Universitatea Alexandru Ioan Cuza din Iași

Facultatea de Informatică

Teză de doctorat

Analiza dinamică a aplicațiilor posibil malițioase

Rezumat

Îndrumător:
prof. dr. Dorel Lucanu

Student doctorand:
Vlad Constantin Crăciun

Iași, 2021

Abstract

Evoluția rapidă a tehnologiilor din domeniul securității în ultimele două decenii a permis dezvoltatorilor de amenințări informatice să își dezvolte abilitățile și modul de operare, astfel încât companiile cu interes in același domeniu au fost forțate să se adapteze, dezvoltând tehnologii noi și actualizând arhitecturi vechi. În timp ce o serie de companii de top își permit încă să construiască propriile soluții pentru a face față mai ușor unor provocări, altele preferă să cumpere sau să utilizeze tehnologii open-source. Situația curentă, în ceea ce privește aplicațiile binare malițioase, se află în punctul în care intervenția sub formă de analiză manuală ar trebui utilizată doar în cazuri delicate, însă în practică pare să fie mai utilizată decât ne-am fi așteptat. Intervenția manuală se bazează de cele mai multe ori pe lipsa tehnologiilor care să automatizeze dezpachetarea aplicațiilor obfuscate, sau pe evitarea tehnicilor anti-analiză.

În această teză, propunem o serie de utilitare specifice analizei aplicațiilor binare, ce sunt utilizate pentru a dezvolta alte două proiecte mai complexe: CFG-Dump (un utilitar capabil să reconstruiască aplicații dezpachetate, pornind de la memoria unui proces echivalent unei aplicații obfuscate polimorfic) și COBAI (Dirijor Complex pentru Analiză și Instrumentare Binară). În timp ce CFGDump poate asista dezpachetarea aplicațiilor polimorfice, COBAI a fost proiectat ca un mediu integrat de analiză, bazat pe un motor DBI scris de la zero și având o arhitectura dinamică. Spre deosebire de predecesorii săi, COBAI nu a fost gândit ca un mediu de dezvoltare (însă pune la dispoziție minimul necesar de resurse pentru a construi mecanisme de analiză dedicată) și vine gata echipat cu o serie de tehnologii pregătite să facă față celor mai actuale amenințări informatice, fără a fi necesar ca operatorul acestui mediu, să cunoască programare.

Cuprins

1. Introducere

În ultimii câțiva ani, industria amenințărilor informatice s-a dezvoltat foarte mult, profitând de:

- criptografie (extrem de populară în rândul amenințărilor informatice de tip ransomware[1]);
- anonimizarea plăților online (utilizarea infrastructurii blockchain pentru a plăti răscumpărările cerute de ransomware);
- infrastructura TOR, unde de câțiva ani se întrețin proiecte RaaS[2](Ransomware ca și Serviciu);
- exploatarea comunității open-source, folosind aplicații aparent bine-intenționate cu scop compromițător;
- desincronizarea dintre aplicațiile care evoluează într-un ritm alarmant și arhitecturile hardware depășite care sunt excluse de la aplicarea ultimelor actualizări;
- lipsa de cunoștințe în rândul micilor afaceri care se expun tot mai mult în lumea internetului.

În timp ce cu câțiva ani în urmă, specialiștii în inginerie inversă se delectau cu analiza manuală a unor viruși informatici, datorită faptului că rețeaua de internet dark-web împreună cu tehnologiile blockchain au amplificat amenințările informatice, astăzi este tot mai necesar un mediu automat care sa conducă aceste analize, petrecând mai puțin timp cu executia aplicațiilor sub analiză, și dedicând mai mult

[1] https://www.kaspersky.com/resource-center/threats/ransomware
[2] https://www.crowdstrike.com/cybersecurity-101/ransomware/ransomware-as-a-service-raas/

timp filtrării rezultatelor.

Automatizând analiza aplicațiilor binare malițioase, analiștii ar putea câștiga timp, transformând cele câteva ore sau zile de analiză manuală, în câteva secunde. Totuși, în timp ce deja există o serie de tehnologii ce favorizează automatizarea analizei aplicațiilor binare (emulare, translare binară, virtualizare), autorii amenințărilor informatice au adăugat aplicațiilor pe care le dezvoltă, mecanisme de evitare a analizei astfel încât amenințările create de aceștia să finalizeze execuția curentă dacă unul din multiplele mecanisme de analiză este identificat. Pe de altă parte, a ascunde mediile de analiză sau a le face transparente pentru aplicația analizată, nu este o sarcină deloc ușoară. Per total, analiza aplicațiilor malițioase nu diferă foarte mult față de construcția unui profil (profiling), diverențele fiind însă legate de faptul ca aplicația țintă nu este una benignă ci una malițioasă, în care există nivele succesive de obfuscare capabile sa termine oricând execuția curentă.

