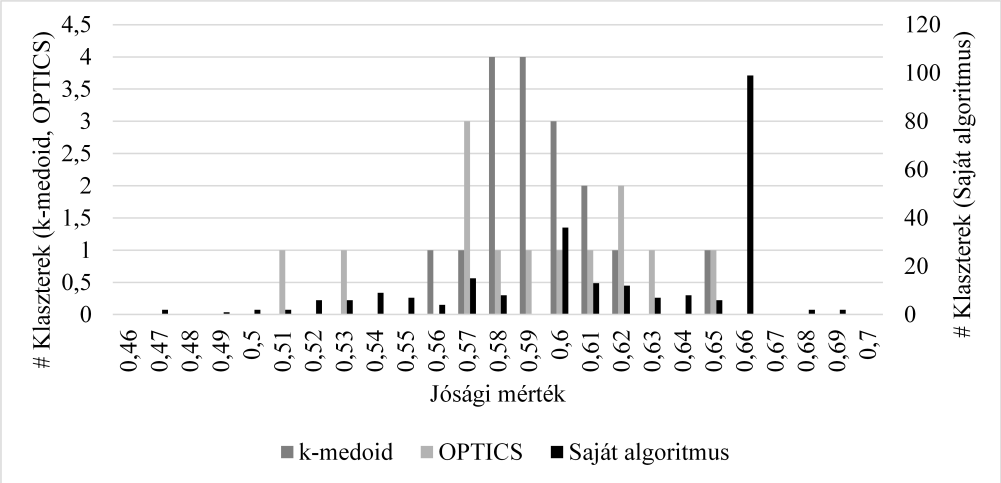
2. ábra. Különböző klaszterező algoritmusok által generált klaszterátmérők

algoritmusom, ami jobban megfelel az adathalmazban található kártékony kód-családok számának. Az én algoritmusom 745 klasztert generál, amelyekből 353 egyelemű, ez elenyésző mennyiség az OPTICS teljesítményéhez képest.

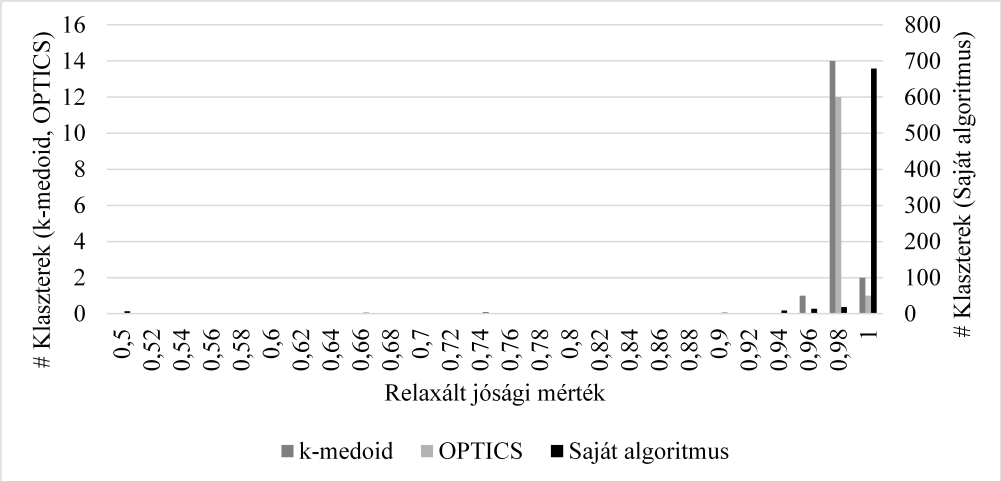
Mindhárom algoritmus esetében a klaszterkonfigurációk átmérőit a 2. ábra mutatja. Mivel az én algoritmusom sokkal több klasztert hozott létre, mint a k-medoid és az OPTICS, ezért a klaszterek számát másik skálán ábrázolom. A disszertációmban részletezett kísérletek azt mutatták, hogy a kártékony kód-családok variánsainak kimutatásához a klaszterátmérőnek 48 alatt kell lennie. A 2. ábra azt mutatja, hogy mind a k-medoid, mind az OPTICS klaszterátmérője túl nagy a variánsok jelöléséhez. Az én algoritmusom klaszterkonfigurációja azonban sokkal közelebb van ehhez a küszöbhöz, mivel a klasztereim 93,69%-a 50 alatti átmérővel rendelkezik. Vagyis a klasztereim nagyobb valószínűséggel képviselik a kártékony kódok variánsait.

Az első jósági mérték, amit bemutatok, megmutatja, hogy egy klaszter mennyire „tiszta”, vagyis a klaszter mintáinak mekkora hányada származik a klaszter legnagyobb családjából. Ez a mérőszám csak a többelemű klaszterek esetében használható, mivel a csak egyetlen mintát tartalmazó klaszterek automatikusan elérik az 1 értéket. A 3. ábra mutatja az algoritmusok teljesítményét ezen jósági mérték szerint. A k-medoid és az OPTICS klaszterkonfigurációi tipikusan 0,56 és 0,63 közötti arányt érnek el. Ezzel szemben az én algoritmusom 392 többelemű klasztere közül 185 esetében az arány meghaladja a 0,6-ot, amelyek közül 103 mind az OPTICS-ot, mind a k-medoidot felülmúlja.

Azt is figyelembe kell vennünk, hogy a kártékony kódok családjai megosztják egy-



3. ábra. Különböző klaszterező algoritmusok jósági mértékei



4. ábra. Különböző klaszterező algoritmusok relaxált jósági mértékei

mással és/vagy másolják egymástól a funkciókat. Következtetésképp a jósági mértékem relaxációja nemcsak azt a családot veszi figyelembe, amelynek a legtöbb mintája van egy adott klaszterben, hanem azokat a családokat is, amelyekkel ismert, hogy osszoznak bizonyos jellemzőkben. Az egyelemű klasztereket is figyelembe vehetjük így, mivel a többelemű klaszterek is esélyesek az 1 arány elérésére. Az összehasonlított algoritmusok által elért relaxált jósági arányok mértékét a 4. ábra mutatja. Az ábrán látható, hogy annak ellenére, hogy mind az OPTICS, mind a k-medoid elérte a 0,95 feletti arányokat, az én algoritmusom felülmúlja mindkettőt, és szinte mindegyik klaszter eléri az 1-es mértéket. Azonban az algoritmusom olyan klasztereket is előállít, amelyek relaxált jósági mértéke jóval alacsonyabb a k-medoid és az OPTICS által elért arányoknál. A probléma vizsgálata során rossz antivírus címkékre utaló jeleket találtam.

