Universitatea
"Alexandru Ioan Cuza"
din Iasi

Analiza si detectia fisierelor malitioase

Rezumatul tezei de doctorat

Author:
Catalin Valeriu Lita

Coordonator stiintific:
Prof. Dr. Ferucio Laurentiu Tiplea

2018

Lista lucrarilor publicate

1. Catalin-Valeriu Lita. “On Complexity of the Detection Problem for Bounded Length Polymorphic Viruses”. In: Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2016 18th International Symposium on. IEEE. 2016, pp. 371–378.

2. Catalin Valeriu Lita, Doina Cosovan, and Dragos Gavrilut. "Anti-emulation trends in modern packers: a survey on the evolution of anti-emulation techniques in UPA packers”. In: Journal of Computer Virology and Hacking Techniques 14.2 (2018), pp. 107–126.

3. Doina Cosovan and Catalin Valeriu Lita. “Practical aspects related to using Hidden Markov Models for detecting metamorphic file infectors”. In: Symbolic and Numeric Algorithms for Scientific Computing (SYNASC). 2017. (to appear)

Prezentari la CARO

Workshopul CARO este specific industriei Anti-Malware iar presa nu are acces la prezentari. Workshop-ul este cu usile inchise si doar abstractele sunt facute publice pe site.

1. Catalin Valeriu Lita and Doina Cosovan. "An Incursion in the Malware Packer Market”. CARO 2017, Krakow, Poland.

   https://www.eiseverywhere.com/ehome/caro2017/

2. Catalin Valeriu Lita and Doina Cosovan. "Evading Detection with Anti-Emulation Techniques". CARO 2018, Portland, United States of America.

   https://caroworkshop2018.wordpress.com/

Cuprins

Capitolul 1

Introducere

In 1985 termenul de virus de calculator a fost definit de catre Fred Cohen ca "un program ce poate 'infecta' alte programe prin modificarea lor astfel incat sa includa o copie posibil evoluata a programului initial."[Coh87]. Virusii erau considerati in acel moment "o problema majora in securatatea calculatoarelor"[Coh87], dar doar una din punct de vedere teoretic deoarece virusi reali inca nu erau prezenti.

Unul dintre primii virusi apararuti este virusul Brain, ce a fost raspandit in ianuarie 1986. Acest virus infecta sectorul de boot al floppy disk-urilor [FS]. Intentia lui era de a proteja un software medical contra pirateriei.

Noi virusi au fost creati la scurt timp dupa aparitia virusului Brain, astfel ca o problema reala de securitate a aparut ca urmare a acestor virusi. Ca o consecinta a aparitiei de noi virusi curand a fost nevoie de produse de securitate, cunoscute ca si antivirusi sau programe Anti-Malware, pentru a proteja impotriva virusilor. Un produs antivirus este o aplicatie software care protejeaza sistemele informatice impotriva a noi posibile infectii de virusi si elimina infectiile curente in caz ca un sistem este deja infectat. Pentru a rezolva aceasta problema intai trebuie sa detecteze codul virusului intr-un sistem infectat, problema cuonscuta ca si problema detectiei pentru un virus de calculator.

La inceput detectia virusilor nu era complicata, o cautare de substringuri era suficienta. Ulterior virusii au evoluat in a folosi metode specifice in evitarea detectiei.

Una dintre cele mai cunoscute metode de evitare a detectiei este polimorfismul. Cu ajutorul polimorfismului instante diferite ale unui virus au acelasi comportament la executie

dar forma lor poate fi diferita, astfel ca o simpla cautare de substringuri nu e suficienta pentru detectia lor. Un generator polimorfic e acea componenta a unui virus care rescrie secvente de instructiuni astfel incat sa aiba acelasi comportament la executie cu secventa originala dar o alta forma.

Cum virusii au continuat sa evolueze in ultimii 30 de ani, una dintre principalele caracteristici care a evoluat este si evitarea detectiei. Astfel ca produsele antivirus si virusii se updateaza in functie de raspunsul celuilalt: virusii pentru a avita noua detectia iar produsele antivirus pentru a adauga detectie la noile versiuni de virusi recent actualizati. De aceea consideram ca problema detectiei pentru virusi este importanta atat pentru cercetarea academica cat si pentru industria produselor antivirus.

Pentru a putea determina complexitatea detectiei pentru virusii polimorfici am folosit gramaticile formale. Am asociat o gramatica cu un generator polimorfic iar regulile de rescriere dintr-o gramatica sunt asociate cu regulile de

rescriere a unui generator polimorfic. Astfel, complexitatea problemei dedectiei instantelor generate de un anumit generator polimorfic este echivalenta cu problema recunoasterii pentru o gramatica fixata G, care este problema de a decide pentru un cuvant w dat, daca acel cuvant apartine sau nu limbajului generat de gramatica G ($w \in L(G)$).