La toate cele menționate anterior se adaugă o distanță considerabilă între felul în care companiile și mediul academic privesc către aceste probleme. Deși există colaborări între universități și industrie, este mai puțin probabil ca în urma unei colaborări să rezulte o soluție mai eficientă decât cele deja existente și mai mult probabil ca noua soluție sa fie mai generică, să cuprindă mai multe puncte de vedere și situații care anterior nu erau tratate, să fie demonstrabil mai corectă decât versiunile anterioare, însa cu riscul de a reduce foarte mult performanța și a crește consumul de resurse. Pe de altă parte, o companie este mai interesată de soluții performante, astfel încât să poată procesa o cantitate mare de date într-un timp cât mai scurt, iar pentru a atinge acest ideal este dispusă sa utilizeze o gamă variată de soluții particulare peste care să construiască un mecanism care să decidă când trebuie să fie aplicată fiecare soluție individuală. Plecând de la afirmația curentă, putem observa faptul că industria care oferă soluții contra cost, deși oferă publicului larg o serie de aplicații open-source, prea puțin utilizează alte aplicații open-source existente, deoarece din acestea lipsește performanța și mecanismul de personalizare a utilizării. Un model ideal din acest pnct de vedere, ar trebui să includă:

- **transparență** - pentru utilitarul/mediul de analiză; în timp ce politicile de transparență sunt de cele mai multe ori statice (sunt ajustate manual în general, atunci când sunt identificate probleme), considerăm că pot fi construite și politici dinamice sau auto-ajustabile care să evite intervenția umană;
- **consistență** - pentru a nu produce rezultate eronate, dar pe de altă parte a declanșa dacă se poate comportamente malițioase (când acestea în mod normal ar fi fost evitate), ca efect a felului în care funcționează mediul de analiză;
- **relevanță** - mediul de analiză trebuie să ofere o gamă variată de rezultate, variind intre superficial si foarte granular, astfel încât o analiză să ofere rezultatele dorite;
- **performanță** - acolo unde este posibil, analiza ar trebui să execute mai repede decât viteza nativă a codului (mizând pe optimizarea dinamică a codului), iar unde nu este posibil, o performanță scăzută ar trebui echilibrată de un conținut valoros, greu de obținut în alte circumstanțe;
- **corectitudine** - pornind de la un set mare de intrare, soluția ar trebui să ofere o rată mare de acuratețe, fără să producă crash-uri, sau să execute comportamente neașteptate.

Această lucrare propune o serie de soluții particulare care să faciliteze analiza aplicațiilor binare malițioase, împreună cu câteva utilitare mai complexe care integrează aceste soluții particulare. Momentan în aceasta teză ne concentrăm mai mult pe primele patru aspecte prezentate mai sus, deoarece a demonstra corectitudinea soluțiilor prezentate, ar implica un model matematic mai elaborat dar și o analiză mai atentă pe un set de date mult mai variat. Perspectiva noastră este derivată din scenarii prezente in industrie la momentul prezent, unde analiștii sunt mai eficienți făcând mici ajustări ale unor configurații, decât să scrie cod pentru un mediu de dezvoltare (framework). Suntem totuși de acord cu faptul că mediile de dezvoltare sunt utile atunci când operatorul acestuia dorește să demonstreze ceva, fără prea mult efort, însă a utiliza mediul în sine în producție, s-ar putea să nu fie o idee foarte bună, cel puțin din motive de performanță.

Soluția propusă în această lucrare, implică instrumentare dinamică și manipularea structurilor CFG (control flow graph), cu ingrediente adiționale pentru a utiliza cât mai multe dintre micile piese asociate problemei de rezolvat, în ideea de a construi cât mai aproape de adevăr poza de ansamblu cu care avem de a face. Soluțiile descrise se bazează pe o experiență de cel puțin zece ani în domeniul analizei aplicațiilor binare malițioase, implicând soluții existente particulare dar si medii de analiză integrate. Sluțiile noastre pot asista analiștii care încearcă să înțeleagă aplicațiile binare malițioase, oferind conținut relevant și reducând efortul și complexitatea operațională, iar în același timp facilitând construcția unor servicii care să automatizeze o serie de sarcini. Adiacent cercetării propuse în această lucrare, este posibil ca analiștii și dezvoltatorii să extindă analiza de profil a propriilor aplicații benigne, operând în cea mai mare măsură pe codul aplicației finale si totodată să construiască aplicații server-client de monitorizare la scară mai mare a evolțuiei aplicației, sau să construiască utilitare care să refacă un sistem de operare afectat de aplicații malițioase, pornind de la o analiză detaliată.