2.3. Kártékony kódok környezeti követelményeinek kinyerése

A kártékony kódok klaszterezésének vannak olyan megközelítései, amelyek futtatás során megfigyelhető jellemzőket használnak. A támadók azonban olyan kártékony kódokat fejlesztettek, amiknek a dokumentálatlan, potenciálisan rosszindulatú viselkedése csak akkor figyelhető meg, ha bizonyos feltételek teljesülnek, például speciális bemenet érkezik. Ez a *trigger-alapú viselkedés*, az ilyen bemeneteket pedig *trigger bemenetnek* hívjuk. A programba kódolt kritériumok szemantikailag sokféle külső követelményt támaszthatnak, pl. meghatározott rendszeridő vagy hely, speciális bemeneti szöveg vagy fogadott üzenet. A kártékony kódok úgy is elkerülhetik a mélyreható elemzést, hogy megvizsgálják környezetüket és leállítják rosszindulatú tevékenységeiket, ha elemzésre utaló jeleket észlelnek[5]. Trigger-alapú viselkedésre másik példa a backdoor, amely a beágyazott firmware-ek esetén elterjedt [9]: ha egy adott karakterlánc a bemenet, akkor a hozzáférés engedélyezett.

A korábbi munkák [5, 11, 17] megmutatták, hogy szimbolikus végrehajtással [3, 24] lehetséges kinyerni a rejtett viselkedéseket védő, környezettel kapcsolatos feltételeket. Ezt a technikát eredetileg a szoftvertesztelés automatizálására fejlesztették ki: képes az elemzett végrehajtási útvonalakhoz olyan teszteseteket generálni, amelyekkel követni lehet az elemzett útvonalat végrehajtás során. A trigger-alapú viselkedést védő feltételek kinyeréséhez elemeznünk kell a program és környezete kölcsönhatását, a környezetnek a program viselkedésére gyakorolt hatását. Ha a környezetből származó adatokat szimbolikus változókkal helyettesítjük, akkor a szimbolikus végrehajtás képes elemezni ezt az interakciót, így kinyerhetjük a programba kódolt feltételeket. A feltételek megoldásával konkrét értékeket kapunk, amelyek felhasználhatók további vizsgálatokhoz. A szimbolikus végrehajtásnak azonban van limitációja: minél több a szimbolikus változó az elemzésben, annál több végrehajtási útvonalat kell elemezni, ami *állapottér robbanáshoz* vezet. A korábbi munkák ezért a lehetséges feltételtípusoknak csak egy részhalmazát vették figyelembe.

[5] https://www.fireeye.com/blog/threat-research/2011/01/the-dead-giveaways-of-vm-aware-malware.html Utolsó látogatás: 2020.01.08.

3.1. TÉZIS: *Általános megközelítést javasoltam a rejtőzködő kártékony programok környezeti feltételeinek kinyerésére. A javasolt megközelítésem képes minden külső adatforrást trigger bemeneti típusnak tekinteni. Igazoltam az új megközelítés forráskód-szintű alkalmazhatóságát valós, nyílt forráskódú szoftverek felhasználásával [C4][6]. Az eredmények azt mutatják, hogy sikeres szimbolikus végrehajtás esetén a rejtett viselkedést védő környezeti feltételek kinyerhetők a programokból. Az állapottér robbanás azonban továbbra is kihívás.*

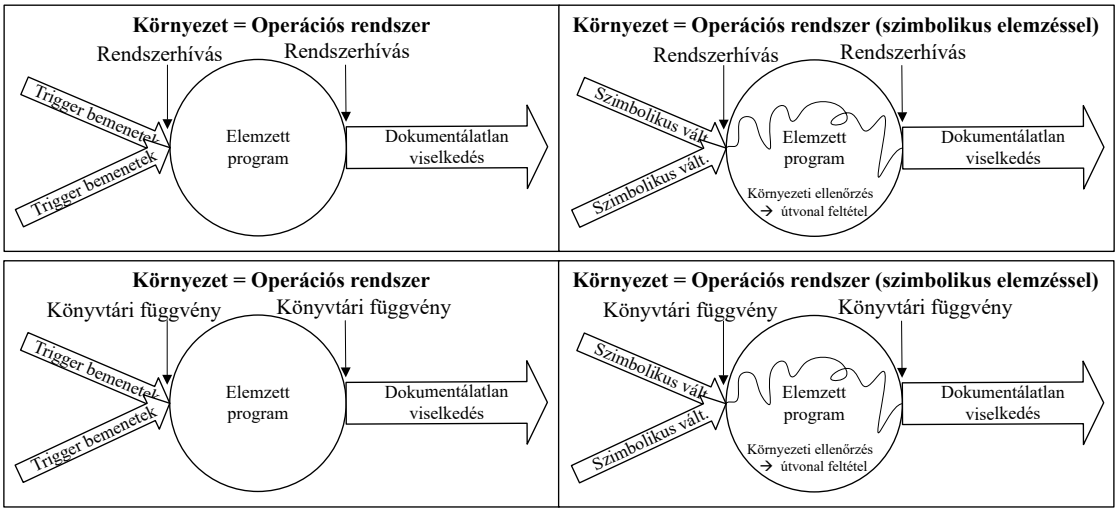
Javítottam a meglévő munkákon azáltal, hogy automatikusan figyelembe vettem az összes lehetséges feltételtípust. Annak érdekében, hogy a nagyobb számú szimbolikus változó és az ebből eredő állapottér robbanási probléma kihívásait megoldhassam, a megközelítésem rosszindulatú viselkedések jól meghatározott eseteire összpontosít. A módszerem a rosszindulatú viselkedés specifikációját rendszerhívások és hozzájuk kapcsolt predikátumok sorozataként várja: $(s_0, p_0), (s_1, p_1), \ldots, (s_n, p_n)$. A tézisfüzetben erre a sorozatra a rosszindulatú viselkedés *célzott rendszerhívás-mintája* néven hivatkozom.

A rosszindulatú viselkedések modellezése rendszerhívások sorozataként egy széles körben alkalmazott megközelítés a kártékony kódok elemzésének területén [19, 28, 14, 8, 13, 18], melynek számos oka van. Először is, a rendszerhívások jelentik az elsődleges kommunikációs csatornát a programok és az operációs rendszer között. Sok, a program végrehajtásához szükséges, funkciót az operációs rendszer szolgáltatásként biztosít, amiket rendszerhívásokon keresztül lehet elérni. Másodszor, az egyes rendszerhívások szemantikája dokumentált és elérhető. Azonban a rendszerhívások önmagukban gyakran nem elegendőek egy adott viselkedés kellő részletességgel történő leírására [7]. Ezért a felhasználó a predikátumok segítségével további követelményeket támaszthat a meghívott rendszerhívásokra. Az ilyen követelmények között szerepelhet például annak meghatározása, hogy milyen argumentumokkal hívódjon meg egy-egy rendszerhívás, mi legyen a végrehajtási állapot memóriájának és/vagy regisztereinek tartalma, stb. Bár a célzott rendszerhívás-minták megadása szakértői ismereteket igényel, azokat elég csak egyszer meghatározni, és a későbbiekben újra felhasználhatók. E tekintetben hasonlóak a kártékony kódok elemzésével kapcsolatos egyéb tudásbázisokhoz, mint amilyenek például a YARA szabályok[7]. Ezenkívül egyes alacsony szintű célzott rendszerhívás-minták kombinálhatók magasabb szintű viselkedési specifikációk felépítéséhez, hasonlóan Lorenzo *et al.* [22] munkájához.