Un program malitios este un termen general care include si termenul de virus in forma in care a fost definit in 1984. Termenul de virus cum a fost definit in 1984 este restrictionat la programele ce infecteaza alte programe. Sunt si programe cu comportament malitios care se executa fara a avea si comportamentul de a infecta alte programe. Un program malitios este definit ca un program care are un comportament malitios, de exemplu poate oferi acces la distanta catre sistemul infectat sau poate fura diferite credentiale.

Creatorii de malware lucreaza pentru a crea programe cu un anumit comportament militios. Dupa ce acele programe

malitioase sunt raspandite la destule systeme, in curand instate ale lor vor ajunge si la firmele antivirus care vor adauga detectie specifica. Urmatoarea data cand vor incerca sa raspandeasca programul malitios pe un sistem care este protejat cu un produs antivirus, acel program va fi detectat daca detectia a fost adaugata intre timp. Acum acel program malitios este detectat, sters de pe sistemele infectate anterior si nu mai poate fi raspandit pe sisteme noi ce sunt protejate de un produs antivirus. Cum acest lucru se poate intampla cu orice program malitios, creatorii de malware trebuie sa-si updateze programele astfel incat sa nu mai fie detectate de produsele Anti-Malware daca vor sa-si raspandeasca programele malitioase in continuare. Una dintre metodele de evitare a detectiei este adaugarea unui nou strat de protectie peste fisierul malitios original. De exemplu cripteaza tot codul malitios original ce urmeaza sa fie decriptat in timpul executiei noului executabil generat. In acest fel codul malitios initial nu este usor accesibil pentru a fi detectat desi o

detectie pentru el exista deja, doar codul ce decripteaza si incarca in memorie codul malitios este usor accesibil pentru produsele antivirus. Acest nou strat de protectie care este responsabil pentru decriptarea si executia codului protejat este obfuscat si generat cu un generator polimorfic pentru a nu fi usor detectabil. In cazul programelor care infecteaza alte programe, acele programe trebuie sa contina si generatorul polimorfic pentru a putea genera noi instante care arata diferit pentru a nu fi usor detectate. Deoarece oricine vrea sa raspandeasca programe malitioase vrea ca fisierele respective sa nu fie detectate de produsele antivirus, a aparut un nou tip de programe, cunoscut ca si protector (packer) malitios care adauga un nou strat de protectie peste un fisier malitios, tehnica similara cu ce era nevoie sa faca creatorii de malware pentru a evita detectia curenta. Dar de data aceasta aceste programe sunt create doar cu scopul de a evita detectia si de obicei sunt disponibile ca un serviciu.

Un packer, denumit si protector, este o aplicatie ce protejeaza software comercial impotriva pirateriei si analizei. Cu un principiu similar un packer malitios protejeaza un fisier malitios impotriva detectiei Anti-Malware pentru o perioada de timp. Pentru a indeplini acest obiectiv, un packer malitios foloseste tehnici specifice ca polimorfism, obfuscare, anti-debugging, anti-emulare.

Deoarece polimorfismul poate fi rezolvat generic cu un emulator, tehnici anti-emulare au fost adaugate pentru a evita detectia dupa ce produsele antivirus au inceput sa foloseasca emulatoare.

Un **emulator** simuleaza un sistem de operare, cu propria memorie, disk, registri si procese. In acest sistem de operare emulat un fisier malitios poate fi executat fara a afecta sistemul de operare gazda. In cazul nostru un emulator este folosit cu scopul detectiei fisierelor malitioase in functie de comportamentul si artifactele rezultate in urma executiei in emulator. Acest emulator nu simuleaza un sistem de operare

complet doar unu minimal deoare are anumite constrangeri:

- nu trebuie sa foloseasca prea mult spatiu pe disk sau in memorie (100MB spatiu pe disk uneori poate fi considerat a fi prea mult)
- nu poate sa contina toate fisierele dintr-un sistem de operare real, nici macar toate fisierele din directorul system32, limitare conditionata de spatiul limitat ce-l poate folosi
- continutul fisierelor executabile din emulator nu este identic cu cel dintr-un sistem real deoarece este o implementare minimala fara toate functionalitatile dintr-un sistem real
- nu are implementate toate functile Windows API
- pentru unele functii Windows API, chiar daca are implementare, nu este una perfecta, pentru anumiti parametri ar putea returna ceva diferit fata de ce returneaza in sistemul de operare real

- trebuie sa fie rapid, sa nu dureze minute executia unui fisier in emulator, de aceea limite pentru timpul de executie sunt introduse

- trebuie sa ruleze pe diferite arhitercturi de procesor, de exemplu pe ARM sau pe AMD64

O **tehnica anti-emulare** este folosita pentru a detecta sau opri executia cand un program ruleaza intr-un emulator si nu intr-un sistem real. Odata ce stie ca nu ruleaza intr-un sistem real programul poate executa cod diferit fata de ce era standard, de obicei executia programului se opreste rapid sau intra intr-o bucla infinita cand detecteaza emulatorul.