1.1 Motivație

Deși utilitarele și mediile de analiză (fie comerciale sau open-source) încearcă să rezolve probleme de securitate specifice, au totuși o serie de limitări:

1. Plaja de probleme pe care o acoperă este fixă, astfel încât în încercarea de a acoperi anumite zone adiacente problemei pe care o rezolvă, sau a cere dezvoltatorilor să adauge anumite functionalități, s-ar putea fie să nu funcționeze, fie sa nu fie ușor de implementat;
2. Lucrând cu medii de analiză orientate mai mult pe dezvoltare (frameworks), sunt necesare abilități de programare, iar atunci când un algoritm devine stabil si verficat în rezolvarea unei probleme, s-ar putea ca utilizarea algoritmului pe o plajă mare de valori de intrare să se dovedească complet ineficientă;
3. Lipsa seturilor de date în faza de testare pentru cele mai multe aplicații open-source, le face pur și simplu inutilizabile când vine vorba de aplicații

malițioase. În unele situații, această lipsă de testare, poate face ca aplicația să crape sau să rămână într-un stadiu eronat neprevăzut;

4. Dezvoltarea rapidă a tehnologiilor forțează politicile și comportamentele statice să devină dinamice.

Pentru a oferi câteva exemple concrete în ceea ce privește afirmațiile anterioare, atingem sumar câteva idei:

1. În timp ce IDAPro este cunoscut ca cel mai popular dezasamblor interactiv și conține și un debugger în același timp, este foarte puțin probabil ca într-un viitor apropiat să adauge tehnologii de automatizare a analizei aplicațiilor binare într-o manieră non-interactivă, fără eforturi suplimentare. Pe de altă parte deși o licență variază intre 400 și 4000 de dolari, multe dintre analizele disponibile online, reflectă capturi de ecran cu utilitare mult mai ieftine precum OllyDbg[3] sau x64dbg[4].
2. Angr este un mediu de analiză ce permite dezvoltarea unor soluții particulare (utilizând un framework) pentru manipularea aplicațiilor binare și printre altele este capabil să extragă structura CFG a aplicației vizată. Totuși, extragerea acestei structuri în cazul unor aplicații mari, ia după evaluarea facută de noi, un timp semnificativ dar și o cantitate mare de resurse;
3. PYIATR, PD32 și SCYLLA sunt utilitare open-source folosite la extragerea conținutului de memorie a unui proces în execuție dar și la repararea conținutului extras, incluzând aici și tabela de importuri. Din experimentele noastre am observat că PYIATR de cele mai multe ori genereaza intrări aleatorii în tabela de importuri făcând rezultatul inutilizabil. Niște probleme similare au fost întâlnite la PD32, unde procesând memoria unor procese active în care execuția se distribuia pe două sau mai multe module, utilitarul a generat module individuale si le-a construit de sine stătător, în loc să își dea seama că sunt parte din aceeași aplicație.
4. Plecând de la premisa că cele mai multe motoare Anti Virus sunt construite

[3] https://www.ollydbg.de/

[4] https://github.com/x64dbg/x64dbg

atât să gestioneze amenințări vechi dar și posibil altele necunoscute, traseul în continuă schimbare a amenințărilor informatice obligă vendorii acestor produse să schimbe arhitecturile și să vină cu idei noi. Astăzi este mult mai importantă validarea securității unei întregi infrastructuri de rețea expusă în internet, decât instalarea individuală pe stații, a unor produse de securitate.

În timp ce mediile obișnuite de analiză de tip SandBox (Cuckoo în [11], AnyRun[5], VMRay[6], Falcon[7], Cape[8], Valkire[9], Hybrid-Analysis[10], JoeSandbox[11], VirusTotal[12], WildFire[13], etc.) sunt capabile să ofere cel mult un nivel superficial de detalii în urma unei analize, un nivel mai granular necesită o monitorizare mai atentă a aplicației vizate. Acest nivel mai granular poate fi atins cu tehnologii precum Instrumentare Binară Dinamică, Emulare, sau cu un control adecvat aplicat unei mașini virtuale. Deoarece un emulator de aplicație (libemu[14], bdshemu in bddisasm[15], PokasEmu[16]) este limitat din punctul de vedere al felului în care au fost implementate anumite funcții API și instrucțiuni, iar mașina virtuală depune un efort în a izola execuția analizată de execuția celorlalte componente ale sistemului de operare, un mediu de analiză bazat pe Instrumentare Dinamică Binară este capabil să preia doar controlul aplicației analizate, limitând astfel efortul depus de o mașină virtuală și eliminând problemele introduse de un emulator. O afirmație cu care suntem și noi de acord este aceea că soluțiile DBI existente, nu au fost construite cu scopul de a analiza aplicații malițioase (concluzie obținută de AVAST

[5] https://any.run/
[6] https://www.vmray.com/
[7] https://go.crowdstrike.com/try-falcon-prevent.html
[8] https://capesandbox.com/
[9] https://valkyrie.comodo.com/
[10] https://www.hybrid-analysis.com/
[11] https://www.joesandbox.com/
[12] https://www.virustotal.com
[13] https://www.paloaltonetworks.com/products/secure-the-network/wildfire
[14] https://github.com/buffer/libemu
[15] https://github.com/bitdefender/bddisasm
[16] https://github.com/AmrThabet/x86Emulator

în [10][17], Kirsch Julian în [12] și Daniele Cono D'Elia în [7][18]), motiv pentru care PIN în [1] și DynamoRIO în [3] fie nu sunt capabile să țină pasul cu provocările aduse de autorii de amenințări informatice, fie performanța acestora se diminuează considerabil prin lipsa unor optimizări necesare în cazul acestui gen de aplicații țintă. Spre exemplu, pentru tratarea injecțiilor de cod, dar și a proceselor fiu, deși DynamoRIO este capabil să urmărească până la un punct crearea de procese dar nu și injecția de cod, PIN cu eforturi mari reușește să facă același lucru. Alte probleme legate de utilitare DBI implică identificarea în timpul analizei, a motorului de instrumentare, utilizând efecte secundare netratate ale unor instrucțiuni sau situații în care întreg contextul de analiză poate fi identificat, făcând în situații izolate să crape aplicație sub analiză.