A megközelítés alapötletét a 5. ábra mutatja. Feltételezem, hogy az elemzett program determinisztikus és kölcsönhatásba lép a környezettel, akár könyvtárak, akár az operációs rendszer és annak API-ja (rendszerhívások) útján. Valóság lefutás során a program több hívást is indítana, és a visszatérési értékek egy részhalmazát vetné össze a logikájába kódolt feltételekkel. A program csak akkor hajtja végre a potenciálisan rosszindulatú viselkedést, ha az összehasonlítás(ok) eredményes(ek). Ezen interakció elemzéséhez a külső forrásokból adatot visszaadó hívásoknak a visszatérési értékeit friss

[6] Thorsten Tarrach felügyelte az AIT Austrian Institute of Technology GmbH. számára végzett munkámat.

[7] https://virustotal.github.io/yara/ (Utolsó látogatás: 2021.03.05.)



5. ábra. Szimbolikus végrehajtás használata a rosszindulatú viselkedést védő környezeti feltételek kinyeréséhez

4. táblázat. Rejtett rosszindulatú viselkedés feltárásával kapcsolatos kísérletek eredményei nyílt forráskódú szoftverek esetén

Minta neve	**Elemzett útvonalak**	**Generált tesztesetek**	**Detekció**
cd00r	1 299	5 (1 kiemelt)	Igen (1/1)
giardia	48	4 (1 kiemelt)	Igen (1/1)
osx-ping-backdoor	212 754	122 (2 kiemelt)	Igen (1/2)
portknockd	11 902 399	1	Nem
portknocking	39 077	8	Nem

szimbolikus változókkal kell helyettesíteni.

Ezt a módszert a GNU és LLVM szoftver-ökoszisztéma felhasználásával implementáltam prototípusként. A prototípusom automatikusan generál szimbolikus összefoglaló függvényeket függvénynevek és szemantikai adatok listája alapján, amely szimbolikus összefoglaló függvények szimbolikus változókat vezetnek be. A prototípus a KLEE [6] szoftvert használja a szimbolikus végrehajtáshoz. Implementációm a `klee_assert()` függvény segítségével jelzi a KLEE-nek, hogy kiemelt teszteseteket generáljon potenciálisan rosszindulatú utasítások elérésekor.

A megközelítés kiértékeléséhez nyílt forráskódú szoftvereket gyűjtöttem a GitHubról a „backdoor”, „logic bomb”, „time bomb” és „portknock” kulcsszavakkal. Mindegyik szoftvert C-ben fejlesztették és valamilyen trigger-alapú viselkedést implementáltak bennük. A kiértékeléshez egy virtuális gépet használtam 4 CPU-val és 10 GB memóriával. A virtuális gépen Ubuntu 14.04.5. LTS-t futott. A KLEE maximum 8 GB memóriát használhatott, és az alapértelmezett útvonal választási stratégiáját használtam.

Az eredményeket a 4. táblázat mutatja. Sok útvonalat elemzett a KLEE, de úgy

konfiguráltam, hogy csak az új útvonalakhoz generáljon tesztesetet. Ezért alacsony a generált tesztesetek száma. Ötből három esetben a KLEE tudott olyan tesztesetet generálni, ami kiváltotta a rosszindulatú viselkedést. A fennmaradó esetekben a KLEE limitációi sikertelen eredményt hoztak. Az eredmények alapján az állapottér robbanás továbbra is kihívás a megközelítésem számára.

3.2. TÉZIS: *Az állapottér robbanás leküzdéséhez egy új elemzési módszert javasoltam, ami irányított szimbolikus végrehajtással [21] irányítja az elemzést kiválasztott programpontok felé, például a binárisokban előforduló potenciálisan rosszindulatú viselkedések felé. A javasolt elemzési módszert az angr [25] keretrendszerben implementáltam, és mind mesterséges, mind valós mintákon kiértékeltem [C5][8]. Kísérleteim bebizonyították, hogy ki lehet nyerni az adott programpontok futtatását védő környezeti feltételeket. Az elemzés elfogadható teljesítményt nyújt, figyelembe véve az elemzett minták összetettségét és megközelítésem általános jellegét.*

A javasolt elemzési módszerem három technikából áll:

1. egy három szintből álló útvonal-választási stratégia a rendelkezésre álló végrehajtási utak rangsorolásához,

2. szimbolikus összefoglaló függvények, amelyek leírják a meghívott rendszerhívások viselkedését és bevezetik a környezetből származó adatok modelljeit szimbolikus változóként, valamint

3. egy mechanizmus, amivel automatizáltan kiválaszthatóak azok az megcélzott programpontok, amelyek segítik a végrehajtási útvonal előrehaladását a célzott rendszerhívásminta elemei között.

Az elemzés a program belépési pontjától indul, és amíg vannak elemzendő végrehajtási állapotok, az útvonal-választási stratégia szerint kiválasztjuk a legígéretesebb állapotokat. A kiválasztott állapot(oka)t szimbolikusan elemezzük kevert konkrét és szimbolikus végrehajtással. A környezet modellezéséhez egyedi szimbolikus összefoglaló függvényeket kell megadni. A javasolt útvonal-választási stratégia pszeudokódja a disszertációmban 2. algoritmusként található meg. Az algoritmus részleteit a 4.4. fejezet tárgyalja.

angrben implementáltam a módszer prototípusát és két kártékony kód mintán értékeltem ki a teljesítményét. Mindkét minta számos kihívást jelent az elemzés számára. Először is, az előfeltételezéseim és azok megvalósítása miatt nagyszámú végrehajtási útvonal áll rendelkezésre elemzés közben a következő okok miatt:

1. Környezeti adatok: A minták felhasználják a rendszeridőt és folyamatazonosítókat, valamint hálózati kommunikációt folytatnak. Mivel feltételezem, hogy nincs előzetes ismeretünk a funkcionalitásról, elemzésem során az összes bemenetet szimbolikus változók segítségével kell elemeznem, ami sok elágazáshoz vezet.