Tehnicile anti-emulare sunt folosite de majoritatea packerelor malitioase pentru a evita detectia. Totusi acest subiect nu a primit o mare atentie in lucrarile de cercetare, in principal deoarece este un domeniu restrictionat, bine cunoscut doar de cei ce lucreaza in industria produselor Anti-Malware sau a creatorilor de packere malitioase.

Aceasta teza este impartita in 3 parti: prima parte este despre complexitatea detectiei virusilor polimorfici de lungime marginita unde am folosit gramatici formale ca si formalism, a doua parte este despre tehnici anti-emulare folosite de packerele malitioare si ultima parte este despre folosirea modelelor Markov ascunse pentru a detecta cod metamorfic.

Capitolul 2

Rezultate de complexitate pentru detectia virusilor polimorfici de lungime marginita

Cohen a propus primele modele formale pentru virusi in 1984 [Coh85] folosind masini Turing. Alte formalisme au urmat, folosind functii recursive [BKM07; Adl90] si gramatici formale [Fil07; Qoz99].

Cohen [Coh85] a aratat ca problema detectiei este nedecidabila la modul general pentru o multime virala. In cazul

unor restrictii, cum ar fi lungime marginita, problema detectiei nu mai poate fi nedecitabila si complexitatea ei este importanta. In 2003 Spinellis a dat exemple de virusi polimorfici marginiti care rezolva problema Satisfiabilitatii si pentru care complexitatea detectiei este NP-completa [Spi03].

Acest capitol al tezei este despre problema detectiei exacte pentru virusi polimorfici de lungime marginita. Gramaticile formale sunt folosite ca si formalism. Vom da un exemplu de gramatica pentru care problema recunoasterii este PSPACE-complata si folosind acea gramatica dam exemplu de un virus pentru care complexitatea detectiei este PSPACE-complata [Lit16].

Problema recunoasterii pentru o gramatica fixata G este importanta in cercetarea noastra deoarece va fi folosita pentru a obtine rezultate de complexitate specifice virusilor care corespund la anumite tipuri de gramatici formale.

Problema recunoasterii pentru o gramatica fixata G (notata $RP(G)$) este definita ca fiind problema de a decide pentru un anumit cuvant w dat, daca acel cuvant $w \in L(G)$. In acest caz gramatica G este fixata si cunoscuta anterior.

In continuare dam exemplu de o gramatica dependenta de context pentru care problema recunoasterii este PSPACE-completa.

Incepem prin a prezenta problema QSAT.

$X = \{x_1, \ldots, x_n\}$ este o multime de variabile booleene. O formula QSAT are urmatoarea forma:

$$\varphi = (Q^1 x_1) \cdots (Q^n x_n) C_1 \wedge \cdots \wedge C_m$$

In aceasta formula C_i se numeste *clauza*. O *clauza* este o disjunctie de variabile, de exemplu $x_5 \vee \bar{x}_0 \vee x_1$. Q^i este un cuantificator care poate fi $\forall$ sau $\exists$.

Problema QSAT este de a decide daca formula φ este adevarata sau falsa. Pentru asta este nevoie de a evalua multe asignari. In cazul in care toti cuantificatorii sunt $\exists$

problema QSAT se reduce la problema Satisfiabilitatii care sa stie ca este NP-completa.

Urmatorul pas este de a codifica o formula QSAT pentru a folosi simbolurile gramaticii noastre.

Pentru urmatorul exemplu particular de formula QSAT,

$$\varphi = (\exists x_1)(\forall x_2)(\forall x_3)(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$$

obtinem urmatoarea codificare:

$$\exists_{\_,\_,0} \forall_{\_,\_,0} \forall_{\_,\_,0} (xaxbbxaaa)(xbxaaxbbb)$$

Acei 3 indecsi de la cuantificatori vor fi folositi ulterior in evaluarea formulei.

Deoarece gramatica are un numar considerabil de reguli, le-am grupat dupa functionalitate in 6 grupuri de reguli:

- R0 - genereaza formule QSAT pentru a fi evaluate
- R1 - asignarea curenta este propagata la clauze
- R2 - evalueaza valoarea de adevar pentru asignarea curenta

- R3 - propaga rezultatul anterior la indecsii cuantificatorilor
- R4 - genereaza urmatoarea asignare in ordine lexicografica
- R5 - obtine rezultatul final dupa ce toate asignarile au fost evaluate

Gramatica PSPACE-completa are urmatoarea forma la final:

- $G = (V, \Sigma, R, S)$
- $V = \{a, b, x, (,), S, S_1, S_2, X, A, B, S_3, S_4, S_{4,0}, S_{4,1}, S_{5,0}, S_{5,1}, S_{6,0}, S_{6,1}, S_6, b_1, b_0, a_0, a_1, S_{exit}, S_7, S_8, S_{8,0}, S_{8,1}, S_9, S_{10,T}, S_{10,F}, S_{11,T}, S_{11,F}, S_{12,T}, S_{12,F}, S_{13}, S_{14}, S_{14,0}, S_{14}, S_{True}, S_{False}, \#, \$\} \cup$