1.2 Contribuții

Contribuțiile tezei includ:

1. O serie de utilitare și tehnologii de sine stătătoare pentru a adresa probleme specifice analizei aplicațiilor binare:
   - Construcția dump-urilor de memorie generice, incluzând atât întreg spațiul de adrese aferent procesului vizat cât și o serie de meta date care vor descrie zonele de memorie prin adresă, dimensiune si drepturi de acces. Totodată mecanismul nostru trece dincolo de limitele arhitecturale ale sistemului de operare, construind același tip de dump atât pe Windows cât și pe Linux sau MacOS, facilitând analiza dump-ului în lipsa arhitecturii pe care a fost obținut.
   - Reconstrucția rapidă și corectă a tebelelor de importuri, în dumpuri de memorie ale unor procese, luând în calcul configurația procesului vizat (legătura dintre modulele non-librării peste care execută aplicația și numărul de aplicații disjuncte).

[17] https://github.com/sf2team/vb2014
[18] https://github.com/season-lab/sok-dbi-security

- Construcția rapidă și comparația structurilor CFG inter-procedurale obținute din aplicații binare, precum și un mecanism de identificare a configurației procesului și aproximare a Entry Point-ului, folosind un mecansim de forță brută asupra întregului spațiu de structuri CFG identificabile în memoria procesului.
- Un mecanism simplu care să faciliteze execuția concolică a aplicațiilor binare sub sistemul de operare Windows, construit peste un motor DBI scris de la zero, oferind suport pentru execuția comportamentului maliții os in cadrul aplicațiilor APT (amenințare avansată persistentă ce reușesc să evite în varii moduri analiza), prin explorare exhaustivă a spațiului de execuții, dar care ar putea fi utilizat și la acoperirea de cod, sau testare concolică.
- O colaborare cu Andrei Mogage, Emanuel Onica, Rafel Pires și Pascal Felber, în tentativa de a reduce suprafața de atac a sistemelor de integrare continuă care oferă soluții software publicului larg. Contribuția personală în această lucrare vizează pregătirea unei distribuții a compilerului TCC pentru a putea opera într-o enclavă Intel SGX, în ideea de a proteja procesul de construcție a unei alte enclave.

2. CFGDump, un utilitar bazat pe o parte din tehnologiile descrise in capitolul anterior, capabil să aproximeze cu o rată mare de acuratețe aplicația binară deobfuscată din procesul unei aplicații. Spre deosebire de alte aplicații care se ocupă de dezpachetarea aplicațiilor împachetate polimorfic și care utilizează entropia pentru a identifica momentul în care are sens să înceapă procedura de dezpachetare, CFGDump utilizează o monitorizare în timp real a evoluției structurii CFG obținută din procesul vizat la intervale scurte de timp. Deoarece CFGDump a fost construit pentru a prelucra procesul aplicațiilor malițioase, ia în calcul faptul că într-un proces pot executa aplicații disjuncte, ca efect al injecției de cod din alte procese în execuție. Pentru a depăși acest impediment, CFGDump clasifică configurația procesului folosind o abordare descrisă în capitolul anterior, unde utilizarea forței brute pentru descoperirea

întregului spațiu de structuri CFG, scoate la suprafață și informații legate de zonele de memorie peste care se întind aceste structuri. Pentru fiecare structură CFG unică din punct de vedere a apațiului de adrese pe care îl traversează, CFGDump construiește o aplicație unică împreună cu tabela de importuri. Utilizând dump-urile generice descrise în capitolul anterior, CFGDump poate opera reconstrucția unui dump generat pe o cu totul altă arhitectură decât cea pe care este executat.

3. COBAI (Dirijor Complex pentru analiza și instrumentarea binarelor), un mediu de analiză bazat pe un motor DBI construit de la zero, cu o arhitectură dinamică, pregătit să facă față unui număr mare de aplicații benigne și malițioase și care oferă totodată o transparență crescută comparativ cu predecesorii săi. Arhitectura dinamică facilitează:
    - analiza amenințărilor informatice care pot să se mute extern în cadrul rețelei din care face parte sistemul implicat în analiză (prin vulnerabilități de execuție la distanță sau folosind aplicații legitimie din cadrul sistemului de operare), printr-o interfață server-client;
    - analiza amenințărilor informatice care pot să se mute local în cadrul sistemului de operare (de la un proces la altul), prin injecții de cod sau creare de procese fiu direct, sau indirect;
    - alegerea operatorului pentru anumite sub-componente ale motorului DBI, cum ar fi disassembler-ul, în cazul în care modulul implicit are probleme, acesta poate fi înlocuit cu oricare altul;
    - o serie de module pregătite să asiste o analiză, configurabile în așa măsură încât ceea ce este util să poată fi adăugat, iar ceea ce nu este util să poată fi exclus, iar pentru programatorii experimentați, module noi se pot dezvolta astfel încât noi funcționalități să fie adăugte;
    - abilitatea de a instrumenta doar corpul aplicației țintă și nu și librăriile sistemului de operare, spre deosebire de felul în care procedează PIN și DynamoRIO;
    - un nivel crescut de transparență comparativ cu alte DBI-uri existente,