[8] Thorsten Tarrach felügyelte az AIT Austrian Institute of Technology GmbH. számára végzett munkámat.

2. Sztring-kezelés: A minták a hálózati bemenetet a szokásos libc függvények, például `strlen` és `strcasecmp` segítségével dolgozzák fel. Ezek a függvények általában karakterenként iterálnak végig a sztring felett. Mivel a bemeneteiket a kernel adja vissza, elemzésem során az egyes karaktereket szimbolikus változóknak kell tekintenünk. Ezek a ciklusok köztudottan hozzájárulnak az állapottér robbanáshoz.

3. Végtelen ciklus: A minták végtelen ciklusban futnak, folyamatosan figyelve a parancs- és vezérlőszerver (C&C szerver) üzeneteit, kommunikációs hiba esetén pedig megpróbálnak újracsatlakozni. Vagyis, az összes végrehajtási útvonal feltárása nem kivitelezhető véges idő alatt.

A mesterséges mintám többféle a trigger-alapú viselkedést is tartalmaz. Először ellenőrzi, hogy virtuális környezetben hajtják-e végre, és kilép, ha úgy ítéli meg. Ha nem észlel virtuális környezetet, a kártékony kódom csatlakozik a belekódolt parancs- és vezérlőszerverhez, és kiszivárogtatja a gazdagép számos paraméterét, beleértve az operációs rendszer és a kernel verzióit, a teljes és a rendelkezésre álló memóriát, valamint a futó folyamatok számát. Ezenkívül fogadja a C&C szerver parancsait. A parancsok hatására TCP PUSH + ACK típusú szolgáltatás-megtagadásos támadást indíthat a megadott cél ellen, vagy leállíthatja magát és az összes elindított gyerekfolyamatot.

Egy virtualizált környezetet észlelő végrehajtási útvonalat kb. 7 perc (437,17 másodperc) alatt és 5,96 GB RAM felhasználásával talált meg az elemzésem. A megfelelő rendszerhívások eléréshez az elemzés összesen 71 végrehajtási utat generált, mielőtt talált volna egy olyat, amely észleli a virtuális környezetet és kilép. Ennek a végrehajtási útvonalnak az útvonal feltételei olyan korlátozásokat tartalmaznak, amelyek megmutatják a virtuális környezetet kódoló szükséges ASCII értékeket: `„Q"`, `„E"`, `„M"`, `„U"`.

A rendszerpatamétereket kiszivárogtató végrehajtási útvonal keresése kb. 9 perc (568,09 másodperc) alatt és 5,8 GB RAM felhasználásával történt. 4 olyan végrehajtási útvonal is volt, amelyekhez az útvonal választási stratégiám azonos prioritásokat rendelt; elemzésük azonos iterációban tárta fel a keresett viselkedést.

Elemzésem során szolgáltatás-megtagadásos támadást indító végrehajtási útvonalat kb. 5 óra (299 perc) alatt és 41,4 GB RAM felhasználásával találtam. Az útvonal feltételei megmutatják a szükséges trigger bemenetet, amit az 5. táblázat mutat. Manuálisan megadtam az SMT megoldó által visszaadott konkrét bemenetet a mintámnak, és megállapítottam, hogy ez valóban elindítja a támadást.

A kiértékelés során egy nyilvánosan elérhető Kaiten mintát[9] is használtam. A szolgáltatás-megtagadásos támadásokat indító egyik függvényt (`tsunami` a forráskódban) választottam megcélzott viselkedésként.

A 6. táblázat mutatja a prototípus implementáció teljesítményét a Kaiten bináris mintán. Egyetlen futás végrehajtási ideje négy összetevőből áll:

1. vezérlési folyam gráf generálása és kiegészítése,

2. végrehajtási útvonalak szimulációja,

[9] https://packetstormsecurity.com/files/25575/kaiten.c.html (Utolsó látogatás: 2021.01.08.)

5. táblázat. A szolgáltatás-megtagadásos támadást kiváltó szimbolikus karakterek megoldásai

Karakter indexe	Kielégítő megoldások
0	`d` vagy `D`
1	`o` vagy `O`
2	`s` vagy `S`
3	SZÓKÖZ
4	bármilyen karakter
5	bármilyen karakter
6	bármilyen karakter
7	bármilyen karakter
8	SZÓKÖZ
9	lezáró `NULL`

6. táblázat. Valós kártékony kódon mért futási teljesítmény

Fázis	**Futási idő (hh:mm:ss)**
Vezérlési folyam gráf generálása és kiegészítése	0:10:42
Végrehajtási utak szimulációja	19:08:54
Legrövidebb út számítása	8:05:44
Egyéb felügyeleti feladatok	5:05:11

3. végrehajtási állapotok rangsorolása legrövidebb út alapján, és

4. egyéb felügyeleti feladatok, pl. naplózás, a cél elérésének ellenőrzése, stb.

Az elemzésem végrehajtási ideje 32,5 óra volt. Az idő nagy részében a végrehajtási utak szimulációja és a legrövidebb utak számítása történt.

A végrehajtási utak elemzési idejét a minta logikája befolyásolta. A tesztek során olyan címeket kellett elemezni, amelyek órákba teltek kevert konkrét és szimbolikus végrehajtással. Ezek a címek a libc részét képezik, beleértve a `rand`-ot és több sztring-manipuláló függvényt, amelyek elemzéséhez összetett szimbolikus változókkal kellett számításokat végzeni. A `rand`-ot a minta arra használja, hogy véletlenszerű, egy karakterből álló sztringeket állítson elő az C&C szerverrel történő kommunikációhoz. Bár ezeket a sztringeket muszáj elemezni a kívánt rendszerhívás eléréséhez, értékük nem számít: a hálózati bemenetet modellező szimbolikus sztring vagy megegyezik velük, vagy nem. Ezért a `rand`-ot lecseréltem az angr beépített szimbolikus összefoglaló függvényére, és egy friss szimbolikus változót használtam az eredményének modellezésére. A sztring-manipulációk eredményei azonban közvetlenül hozzájárulnak a célzott viselkedés felé vezető végrehajtási útvonalhoz: befolyásolják, hogy a hálózati bemenetet ábrázoló szimbolikus sztring milyen hosszú, és milyen feltételek vannak a karakterekkel szemben. Emiatt nem befolyásoltam ezek szimulációját és megelégedtem a növekedett elemzési idővel.