  $\{\exists_{k_1,k_2,k_3} \mid k_1, k_2 \in \{T, F, \_\}, k_3 \in \{0, 1\}\} \cup$

  $\{\exists_{k_1,k_2,k_3,M} \mid k_1, k_2 \in \{T, F, \_\}, k_3 \in \{0, 1\}\} \cup$

  $\{\forall_{k_1,k_2,k_3} \mid k_1, k_2 \in \{T, F, \_\}, k_3 \in \{0, 1\}\} \cup$

  $\{\forall_{k_1,k_2,k_3,M} \mid k_1, k_2 \in \{T, F, \_\}, k_3 \in \{0, 1\}\}\ \};$

- $\Sigma = \{S_{exit}, S_{False}, S_{True}, \#, \$ \}$;
- $R = R_0 \cup R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5$.

Deoarece aceasta gramatica poate decide orice formula QSAT rezulta ca problema recunoasterii pentru aceasta gramatica fixata G este PSPACE-completa.

Acum ca avem un exemplu de gramatica PSPACE-completa putem da cu usurinta exemplu de un virus pentru care complexitatea detectiei este PSPACE-completa.

Consideram exemplul nostru de virus ca fiind un program care:

- cunoaste regulile gramaticii definite anterior
- contine o variabila 'w' care initial contine doar simbolul de start
- are o functie care genereaza urmatoarea valoare pentru variabila 'w' aplicand regulile gramaticii
- genereaza un nou program care este identic cu programul curent si ceea ce difera este doar valoarea variabilei

'w', acest nou program generat va fi scris pe disk si executat

Stim din rezultatele lui Cohen [Coh85] faptul ca problema detectiei virusilor este nedecidabila la modul general. Acest rezultat corespunde cazului in care avem gramatici nerestrictionate. In 2003 Spinellis a dat exemple de virusi marginiti pentru care complexitatea detectiei exacte este NP-completa [Spi03]. Acei virusi rezolva instante ale problemei Satisfiabilitatii.

Am dat exemplu de un virus marginit pentru care complexitatea detectiei este PSPACE-completa. Acest rezultat de complexitate este pentru cazul unei detectii exacte si nu in cazul unei detectii euristice. O detectie euristica ar putea detecta acest program in timp liniar, de exemplu ignorand valoarea cuvantului 'w' dar va avea si alarme false in acelasi timp.

Capitolul 3

Tehnici anti-emulare folosite de packerele malitioase

In acest capitol vom prezenta cum packerele malitioase folosesc si updateaza in continuu tehnicile anti-emulare pentru a evita detectia produselor antivirus.

Vom prezenta cum 5 packere malitioase au evoluat in utilizarea tehnicilor anti-emulare pe o perioada cumulata de timp din 2009 pana in 2015 [LCG18]. La sfarsit vom prezenta si ce tehnici anti-emulare am gasit la packere malitioase recente din 2018.

Bredolab (2009-2010) este un packer care este aproape de

momentul cand packerele malitioase au inceput sa fie folosite ca un serviciu. A inceput prin a folosi intreruperea INT 2E ca o tehnica anti-emulare. A folosit si instructiuni rare pentru o perioada scurta de timp. Principala tehnica anti-emulare pe care a tot actualizat-o a fost generarea de exceptii si redirectarea executiei din functia de tratare a exceptiilor ce a inregistrat-o anterior. Ce a schimbat de-a lungul timpului a fost modalitatea de generare a exceptiilor. A folosit instructiunile 'hlt', 'int 3' si variate moduri de a scrie la o locatie de memorie unde aplicatia curenta nu are drepturi de scriere: 'sub [0x401000], eax', 'xchg [0x401100], edx', 'and [0x401000], ecx'. Acesta este un packer vechi iar tehnicile pe care le folosea cel mai probabil nu vor avea succes si in ziua de azi deoarece emulatoarele au fost imbunatatite si unele tehnici folosite ar putea fi considerate ca si comportament suspect prin simpla lor prezenta.

VIZ (2010-2014) a inceput prin a verifica valoarea de return a functiilor API, a mai verificat si daca codul functiilor

API incepe cu o anumita instructiunie de asamblare. Principala tehnica anti-emulare pe care a folosit-o a fost verificarea integritatii fisierelor de biblioteca. Ceea ce facea era sa incarce in memorie un anumit executabil de biblioteca, de exemplu kernel32.dll si apoi sa verifice anumite campuri din headerul executabilului daca au valoarea intre anumite limite. A inceput prin a verifica valoarea pentru campul EntryPoint si apoi pentru campul SizeOfCode. Aceasta tehnica functioneaza deoarece sistemul de operare emulat nu este identic cu unul real iar acele executabile de biblioteca nu sunt identice cu cele dintr-un sistem real si prin urmare nu au aceleasi caracteristici. Mai tarziu a inceput sa calculeze o valoare aproximativa pentru marimea codului unui executabil fara a citi direct valoarea din headerul executabilului. In acest fel emulatorul este mai greu de updatat pentru a rezolva aceasta tehnica anti-emulare iar pentru creatorii packerului malitios este usor de actualizat prin a verifica alt executabil dupa ce un anumit executabil a fost actualizat in

emulator.