tratând un număr mare de scenarii în care se încearcă identificarea mediului de analiză, precum cele implementate în al-khaser[19](> 200 indicatori), pafish[20](> 50 indicatori), safemachine([10] - 12 indicatori pentru prezența motoarelor DBI);

COBAI este prezentat in Capitolul 4.

1.3 List of Presentations

1. Vlad Craciun, Andrei Nacu and Mihail Andronic, *"It's A File Infector... It's Ransomware... It's Virlock"*, VB2015, "https://www.virusbulletin.com/uploads/pdf/conference/vb2015/Craciun-etal-VB2015.pdf",
   Prague 2015
2. Vlad Craciun and Cristina Vatamanu, *"Win32.Dorbot – A splice between a file infector and a botnet"*, AVAR2015, "http://avar2015.org/vs/agenda/", Danang 2015
3. Vlad Craciun, *"Prospecting ransomware tech"*, BSidesSF, "https://bsidessf2018.sched.com/event/E6jT/prospecting-ransomware-tech", San Francisco 2018

1.4 List of Publications

1. Vlad Craciun, *"Automated analysis of possible malware applications"*, Working Formal Methods Symposium, FROM 2017, publication of category D
2. Vlad Craciun, *"Automated dynamic deobfuscation of static obfuscated binaries (short talk)"*, Working Formal Methods Symposium, FROM2018, publication of category D
3. Craciun Vlad Constantin, Mogage Andrei and Simion Emil, *"Trends in Design of Ransomware Viruses"*, International Conference on Security for Informa-

[19]https://github.com/LordNoteworthy/al-khaser
[20]https://github.com/a0rtega/pafish

tion Technology and Communications, Lecture Notes in Computer Science 11359, Springer 2019, p. 259–272, publication of category C

4. Arusoaie Andrei, Ciobâca Stefan, Craciun Vlad, Gavrilut Dragos and Lucanu Dorel, *"A comparison of open-source static analysis tools for vulnerability detection in c/c++ code"*, 2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), p. 161–168, publication of category C (CORE 2018)
5. Mogage Andrei, Pires Rafael, Crăciun Vlad, Onica Emanuel and Felber Pascal, *"Supply chain malware targets SGX: Take care of what you sign"*, 38th Symposium on Reliable Distributed Systems (SRDS), p. 52-60, publication of Category A (CORE 2020)
6. Craciun Vlad, Felber Pascal, Mogage Andrei, Onica Emanuel and Pires Rafael, *"Malware in the SGX supply chain: Be careful when signing enclaves!"*, IEEE Transactions on Dependable and Secure Computing, publication of category A^* (according to CNATDCU Standards, https://www.info.uaic.ro/wp-content/uploads/2021/03/evaluare-teze-2018.10.01-acum.pdf, and the UEFISCDI 2020 IF list, https://uefiscdi.gov.ro/resource-821958-if_clasament-e2020.pdf)
7. Craciun Vlad, Andrei Mogage, Dorel Lucanu, *"Building deobfuscated applications from polymorphic binaries"*, SPOSE 2021 : 3rd Workshop on Security, Privacy, Organizations, and Systems Engineering (an ESORICS 2021 affiliated workshop, paper submitted on Jul 30, 2021)

2. Un set de utilitare pentru analiza aplicațiilor binare

În acest capitol prezentăm o serie de utilitare ce pot fi folosite de sine stătătoare și care pe de altă parte vor fi utilizate în Capitolele 3 și 4. Tot aici este prezentat și un modul de execuție concolică cu utilizare pentru sistemele de operare Windows cu scopul de a asista analiza aplicațiilor malițioase de tip APT (amenințare avansată persistentă), dar și contribuția la o lucrare în colaborare cu Andrei Mogage, Emanuel Onica, Rafael Pires și Pascal Felber, în încercarea de a oferi o soluție sistemelor de integrare continuă, pentru a reduce suprafața de atac a situațiilor în care un rău făcător poate întrerupe lanțul compilare-linkare-semnare a unei enclave Intel SGX.