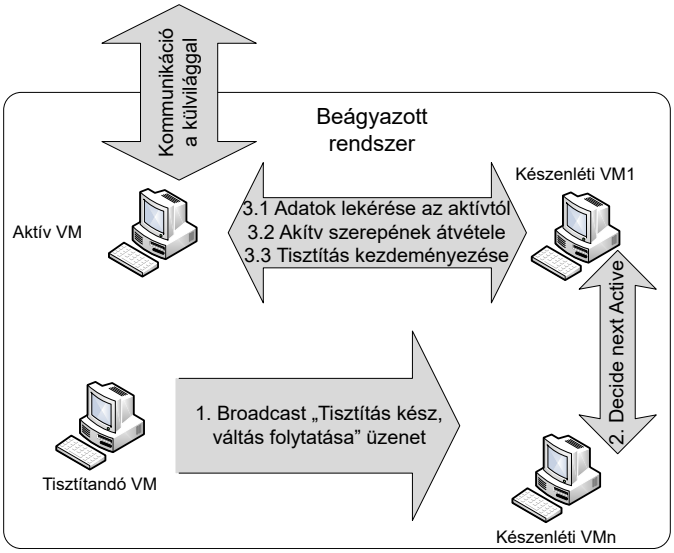
Az összes előbb említett kísérletet egy kétszer 10 magos, 2,8 GHz-en működő Xeon E5-2680 CPU-val rendelkező géppel végeztem. A gépben 378 GB RAM állt rendelkezésre. Az eredmények értékelésénél figyelembe kell venni, hogy az angr nem többszálú, és csak egyetlen magot használ. Az angr-t 330 GB memóriára korlátoztam.

A megcélzott programpontokat (vagy azok sorozatát) elérő végrehajtási állapotok útvonal-feltételeinek értelmezése változó nehézségű. Az olyan korlátozások, mint például a `<Bool socket_retval_23127_32 == 0x3>`, könnyen értelmezhetők a modellezett rendszerhívás szemantikájának ismeretében. A `socket_retval_23127_32` egy olyan szimbolikus változó, amely a `socket` rendszerhívás visszatérési értékét modellezi. A két számot az angr teszi a változó neve mögé: az első egy egyedi azonosító, míg a második a változó hossza bitben. A `socket` visszatérési értéke siker esetén egy fájlleíró (pozitív egész szám), hiba esetén pedig -1. Mivel az egyenlet jobb oldala pozitív, arra következtethetünk, hogy a `socket` rendszerhívásnak sikeresen kellett lefutni.

Más feltételek emberi értelmezése azonban összetettségük miatt meglehetősen nagy kihívást jelent. Például a Kaiten minta az egy socketből kiolvasható karakterek maximális számára a 4 096-os felső korlátot szabja. Mivel a `recv` szimbolikus összefoglaló függvényem felső határa a visszaadandó karakterek számára 10 (állapottér robbantást elkerülendő), a hívás egy legfeljebb 10 karakterből álló sztringet ad vissza. A minta ezután több sztring-manipulációs függvényt hív meg, amelyek karakterenként iterálnak végig a bemenetükön. Az ezt implementáló bináris utasítások sok esetben feltételesek, ami azt jelenti, hogy a valós életben a CPU csak szükség esetén hajtaná végre őket. Az elemzés során azonban egyik operandus szimbolikus karakter, ezért nem hagyhatók ki. Ezért, ha lehetséges, az eredményeket az útvonal-feltételek között `If-Then-Else` struktúrák kódolják: ha a feltétel igaznak bizonyul, akkor az eredmény a `Then` érték, máskülönben az `Else` érték. Ezek a struktúrák egymásba ágyazhatók, ami olyan feltételekhez vezet, amelyek manuális értékelése időigényes. Ilyen esetekben egy feltételrendszer-megoldó használható a kielégítő érték-hozzárendelések kiszámítására, konkrét bemeneteket adva a célzott viselkedés kiváltására. A bemeneti sztringekkel kapcsolatos feltételek értelmezésének másik kihívása az, hogy a feltétel maga nem tartalmaz karaktereket. Ehelyett az (egyenlőség) ellenőrzések a karakterek numerikus ASCII értékeit használják. A szimbolikus változók elnevezési konvenciója azonban megmondja az elemzőknek, hogy melyik rendszerhívásnál vezettek be egy változót az elemzésbe. Ezáltal az elemző szemantikus információt nyer, hogy a szimbolikus változó értéke karakterként értelmezendő.

2.4. Proaktív biztonság beágyazott IoT eszközök számára

A veszélyforrásokkal kapcsolatos eredményeim azt mutatják, hogy a kártékony kódokkal fertőzött beágyazott IoT-eszközök veszélyeztethetik a különböző alkalmazási területeket. Ezért ezeknek az eszközöknek meg kell felelniük biztonsági követelményeknek, amelyeket biztonsági ellenőrzésekkel és mechanizmusokkal érhetünk el. Ugyanakkor számos sebezhetőség is van, amelyekkel foglalkozni kell. Az egyes sebezhetőségek kezelése egyenként időigényes feladat, sajnos nem skálázódik jól. Ezért figyelmemet a proaktív biztonság felé fordítottam. A reaktív biztonsági megközelítéssel szemben, amelynek célja a támadások és kompromittálódások észlelése és az azokra való reagálás, a proaktív biztonsági



6. ábra. A RoViM működése magas szinten

mechanizmusok előre számítanak a támadásokra, és intézkedéseket tesznek a kompromittálódás bekövetkezésének megakadályozása érdekében.

4. TÉZIS: *Terveztem egy új működési módot, a RoViM-ot, ami lehetővé teszi a beágyazott IoT-eszközök számára, hogy időszakosan visszaállítsák magukat egy kompromittálódás-mentes állapotba [C6][10]. Formálisan verifikáltam a működési módot megvalósító eljárást* Uppaalban *[4] és elérhetőségi, élőségi és biztonsági tulajdonságokat igazoltam. Ezen kívül megvalósítottam egy IPsec [16] átjárót a RoViM működési módra építve, és lemértem annak teljesítményét. Mérési eredményeim azt mutatták, hogy az általam javasolt új működési mód nincs jelentős hatással a felhasználói élményre.*

A RoViM működési mód alapötlete, hogy a beágyazott eszközön több virtuális gépet futtatunk és a virtuális gépeket periodikusan rotáljuk. Mielőtt a rendszer magas szintű áttekintését bemutatnám, néhány definíciót kell bevezetni. A külvilággal kapcsolatban álló és azzal kommunikáló virtuális gépet, amely elvégzi a beágyazott eszköz feladatát, *aktív* virtuális gépnek nevezzük. A *készenléti* virtuális gép(ek) redundanciát biztosítanak és készenlétben állnak az aktív virtuális gép cseréjéig. Azt a készenléti virtuális gépet, amely a rotáció során aktív virtuális géppé válik, *következő aktív* virtuális gépnek nevezzük. A *tisztítandó* virtuális gép korábban aktív virtuális gépként működött, és visszaáll a potenciális kompromittálódás előtti állapotában. A gyakorlatban a kompromittálódásmentes állapot lehet egy pillanatkép, amelyet a beágyazott eszköz telepítése előtt készítenek. A virtuális gépek közötti rotáció időszakosan történik.