VIZ 2 (2013-2015) a inceput cu verificarea valorii de return a functiilor API dar intr-un mod mai complicat. A verificat daca anumite functii API au o implementare corecta prin testarea faptului ca functia returneaza rezultatul corect in functie de parametri si context. De exemplu la un moment dat a verificat faptul ca functia API 'TF_IsCftmonRunning' verifica existenta unui anumit mutex. A testat asta creand mutex-ul inainte de a apela functia API 'TF_IsCftmonRunning' si apoi verificand daca valorea returnata este cea corecta. Dupa o perioada de timp creatorii acestui packer malitios au decis sa foloseasca alte tipuri de tehnici anti-emulare. Prima schimbare majora a fost sa verifice miscarea mouse-ului. A fost interesant faptul ca aceasta tehnica a fost folosita timp de aproape 6 luni de zile fara a fi actualizata. Ceea ce facea era sa verifice ca mouse-ul se misca macar o data in 60 de secunde dar nu se misca in prima milisecunda. Dupa ce au folosit 6 luni de zile aceasta tehinica au inceput sa foloseasca

o tehnica similara cu cea folosita de packer-ul malitios VIZ si anume verificarea integritatii fisierelor de biblioteca. A folosit aceasta tehnica din iunie 2014 pana in decembrie 2015. Campul verificat era BaseRelocationTableSize si ceea ce a actualizat de-a lungul timpului a fost fisierul executabil verificat. Ceea ce a fost interesant e faptul ca incepand cu august 2015 numele executabilelor de biblioteca ce erau folosite erau sortate in ordine alfabetica sugerand faptul ca in acel moment creatorii acestui packer malitios au automatizat aceasta tehnica anti-emulare pentru a le spune automat care este urmatorul fisier executabil de biblioteca ce poate fi folosit dupa ce fisierul curent a fost actualizat in emulator.

UPA 1 (2013-2015) este un packer malitios care a inceput prin a folosi instructiuni rare si functii API rare. Apoi ca si tehnica pe care a updatat-o in continuu a fost folosirea de functii de callback. A folosit calback-uri TLS, RtlRegisterSecureMemoryCacheCallback si altele. Consideram folosirea functiilor de callback ca fiind o tehnica complicata deoarece

nu este usor de rezolvat intr-un emulator dar in acelasi timp este destul de complicat de gasit alta functie de callback de catre creatorii packer-ului malitios dupa ce functia curenta a fost rezolvata intr-un emulator. Dupa o perioada de timp au inceput sa foloseasca o alta tehnica anti-emulare: folosirea de bucle mari. Aceasta tehnica functioneaza deoarece un emulator nu poate executa un fisier cateva minute si de obicei executia in emulator nu este la fel de rapida ca executia intr-un sistem real. Ceea ce au schimbat de-a lungul timpului a fost numarul de iteratii si modalitatea de integrare a codului pentru buclele mari. Initial buclele erau implementate ca functii ce primeau ca parametru numarul de iteratii iar apoi au integrat codul buclelor in codul principal al fisierului fiind mai complicat de identificat unde anume este codul buclelor.

UPA 3 (2013-2015) este diferit fata de packerele malitioase anterioare prin faptul ca foloseste o tehnica anti-emulare diferita si prin faptul ca a fost folosit exclusiv de downloaderul Upatre. Principala tehnica anti-emulare folosita este una

interesanta: foloseste mesajele Windows pentru a opri executia intr-un emulator. Ceea ce face este sa creeze o aplicatie cu interfata, sa puna un buton, un text-box, un text-area si apoi sa trimita mesaje Windows catre acea aplicatie si sa verifice daca comportamentul este cel asteptat. De exemplu la un moment dat a creat un text-area in care un text a fost setat la initializare si apoi trimite mesajul EM_GETLINECOUNT si se astepta sa primeasca raspuns cu numarul de linii specifice acelui text initial. A fost intersant faptul cum creatorii aceslui packer malitios au insistat in a folosi doar aceasta tehnica anti-emulare, de fiecare data au gasit un nou mesaj Windows care nu era tratat corespunzator de catre emulatoare.