Construcția dumpurilor de memorie generice este necesară cu precădere atunci când analiza conținutului de memorie trebuie făcută pe un cu totul alt sistem (soluții de securitate care nu consumă resurse). Soluțiile existente fie încearcă să reconstruiască o aplicație pornind de la conținutului de memorie a unui proces țintă, fie încearcă să construiască un dump de memorie care să fie utilizat în identificarea motivului pentru care o aplicație a crăpat. În cazul în care rețeaua din care face parte sistemul în cauză, nu utilizează același sistem de operare, apare o altă dificultate introdusă de faptul că dumpul de memorie, nu poate fi analizat pe orice alt sistem. Contribuția noastră în această zonă este de a construi dumpuri de memorie generice, astfel încât acestea să poată fi operate pe arhitecturi diferite față de cea pe care au fost colectate. Acest lucru este facilitat de faptul că o aplicație și procesul său

corespunzător împărtășesc cel puțin descrierea spațiului de adrese necesar aplicației împreună cu conținutul celulelor de memorie critice pentru funcționarea acesteia. Soluția noastră GD (Dump Generic) a fost evaluată comparativ cu ProcessDump PD32 și ProcDump din pachetul oferit de Sysinternals, observând faptul că GD a reușit să construiască dumpuri de memorie la fel de mari precum ProcDump însă mult mai rapid, în timp ce PD32 a reconstruit eronat o serie de dumpuri din setul a șase aplicații malițioase facând imposibilă recuperarea sau repararea acestor erori.

Reconstrucția tabelei de importuri în dumpuri de memorie aferente proceselor, este necesară atunci când vizăm aplicații malițioase. Acest mecanism de reparație nu doar fixează alinierea unor adrese și sincronizarea intre datele din memorie și antetul aplicației finale, dar în teorie ar trebui să se ocupe și de tabele de referințe către API-uri ce au fost încărcate dinamic, eventual dacă în memoria procesului rulează multiple aplicații, să nu intersecteze importurile destinate unei aplicații, cu importurile destinate alteia. Deși există utilitare publice pentru a servi acestui scop, din lipsa de atenție și teste a dezvoltatorilor, reconstrucția tabelelelor de importuri se face eronat (cum ar fi la SCYLLA în evaluarea noastră), sau incomplet (cum ar fi la PD32 în evaluarea noastră). Pentru a putea reconstrui corect tabela de importuri, pornim prin a identifica corect configurația procesului ca o relație între numărul de module și aplicații disjuncte ce execută în spațiul unui proces. Clasificarea identifică una din patru configurații posibile:

- (SM,SA) - Modul Unic, Aplicație Unică: Majoritatea aplicațiilor benigne utilizează această configurație;
- (SM, MA) - Modul Unic, Aplicație Multiplă: Majoritatea aplicațiilor benigne în care s-a injectat o bucată de cod din alt proces, vor avea un singur modul (cel principal), și o serie de injecții de cod fără antet asociat, astfel încât aplicații cu scop diferit execută în același spațiu de adrese;
- (MM,SA) - Modul Multiplu, Aplicație Unică: Configurație utilizată de o serie de aplicații malițioase utilizând module ascunse pe lângă modulul principal pentru a deruta aplicațiile care fac dump-uri de proces, în acest caz obținerea unui modul ar echivala cu obținerea unui PFG (Partial Flow Graph);

- (MM, MA) - Modul Multiplu, Aplicație Multiplă: O configurație de asemenea vizată de aplicațiile malițioase și care prinde contur prin lansarea în execuție a unei aplicații benigne din cadrul sistemului de operare și maparea în memoria procesului a aplicației malițioase, astfel încât multiple module vor executa aplicații complet diferite.

Clasificarea configurației proceselor ne asigură faptul că nu corelăm aplicația vizată cu apeluri API din altă aplicație, construind astfel aplicații eronate. Un alt pas către reconstrucția tabelelor de importuri constă în identificarea pointerilor API dinamici (încărcați în timpul execuției) la fel ca și unificarea tabelelor de importuri în configurații (MM, SA) și în timp ce aceștia vor face parte dintr-o tabelă mai mare de importuri, instrucțiunile care referă acești pointeri vor trebui actualizate pentru a relaționa corect cu intrările din tabela de importuri reconstruită. Am comparat soluția noastră GCIR (reconstrucția generică a importurilor bazată pe configurație) cu PD32, PYIATR și SCYLLA și am observat că SCYLLA adaugă valori aleatorii în tabela de importuri reconstruită, PYIATR se apropie de soluția noastră pentru aplicațiile benigne și pe de altă parte produce rezultate destul de diferite pentru aplicațiile malițioase (acolo unde EntryPoint-ul și tabelele de importuri nu sunt neapărat corelate cu modulul principal). PD32 reușește uneori să construiască o tabelă de importuri, însă o leagă în mod eronat de antetul aplicației (utilitare de analiză statică precum IDAPro vor afișa valori invalide pentru pointerii API parte din tabela reconstruită).

Gestionarea structurilor CFG în aplicații binare în literatura de specialitate este cel mai adesea legată de reprezentări intermediare și izomorfism de grafuri. Deoarece această abordare este oarecum greoaie din punct de vedere al performanței și a resurselor implicate (așa cum am observat în evaluarea noastră - angr construiește structuri GFG cel puțin de 200 de ori mai greu decât soluția noastră, utilizând în majoritatea cazurilor mai mult de 10 GB de memorie pentru aplicații mari), soluția devine nefezabilă pentru un număr mare de intrări (multe aplicații ce urmează a fi procesate). Evaluarea familiei de ransomware ACCDFISA a luat mai bine de o ora, folosind angr și doar 5.4 secunde, folosind soluția noastră.