A 6. ábra egy rotációt mutat be az egyetlen ciklus befejezéséhez szükséges magas

[10] Zhendong Ma felügyelte az AIT Austrian Institute of Technology GmbH. számára végzett munkámat.

szintű interakciókkal. Az aktív virtuális gép és a külvilág közötti számozatlan nyíl kiemeli, hogy az aktív virtuális gép és a külvilág közötti kommunikációt nem zavarja a rotáció. A rotációt a tisztítandó virtuális gép időszakosan váltja ki, miután visszaállította kompromittálódás-mentes állapotát. Ez a virtuális gép küldi meg a rotációt kiváltó broadcast üzenet az összes készenléti virtuális gépnek, utasítva őket, hogy kezdjék meg a rotáció második szakaszát.

A második szakaszban az egyik készenléti virtuális gép lesz a következő aktív virtuális gép. A készenléti virtuális gépek számától függően két esetet kell elkülöníteni. Ha csak egy készenléti virtuális gép van (a korábbi tisztítandó virtuális gép kivételével), akkor ez a virtuális gép lesz automatikusan a következő aktív virtuális gép. Ha több készenléti virtuális gép van, akkor meg kell állapodniuk arról, hogy melyik készenléti virtuális gép legyen a következő aktív virtuális gép. Ez a probléma a jól ismert vezetőválasztási probléma [20] megoldását jelenti.

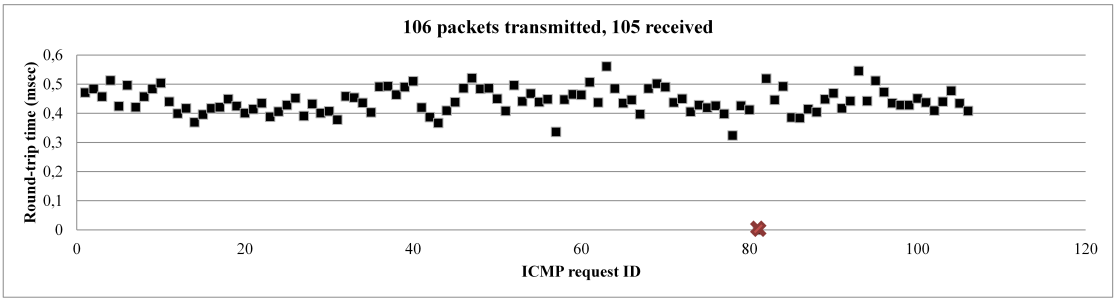
Az aktív virtuális géppé váláshoz egy háromfázisú interakcióra van szükség az aktív és a következő aktív virtuális gépek között. A három alfázis magas szintű leírását itt is bemutatom, további részleteket a disszertációm 5.3. fejezete tartalmaz.

1. A következő aktív virtuális gépnek meg kell szereznie a beágyazott eszköz feladatának ellátásához szükséges összes adatot. Mivel az aktív virtuális gép elérhető a külvilágból és kompromittálódhat, a rajta lévő adatok megsérülhetnek, vagy rosszindulatú programok telepíthetők. A tervezett rendszer kibővíthető az alkalmazás adatainak input validációjával annak érdekében, hogy ne kerülhessen kártékony tartalom más virtuális gépekre. Ezenkívül az aktív virtuális gépről származó adatok továbbítása közben az aktív virtuális gép nem változtathatja meg az alkalmazás adatait. Ellenkező esetben a következő aktív virtuális gépen futó alkalmazás és a külvilág entitásai nem lennének szinkronban. Bizonyos értelemben az alkalmazás számára meg kell állnia az időnek, de ez ellentétes lehet az alkalmazás rendelkezésre állási követelményeivel. Ezért a megvalósításnak meg kell határoznia egy határidőt, amennyi idő alatt a következő aktív virtuális gép átveheti az aktív virtuális gép helyét. Ha a szerep átvétele ezen a határidőn belül nem sikerül, akkor a rotációt meg kell szakítani.

2. A következő aktív virtuális gépnek értesítenie kell a helyi hálózat összes csomópontját, hogy az aktív virtuális gépnek szánt csomagokat a következő aktív virtuális gépnek továbbítsák.

3. A következő aktív virtuális gépnek meg kell kezdenie az aktív virtuális gép visszaállítását a kompromittálódás-mentes állapotba. Az aktív virtuális gép azonban csatlakozik a külvilághoz, és kompromittálódhat. Feltételezhetjük ezért, hogy a kompromittálódás-mentes állapotba való visszaállítás ellentétes a támadó érdekeivel. Ezért a következő aktív virtuális gépnek kényszerítenie kell a tisztítást. Ilyen tisztítási eljárás lehet például az aktív virtuális gép visszaállítása a beágyazott eszköz kihelyezése előtt készített pillanatképre.

A csomagvesztés elkerülése érdekében az interakció alatt a beérkező hálózati csomagok pufferelhetők, majd később újraküldhetők.



7. ábra. ICMP kérések és válaszok megfordulási ideje

A háromfázisú interakció eredményeképp a virtuális gépeknek olyan globális állapotot kell elérnie, miszerint vagy

- a váltás hibák nélkül történt, és a megválasztott készenléti virtuális gép átvette az aktív virtuális gép szerepét, vagy
- az interakció előtti globális állapot állt vissza hiba esetén, és az aktív virtuális gép továbbra is az aktív szerepet tölti be.

Fontos ellenőrizni, hogy a javasolt harmadik fázis követése nem vezethet-e inkonzisztens állapotú virtuális gépekhez, és hogy a fázist implementáló protokollban van-e holtpont. A kérdések megválaszolásához a javasolt protokollt formális verifikációnak vetettem alá az Uppaal [4] segítségével. A verifikáció során nem a biztonsági problémákra, hanem a protokoll funkcionális helyességének ellenőrzésére fókuszáltam. Az Uppaal bebizonyította a protokoll elérhetőségi, élőségi és biztonsági tulajdonságokkal rendelkezik. Ezen elemzés részletei a disszertációm 5.4. fejezetében találhatók.

A prototípus IPsec átjáró teljesítményét csomagvesztési és felhasználói élmény szempontokból értékeltem ki. A csomagvesztéssel kapcsolatos kísérletben ICMP kéréseket és válaszokat használtam: `ping` paranccsal kéréseket küldtem egy `kliens` virtuális gépről egy `szerver` virtuális gép felé (kettejük között IPsec-alagút biztosítja a biztonságos kapcsolatot). Eközben manuálisan indítottam egy rotációs ciklust az IPsec átjárón.