Am analizat si 3 fisiere protejate cu packere malitioase din 2018 pentru a vedea ce tehnici anti-emulare sunt folosite recent. In primul caz am analizat un fisier al familiei Kirts, cunoscuta si ca Worm.Phorpiex. Principala tehnica anti-emulare este folosirea buclelor mari. Avea multiple bucle

mari care erau usor de identificat, o bucla in particular avea 3.714.383 iteratii. Ce a fost intersant a fost faptul ca undeva in mijlocul acelei bucle mari, la iteratia 2.015.748, initializa o variabila cu adresa bibliotecii kernel32.dll si cel mai probabil mai tarziu verifica daca acea variabila a fost initializata corespunzator. Presupunem ca in acel caz particular incerca sa detecteze daca un emulator identifica bucla si decide sa sara peste ea fara a o executa complet, iar in acea situatie variabila nu ar fi fost initializata corespunzator si mai tarziu ar putea detecta ca bucla nu a fost executata complet. Urmatorul fisier analizat apartine familiei Ursnif care este un banker. In acest caz packer-ul malitios folosit pentru protectie avea tot buclele mari ca si tehnica anti-emulare. Ca si diferenta fata de cazul anterior, in acest caz buclele nu erau usor de identificat, de aflat de unde incep si unde se termina. Ultimul fisier analizat apartine familiei Ramnit care este tot un banker. In acest caz tehnica anti-emulare folosita este verificarea continutului fisierelor de biblioteca, o tehnica

similara cu cea folosita de packerele VIZ. In particular acel fisier analizat verifica BaseRelocSize pentru biblioteca ntdsapi.dll. Observam cu sunt folosite cu succes aceleasi tehnici anti-emulare care erau folosite si cu ceva ani in urma.

Prin exemplele date anterior am evidentiat cum tehnicile anti-emulare sunt folosite de packerele malitioase si cum sunt actializate in continuu pentru a avita detectia produselor antivirus. Am observat ca de obicei la inceputul evolutiei unui packer malitios sunt folosite tehnici complicate, probabil deoarece creatorii lor au mai mult timp de dezvoltare la dispozitie la inceput si mai multe idei, ca apoi sa decida ca folosirea unei tehnici care este mai simpla si mai usor de actualizat pentru ei este indeajuns pentru a opri emulatoarele. De cele mai multe ori packerele malitioase ajung sa foloseasca in cele din urma buclele mari sau verificarea continutului fisierelor de biblioteca, acestea fiind si tehnicile anti-emulare pe care le-am gasit in fisiere recente din 2018.

Deoarece creatorii de packere malitioase au acces la ultimele detectii ale produselor Anti-Malware ei pot testa detectia proprielor fisiere si primi notificari cand un produs antivirus detecteaza fisierele protejate de ei. Astfel ca isi vor actualiza codul la scurt timp dupa ce o detectie este adaugata pentru ca noile fisiere protejate sa nu mai fie detectate. Prin faptul ca au acces la detectiile produselor antivirus, ceea ce poate fi considerat un oracol, detectia de noi fisiere ce vor fi protejate cu packerele malitioase este nedecidabila.

Capitolul 4

Folosirea modelelor Markov ascunse pentru detectia de cod metamorfic

Am folosit modele Markov ascunse pentru a testa detectia fisierelor infectate cu file infectorul Evol [CL17]. Evol este cunoscut ca fiind un file infector extrem de metamorfic si care evolueaza semnificativ de la generatie la generatie.

In [Des08] modelele Markov ascunse au fost folosite pentru a detecta cod generat de un generator polimorfic care avea implementate tehnicile folosite de generatorul polimorfic al virusului Evol. Autorii au creat un generator polimorfic

pornind de la generatorul virusului Evol. In comparatie cu aceasta cercetare noi am testat pe fisiere reale infectate cu virusul Evol.

Initial am incercat sa folosim un generator polimorfic existent care emuleaza generatorul de la Evol dar nu am fost multumiti de codul generat astfel ca am cautat fisiere reale ale virusului Evol pentru a testa detectia pe fisiere reale. Am fost norocosi prin a gasi doua fisiere ale virusului Evol pe website-ul vxheaven.org [Evo] iar acele fisiere erau fisiere de tipul pacient 0. Un fisier de tipul pacient 0 este un fisier special creat pentru a raspandi acel virus si nu este rezultatul unei infectii cu acel virus.

Pornind de la un fisier de tipul pacient 0 am executat fisierul intr-o masina virtuala pentru a obtine fisiere infectate iar fisierele infectate le-am grupat pe generatii. Pentru prima generatie am executat un fisier de tipul pacient 0 si am pastrat un fisier infectat care a fost rezultatul infectiei fara a folosi generatorul polimorfic. Evol nu foloseste

generatorul polimorfic la orice infectie asa ca am pastrat pentru prima generatie doar un fisier infectat pentru care generatorul polimorfic nu a fost folosit, adica pentru care codul virusului nu are nicio mutatie. Pentru a doua generatie am executat fisierul din prima generatie si am colectat doar fisierele infectate care au fost rezultatul unei mutatii. Am repetat acest pas pana cand am colectat 100 de fisiere pentru a doua generatie. Pentru a treia generatie am ales un fisier aleator dintre cele 100 de fisiere din a doua generatie, l-am executat intr-o masina virutala si am colectat doar fisierele care au fost rezultatul unei mutatii. Am repetat acest pas pana cand am colectat 100 de fisiere si pentru a treia generatie. Am continuat cu acesti pasi pana am colectat 100 de fisiere pentru generatiile de la 4 la 9. Nu am putut merge mai departe de generatia 9 deoarece fisierele din generatia 9 nu mai infectau alte fisiere in masina virtuala care sa fie rezultatul unei mutatii. Cel mai probabil acest lucru se intampla deoarece codul virusului creste semnificativ in marime de la

generatie la generatie si nu mai gasea fisiere gazda in masina virtuala care sa poata include noul cod rezultat din mutatie.