Pentru a avea un algoritm de construcție destul de rapid, am eliminat instrucțiunile din structura nodului, reducând câmpurile principale la adresa absolută a nodului și la tipul legăturii cu următorul nod, evitând astfel utilizarea unor structuri specifice legăturii dintre noduri. Faptul că nu folosim un limbaj intermediar accelerează mai mult întregul proces de construcție astfel încât întregul procedeu are loc doar pe seama instrucțiunilor dezasamblate. Algoritmul de construcție este un algoritm standard de construcție al ICFGs (CFG-urilor inter-procedurale) ușor modificat, astfel încât funcții callback și tabele de pointeri sunt legate de un nod sursă acolo unde aceste referințe sunt prezente ca pointeri constanți printre instrucțiunile nodului în cauză. Pentru a compara aceste structuri ICFG am proiectat un algoritm capabil să transforme o structură ICFG într-o valoare hash pe 64 biți. Algoritmul utilizează trei nivele de abstractizare:

- o structură ICFG este redusă la o structură RTCFG asemănătoare unui arbore, având aceleași noduri din structura ICFG, însă mult mai puține săgeți, eliminând săgețile duplicate către aceleași noduri destinație;
- structura RTCFG este traversată într-o manieră iterativă, folosind un algoritm DFS (parcurgere în adâncime), pentru a construi un tablou aferent traversării;
- tabloul aferent traversării este convertit într-o valoare hash pe 64 biți.

Folosim algoritmul descris în Capitolul 3 pentru a compara capturi succesive (valori hash ale traversărilor ICFG) ale unui proces în execuție și am considerat ca factor declanșator pentru construcția dump-ului, un număr fix de capturi identice. Am evaluat mecanismul de comparație pe două familii de aplicații malițioase (Evol și Crysis), folosind câteva seturi de fișiere de antrenament și test și am concluzionat faptul că pentru aplicațiile care suferă transformări polimorfice/metamorfice la nivelul nodului, dar structura ICFG rămâne constantă, algoritmul nostru de comparație este capabil să identifice în mod unic structura ICFG ca o valoare numerică, păstrând o rată cu atât mai mică de alarme false cu cât structura ICFG devine mai mare.

Pe seama unei versiuni anterioare a COBAI (prezentată în Capitolul 4), am

elaborat un modul de execuție concolică capabil să execute aplicații Windows de un număr de ori echivalent cu numărul de simboluri introdus în fluxul de execuție. Modulul de execuție concolică folosește o semantică ușoară peste instrucțiunile dezasamblate și utilizează Microsoft Z3 ca solver SMT pentru a modela execuții laterale. O stare concolică este utilizată pentru a purta modelele rezolvate anterior cu scopul de a modifica viitoare execuții, astfel încât crearea unor simboluri noi aferente unor valori concrete va forța utilizarea unei valori din model în locul valorii concrete produsă de execuție. Pentru a minimiza impactul adus de analiză, abordarea noastră asociază simboluri numai cu elemente rezultate în urma apelurilor API (valori intoarse și parametri de ieșire). Dintre toate API-urile posibile doar unele sunt candidați pentru inserția de simboluri, deoarece unele apeluri API produc pointeri imposibil de reprodus cu modelul SMT. Am evaluat soluția noastră, utilizând patru aplicații demo și am concluzionat că în medie, în timp ce unele aplicații au luat până la zece execuții pentru a epuiza întreg spațiul, timpul pentru o singură execuție a fost de o secundă. În fiecare din cele patru scenarii, modulul nostru de execuție concolică a reușit să surprindă execuția maximă, aducând la suprafață execuția de interes, în timp ce trei din cele patru aplicații utilizează mecanisme de evitare a sandbox-urilor. Execuția de interes în această situație urmărește să trecă cu succes de toate verificările legate de sandbox.

Într-o lucrare [5] în colaborare cu Andrei Mogage, Emanuel Onica, Rafael Pires și Felber Pascal, am încercat să modelăm o variantă a compilerului TCC, astfel încât să poată compila alte aplicații din interiorul unei enclave Intel SGX, cu scopul de a media o serie de atacuri software supply-chain ce vizeaza construcția de enclave pentru infrastructura cloud. Contribuția a constat in ajustarea codului sursă a compilerului TCC, astfel încât să poată opera cu zone de memorie, în loc de fișiere, zonele de memorie fiindui parțial transferate din afara enclavei, parțial deja prezente in enclava sub forma de definiții statice.

3. Dezpachetarea aplicațiilor binare polimorfice

CFGDump este un utilitar construit, utilizând GD prezentat în capitolul anterior și care încearcă să reconstruiască aplicații deofuscate, pornind de la procesul unei aplicații împachetate polimorfic. În reconstrucția aplicației intră și reconstrucția tabelei de importuri, descoperirea EntryPoint-ului și a configurației procesului în cauză (relația dintre numărul de module și aplicații dizjuncte). În acest sens utilizăm un mecanism de forță brută pentru a descoperi:

- spațiul de structuri ICFG prezente în memoria aplicației;
- EntryPoint-ul fiecărei aplicații unice;
- numărul total de celule de memorie parte din structurile ICFG descoperite;
- momentul în care aplicația este dezpachetată pe baza unei monitorizări a schimbărilor ce au loc în structurile ICFG în loc de utilizarea entropiei.