A mérés eredményeit a 7. ábra mutatja. A rotáció akkor indult el, amikor az 81-es ICMP kérés elindult a `kliens` gépről. Az egyes ICMP kérések és válaszok megfordulási ideje azt mutatja, hogy a csomagáramlás során nem lépett fel jelentős késés. Azonban a `kliens` csomagvesztést szenvedett el az 81-es ICMP kérésnél. A prototípus megvalósításának naplófájljai szerint az aktív virtuális gép nem tudta elég gyorsan feldolgozni a 81-es ICMP-kérést, ami így nem jutott el a következő aktív virtuális gépre. A mérés többszöri futtatása ugyanezt az eredményt hozta.

A második mérés a felhasználói élmény változását vizsgálta. Ehhez letöltöttem egy 500 MB-os fájlt a `szerverről` a `kliensre`. Az előző méréshez hasonlóan a RoViM-prototípus IPsec átjárón manuálisan indítottam el a rotációt. A fájl letöltésére használt HTTP protokoll a TCP-t használja, ezért a csomagfolyam betekintést nyújtott abba is, hogy a háromfázisú interakció miatti késleltetés hogyan befolyásolja a TCP-t. A `szerveren` Wiresharkkal figyeltem a csomagfolyamot és gyűjtöttem statisztikákat.

7. táblázat. A TCP csomagfolyam statisztikái rotációval és rotáció nélkül

	Rotáció nélkül	Rotációval
Átviteli idő	23,680 mp	23,684 mp
Duplicate IP address configured (172.16.5.254)	0	2
Újraküldések	55	315
Nem sorrendben érkező szegmensek	32	39

Az eredményeket a 7. táblázat tartalmazza. A `kliens` és a `szerver` közötti TCP kapcsolat nem szakadt meg a rotáció alatt. A várakozásaimnak megfelelően a rotáció késleltetést vezetett be az átvitelbe, de az átviteli idő mindössze 4 ezredmásodperccel nőtt, ami nem befolyásolja érdemben a felhasználói élményt.

A következő aktív virtuális gép hálózati interfészének IP-címe megegyezett az aktív virtuális gép hálózati interfészének IP címével. A `szerver` oldalán úgy tűnt, hogy bár az átjáró IP-címe nem változott, a MAC-címe viszont igen. Ezért a Wireshark `Duplicate IP address configured` figyelmeztetést adott. Ez a figyelmeztetés kétszer volt jelen a csomagfolyamban az IPsec átjáró rotációja miatt.

A rotáció jelentősen növelte a megismételt TCP szegmensek számát. Esetünkben a tesztkörnyezetben a hálózat átbocsátási képessége a rotációig nagyon magas volt. Ennek oka egyrészt, hogy nincs más forgalom, másrészt a tesztkörnyezet is virtuális, és a virtuális gépek egyszerűen portok a gazdagép szempontjából. A Wireshark a rotáció előtt 0,25 másodperces újraküldési időkorlátot mért. Ezután megtörtént a három alfázisból álló interakció, és hirtelen az összes bejövő csomagot pufferelte az aktív virtuális gép. A szegmensekre adott nyugták nem érkeztek meg időben, ezért a `szerver` azt hitte, hogy valamilyen hálózati hiba történt. Az ismétlésekhez összesen 6,556 ezredmásodpercre volt szükség.

A nem sorrendben érkező szegmensek számának növekedését is a rotáció okozta. Amikor a következő aktív virtuális gép felhúzta interfészeit, új csomagok is elkezdhettek áramolni, és a pufferelteket újraküldte az IPsec átjáró. A mérés során ez az előző szegmensekre vonatkozó pufferelt nyugtákat jelentette, ami miatt a szegmensáramlat sorrendje felborult.

A bemutatott eredmények alapján úgy tűnik, hogy a rotáció opcionális pufferelési funkciója csak akadályozza a TCP teljesítményét. A kísérletet megismételtem a funkció letiltásával, és ennek eredményeként a TCP-nek csak 5,113 ezredmásodpercre volt szüksége a pufferelt szegmensek megismétléséhez.

Saját publikációk listája

Konferencia és workshop cikkek

[C1] Papp, D., Ma, Z., and Buttyán, L. Embedded systems security: Towards an attack taxonomy. In *Mesterpróba 2015* (2015), pp. 29–32.

[C2] Papp, D., Ma, Z., and Buttyan, L. Embedded systems security: Threats, vulnerabilities, and attack taxonomy. In *2015 13th Annual Conference on Privacy, Security and Trust (PST)* (2015), pp. 145–152.

[C3] Bak, M., Papp, D., Tamás, Cs., and Buttyán, L. Clustering IoT malware based on binary similarity. In *6th IEEE/IFIP Workshop on Security for Emerging Distributed Network Technologies (DISSECT)* (2020), NOMS '20, pp. 1–6.

[C4] Papp, D., Buttyán, L., and Ma, Z. Towards semi-automated detection of trigger-based behavior for software security assurance. In *4th International Workshop on Software Assurance (SAW 2017)* (New York, NY, USA, 2017), ARES '17, Association for Computing Machinery.

[C5] Papp, D., Tarrach, T., and Buttyán, L. Towards detecting trigger-based behavior in binaries: Uncovering the correct environment. In *Software Engineering and Formal Methods* (Cham, 2019), P. C. Ölveczky and G. Salaün, Eds., Springer International Publishing, pp. 491–509.

[C6] Papp, D., Ma, Z., and Buttyán, L. RoViM: Rotating virtual machines for security and fault-tolerance. In *EMC2 Summit at CPS Week* (2016).

[C7] Papp, D., Kócsó, B., Holczer, T., Buttyán, L., and Bencsáth, B. ROSCO: Repository Of Signed COde. In *Virus Bulletin Conference* (Prague, Czech Republic, 2015).

[C8] Juhász, M., Papp, D., and Buttyán, L. Towards secure remote firmware update on embedded IoT devices. In *12th Conference of PhD Students in Computer Science* (2020).

[C9] Tamás, C., Papp, D., and Buttyán, L. SIMBIoTA: Similarity-based malware detection on IoT devices. In *6th International Conference on Internet of Things, Big Data and Security* (2021), IoTBDS '20.

Folyóirat cikkek

[J1] Papp, D., Tamás, K., and Buttyán, L. IoT hacking–a primer. *Infocommunications Journal 11*, 2 (2019), 2–13.