Dupa ce am colectat fisiere infectate cu virusul Evol pentru toate cele 9 generatii, am procesat fiecare fisier pentru a putea antrena si testa modelele Markov ascunse. Deoarece codul virusului Evol este la EntryPoint-ul fisierului infectat, este usor de stiut de unde incepe dar nu este usor de stiut unde se termina. Astfel ca am decis sa extragem 6000 de bytes de la EntryPoint pentru a fi siguri ca extragem doar cod ce apartine virusului Evol atunci cand procesam un fisier infectat. Cei 6000 de bytes contin doar cod de la Evol deoarece marimea virusului original (care nu are nicio mutatie) este mai mare de 6000 de bytes. Dupa ce am extras o secventa de 6000 de bytes am convertit-o la instructiuni de asamblare x86 si am pastrat doar numele instructiunilor ignorand parametrii. Astfel ca am obtinut o lista cu nume de instructiuni de asamblare. In continuare am codificat fiecare nume de instructiune de asamblare cu un numar pentru a obtine

o lista de numere ce ultirior poate fi folosita la antrenarea si testarea modelelor Markov ascunse.

Pentru estimarea parametrilor modelolor Markov ascunse am folosit algoritmul Baum-Welch ce a primit ca input listele de numere extrase din fisierele infectate. Pentru a obtine similaritatea am calculat log-likelihood-ul folosind un model antrenat anterior si o lista de numere rezultata din procesarea unui fisier.

Pentru exeprimentele ce le vom prezenta in continuare este important sa prezentam cat de mult codul virusului Evol creste in marime de la generatie la generatie. Codul virusului original fara nicio mutatie are 6344 de bytes. In a noua generatie, dupa 8 mutatii, marimea medie pentru codul virusului este de 32468 bytes, o crestere in marime de aproape 5 ori.

Primul experiment pe care l-am efectutat a fost sa testam presupunerea ca este important sa extragem si sa procesam doar cod ce apartine virusului Evol. Stim ca pentru a doua

generatie codul virusului are o marime intre 7857 si 9538 de bytes si am decis sa extragem 100 de secvente de bytes de marime 7000 si 100 de secvente de bytes de marime 10000 din cele 100 de fisiere din a doua generatie. Stim ca daca extragem o secventa de lungime 7000 ea va contine doar codul virusului si daca extragem o secventa de lungime 10000 atunci va contine si o parte din codul fisierului gazda care a fost infectat. Am antrenat un model pe primele 50 de secvente de lungime 7000 si am testat modelul cu celelalte 50 de secvente de lungime 7000 si cu cele 100 de secvente de lungime 10000. Rezultatele au fost cele asteptate, celelalte secvente de lungime 7000 au fost considerate similare iar secventele de lungime 10000 nu au fost considerate similare. Cunoscand acest rezultat am decis sa extragem secvente de lungime 6000 pentru toate generatiile pentru a fi siguri ca procesam doar cod de apartine virusului Evol.

Urmatorul experiment pe care l-am efectuat a fost sa aflam ce se intampla daca antrenam un model pe fisierele

din a doua generatie si testam pe fisierele din celelalte generatii de la 3 la 9 si ce se intampla daca antrenam un model pe fisierele din generatia 9 si testam pe fisierele din generatiile de la 2 la 8. In primul caz, cand am antrenat un model pe secventele din a doua generatie, fisierele din toate celelalte generatii au fost considerate similare. Acesta a fost un rezultat asteptat. Dar cand am testat modelul antrenat pe fisierele din generatia 9 doar generatiile 7 si 8 au fost considerate similare iar pentru fisierele din generatia 6 doar unele fisiere au fost considerate similare. Presupunerea noastra pentru acest rezultat este ca in momentul cand am extras secvente de lungime 6000 de bytes din fisierele din a noua generatie am extras doar o cincime din codul virusului iar in momentul cand am extras tot o secventa de lungime 6000 din generatiile anterioare am extras mai mult de o cincime din

codul virusului si astfel mai multa functionalitate a virusului. Cunoscand acest rezultat neasteptat am decis sa procesam secvente de lungimi mai mici pentru generatiile anterioare si sa testam similaritatea folosind acelasi model care a fost antrenat pe fisierele din generatia a noua cu secvente de lungime 6000. Am procesat secvente de lungime 16, 32, 64, 128, 256, 512, 1000 si 3000. Dupa ce am testat similaritatea pentru noile secvente procesate am obtinut rezultate de similaritate pentru toate generatiile de la 2 la 8 cand am testat pe secventele de lungime 3000. Avand in vedere rezultatele acestui experiment putem trage concluzia ca daca este posibil sa separam fisierele dupa generatii atunci este mai bine sa antrenam un model pe primele generatii. In cazul in care nu se poate face acea separare pe generatii atunci se pot obtine rezultate mai bune daca secventele folosite pentru antrenarea modelului sunt de lungime mai mare decat secventele folosite pentru testarea similaritatii.