[J2] Papp, D., Zombor, M., and Buttyán, L. TEE based protection of cryptographic keys on embedded IoT devices. In *Annales Mathematicae et Informaticae*

(Special Issue on the Conference on Information Technology and Data Science) (2020). Az írás időpontjában (2021.05.08.) megjelentetési fázisban van.

[J3] Nagy, R., Németh, K., Papp, D., and Buttyán, L. Rootkit detection on embedded IoT devices. In *Acta Cybernetica* (2021). Az írás időpontjában (2021.05.08.) megjelentetési fázisban van.

Hivatkozások

[1] Ankerst, M., Breunig, M. M., Kriegel, H.-P., and Sander, J. Optics: ordering points to identify the clustering structure. In *ACM Sigmod record* (1999), vol. 28, ACM, pp. 49–60.

[2] Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M., Kumar, D., Lever, C., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K., and Zhou, Y. Understanding the Mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)* (Vancouver, BC, Aug. 2017), USENIX Association, pp. 1093–1110.

[3] Baldoni, R., Coppa, E., D'Elia, D. C., Demetrescu, C., and Finocchi, I. A survey of symbolic execution techniques. *ACM Comput. Surv. 51*, 3 (2018).

[4] Behrmann, G., David, A., and Larsen, K. G. A tutorial on uppaal. In *Formal methods for the design of real-time systems*. Springer, 2004, pp. 200–236.

[5] Brumley, D., Hartwig, C., Liang, Z., Newsome, J., Song, D., and Yin, H. Automatically identifying trigger-based behavior in malware. In *Botnet Detection*. Springer, 2008, pp. 65–88.

[6] Cadar, C., Dunbar, D., and Engler, D. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2008), OSDI'08, USENIX Association, pp. 209–224.

[7] Canali, D., Lanzi, A., Balzarotti, D., Kruegel, C., Christodorescu, M., and Kirda, E. A quantitative study of accuracy in system call-based malware detection. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis* (New York, NY, USA, 2012), ISSTA 2012, Association for Computing Machinery, p. 122–132.

[8] Canzanese, R., Mancoridis, S., and Kam, M. System call-based detection of malicious processes. In *2015 IEEE International Conference on Software Quality, Reliability and Security* (Aug 2015), pp. 119–124.

[9] Costin, A., Zaddach, J., Francillon, A., and Balzarotti, D. A large-scale analysis of the security of embedded firmwares. In *23rd USENIX Security Symposium (USENIX Security 14)* (San Diego, CA, 2014), USENIX Association, pp. 95–110.

[10] Cozzi, E., Vervier, P.-A., Dell'Amico, M., Shen, Y., Bigle, L., and Balzarotti, D. The tangled genealogy of IoT malware. In *Annual Computer Security Applications Conference (ACSAC2020)* (Austin, USA, 2020). At the time of writing (Nov 18, 2020), the paper has been accepted to the conference but not yet published. The authors made the paper available to the public.

[11] Fratantonio, Y., Bianchi, A., Robertson, W., Kirda, E., Kruegel, C., and Vigna, G. Triggerscope: Towards detecting logic bombs in android applications. In *2016 IEEE Symposium on Security and Privacy (SP)* (May 2016), pp. 377–396.

[12] Gibert, D., Mateu, C., and Planes, J. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications 153* (2020), 102526.

[13] Gupta, S., Sharma, H., and Kaur, S. Malware characterization using windows api call sequences. In *Security, Privacy, and Applied Cryptography Engineering* (Cham, 2016), C. Carlet, M. A. Hasan, and V. Saraswat, Eds., Springer International Publishing, pp. 271–280.

[14] Hubballi, N., Biswas, S., and Nandi, S. Sequencegram: n-gram modeling of system calls for program based anomaly detection. In *2011 Third International Conference on Communication Systems and Networks (COMSNETS 2011)* (Jan 2011), pp. 1–10.

[15] Kaufman, L., and Rousseeuw, P. J. *Finding groups in data: an introduction to cluster analysis*, vol. 344. John Wiley & Sons, 2009.

[16] Kent, S., and Seo, K. Security architecture for the internet protocol, 2005. Also known as RFC4301.

[17] Kolbitsch, C., Livshits, B., Zorn, B., and Seifert, C. Rozzle: De-cloaking internet malware. In *2012 IEEE Symposium on Security and Privacy* (May 2012), pp. 443–457.

[18] Lee, T., Choi, B., Shin, Y., and Kwak, J. Automatic malware mutant detection and group classification based on the n-gram and clustering coefficient. *The Journal of Supercomputing 74*, 8 (2018), 3489–3503.

[19] Liu, M., Xue, Z., Xu, X., Zhong, C., and Chen, J. Host-based intrusion detection system with system calls: Review and future trends. *ACM Comput. Surv. 51*, 5 (Nov. 2018).

[20] Lynch, N. A. *Distributed Algorithms.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[21] Ma, K.-K., Yit Phang, K., Foster, J. S., and Hicks, M. Directed symbolic execution. In *Static Analysis* (Berlin, Heidelberg, 2011), E. Yahav, Ed., Springer Berlin Heidelberg, pp. 95–111.

[22] Martignoni, L., Stinson, E., Fredrikson, M., Jha, S., and Mitchell, J. C. A layered architecture for detecting malicious behaviors. In *Recent Advances in Intrusion Detection* (Berlin, Heidelberg, 2008), R. Lippmann, E. Kirda, and A. Trachtenberg, Eds., Springer Berlin Heidelberg, pp. 78–97.

[23] Oliver, J., Cheng, C., and Chen, Y. Tlsh–a locality sensitive hash. In *2013 Fourth Cybercrime and Trustworthy Computing Workshop* (2013), IEEE, pp. 7–13.

[24] Schwartz, E. J., Avgerinos, T., and Brumley, D. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *2010 IEEE Symposium on Security and Privacy* (May 2010), pp. 317–331.

[25] Shoshitaishvili, Y., Wang, R., Salls, C., Stephens, N., Polino, M., Dutcher, A., Grosen, J., Feng, S., Hauser, C., Kruegel, C., and Vigna, G. Sok: (state of) the art of war: Offensive techniques in binary analysis. In *2016 IEEE Symposium on Security and Privacy (SP)* (May 2016), pp. 138–157.

[26] Ucci, D., Aniello, L., and Baldoni, R. Survey of machine learning techniques for malware analysis. *Computers & Security 81* (2019), 123 – 147.

[27] Ye, Y., Li, T., Adjeroh, D., and Iyengar, S. S. A survey on malware detection using data mining techniques. *ACM Comput. Surv. 50*, 3 (June 2017).

[28] Yu, B., Fang, Y., Yang, Q., Tang, Y., and Liu, L. A survey of malware behavior description and analysis. *Frontiers of Information Technology & Electronic Engineering 19*, 5 (2018), 583–603.