Deoarece initial am avut doua fisiere de tipul pacient 0

pentru virusul Evol am decis sa testam ce se intampla daca antrenam un model pe fisierele de la un pacient 0 si testam pe fisierele de la celalalt pacient 0. Am antrenat un model pe fisierele din a doua generatie de la primul pacient 0 si am testat cu fisierele din generatiile de la 2 la 9 de la celalalt pacient 0. Rezultatele au fost de similaritate pentru toate fisierele testate din toate generatiile de la celalalt pacient 0 rezultand ca acest tip de detectie este destul de generica ca sa detecteze si fisiere care apartin unei tulpini a virusului similara cu virusul original dar care nu a fost inclusa in etapa de antrenare a modelului.

Capitolul 5

Concluzii

In primul capitol am dat exemplu de o gramatica pentru care problema recunoasterii este PSPACE-completa si pornind de la acea gramatica am dat un exemplu de un virus polimorfic marginit pentru care problema detectiei este PSPACE-complata. Acest rezultat de complexitate este pentru o detectie exacta care nu admite alarme false.

In al doilea capitol am aratat cum tehnicile anti-emulare sunt folosite de catre packerele malitioase si cum sunt constant actializate pentru a evita detectia produselor antivirus. Tehnicile anti-emulare inca sunt folosite in fisiere recente iar acesta este unul dintre nivelele la care se tine o lupta stransa intre produsele antivirus si packerele malitioase.

In al treilea capitol am folosit modele Markov ascunse pentru a detecta fisiere infectate cu virusul Evol care este un virus metamorfic ce evolueaza considerabil de la o generatie la alta. Putem conluziona ca este important cum extragem si procesam codul virusului, mai ales sa nu includem si cod din fisierul gazda. In cazul in care putem separa fisierele pe genratii, rezultate mai bune de similaritate se pot obtine daca antrenarea se face pe fisierele din primele generatii. In cazul in care nu se poate face aceasta separabilitate atunci antrenarea pe secvente de bytes de lungime mai mare decat secventele folosite pentru testare poate avea rezultate mai bune de similaritate. Pentru a folosi aceasta metoda de detectie intai trebuie sa identificam fisierele care apartin unui anumit virus, sa extragem secventa de cod care contine doar codul virusului si apoi sa antrenam un model.

Bibliografie

[Adl90] Leonard M Adleman. "An abstract theory of computer viruses (invited talk)". In: *Proceedings on Advances in cryptology*. Springer-Verlag New York, Inc. 1990, pp. 354–374.

[BKM07] Guillaume Bonfante, Matthieu Kaczmarek, and Jean-Yves Marion. "A classification of viruses through recursion theorems". In: *Computation and Logic in the Real World*. Springer, 2007, pp. 73–82.

[CL17] Doina Cosovan and Catalin Valeriu Lita. "Practical aspects related to using Hidden Markov Models for detecting metamorphic file infectors". In: *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. 2017.

[Coh85] Fred Cohen. "Computer viruses". PhD thesis. University of Southern California, 1985.

[Coh87] Fred Cohen. "Computer viruses: theory and experiments". In: *Computers & security* 6.1 (1987), pp. 22–35.

[Des08] Priti Desai. "Towards an undetectable computer virus". In: (2008).

[Evo] *Win32.Evol patient 0 samples*. [Accessed: 2017-06-25]. URL: http://vxheaven.org/vl.php?dir=Virus.Win32.Evol.

[Fil07] Eric Filiol. "Metamorphism, Formal Grammars and Undecidable Code Mutation". In: *International Journal of Computer Science and Engineering* 1.2 (2007), pp. 0 –6. ISSN: 1307-6892. URL: http://waset.org/Publications?p=2.

[FS] F-Secure. *Threat description Brain*. URL: https://www.f-secure.com/v-descs/brain.shtml.

[LCG18] Cătălin Valeriu Liţă, Doina Cosovan, and Dragoş Gavriluţ. "Anti-emulation trends in modern packers: a survey on the evolution of anti-emulation techniques in UPA packers". In: *Journal of Computer Virology and Hacking Techniques* 14.2 (2018), pp. 107–126.

[Lit16] Catalin-Valeriu Lita. "On Complexity of the Detection Problem for Bounded Length Polymorphic Viruses". In: *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2016 18th International Symposium on*. IEEE. 2016, pp. 371–378.

[Qoz99] Qozah. *Polymorphism and Grammars*. 1999. URL: http://vxheaven.org/lib/static/vdat/tupolgra.htm.

[Spi03] Diomidis Spinellis. "Reliable identification of bounded-length viruses is NP-complete". In: *Information*

Theory, IEEE Transactions on 49.1 (2003), pp. 280–284.