Pe seama setului celor mai mari structuri ICFG disjuncte, CFGDump construiește aplicații noi, reconstruind tabele de importuri conform cu fiecare structură ICFG și leagă celulele de memorie și noile tabele de importuri de un antet nou. Aplicațiile rezultate pot fi analizate static, având reparate cu o acuratețe mare atât apelurile către funcțiile API inițiale cât și cele încărcate dinamic.

CFGDump clasifică obfuscarea polimorfică în trei nivele: entry-level, mid-level și high-level și pretinde să obțină rezultatele așteptate pentru primele două nivele. Comparând CFGDump cu soluții similare, observăm că, ignorând configurația procesului obținem rezultate eronate, ca de exemplu în cazul lui Emotet unde PD32

crează două dump-uri aferente a două module diferite fiiecare cu propria tabelă de importuri când de fapt cele două module sunt parte ale aceleași aplicații și tabela de importuri ar fi trebuit să fie reconstruită din acest punct de vedere.

Rezulate impresionante au fost obținute în acestă direcție de către Preda Dalla în [6] unde se prezintă o interpretare abstractă orientată spre deofuscare și detecție și Matthieu Tofighi Shirazi în [18] care descrie în detaliu mecanismele de obfuscare și deobfuscare în Secțiunile 2 și 3. Deși cadrul nostru, vizând aplicațiile malițioase duce lipsă de codul original sau de obfuscator, lucrarea prezintă un bun punct de plecare in acest sens. Câteva abordări în legătură cu deofuscarea aplicațiilor de împachetare comerciale (majoritatea utilizând polimorfism entry-level) sunt prezentate în [15, 22] și cu toate că prezintă strategii interesante precum instrumentare binară sau varii tehnici de evitare a analizei, contextul nostru mid-level necesită o abordale diferită sau îmbunătățită. Așa cum a fost menționat în introducere, câteva lucrări[17, 9, 20, 21] au încercat să considere o clasă diferită de ofuscare comparativ cu cele descrise de noi. Acestea au fost analizate și testate, însă nu au făcut scopul domeniului nostru de analiză. În [19] se prezintă un exemplu bun de efort necesar în realizarea unei analize specializate pe un tip de obfuscator concret (Themida [2]) împreună cu implementarea ce o vizează. În implementarea soluției curente am încercat să evităm necesitatea dezvoltării unor soluții particulare împreună cu nivelele de complexitate cu care vin o serie de obfuscatoare.

Pe seama lucrării din [8] (un mediu de analiză bazat pe QEMU, capabil să înregistreze și să redea execuții), Charles Lim prezintă Mal-Xtract în[13] și Mal-Flux în [14], o metodă de a detecta sfârșitul rutinelor de dezpachetare, analizând o redare a execuției unui binar în mediul de analiză Panda Qemu. Autorii evaluează trei aplicații benigne pe opt packere dintre care două sunt folosite și în evaluarea noastră (UPX și Themida). Un alt studiu ce vizează detecția codului de dezpachetat, folosind un emulator este oferit de Paul Royal în [16] unde autorii propun suplimentar un cadru formal pentru abordarea lor.

4. Un motor de Instrumentare Binară Dinamică pentru analiza aplicațiilor malițioase

Punctul de vedere actual asupra motoarelor de Translare Binară Dinamică le face medii de dezvoltare (dificil de utilizat de către non-programatori) și nefezabile în cazul aplicațiilor malițioase evazive. Construcția arhitecturală statică scoate la suprafață dificultăți suplimentare, astfel încât necesitatea de a actualiza modulul de dezasamblare sau operând diferit anumite operații de translare, ar implica cunoașterea arhitecturii și subcomponentelor proiectului. Plecând de la acest comportament, un motor DBI open source ar necesita oameni experimentați să întrețină întregul proiect în loc să permită ca o parte din funcționalitățile motorului să fie înlocuite de alte componente. O altă problemă legată de analiza aplicațiilor malițioase se leagă de monitorizarea codului în mișcare (fie local de la proces la proces prin injecții de cod, fie la nivel de rețea prin execuție la distanță). Nu există o metodă cunoascută la acest moment în cadrul mediilor de analiză, pentru a urmări într-un mod facil acest tip de cod.

COBAI (Dirijor Complex pentru Analiza și Intrumentarea Binarelor) nu este un mediu de dezvoltare DBI, ci un mediu de analiză cu o arhitectură dinamică, cu o interfața server-client pentru a monitoriza codul în mișcare, utilizând un fișier de configurare pentru a permite particularizarea unei analize (parametrizând fiecare modul în parte) și acoperind o gamă largă de transparențe de la nivelul motorului

DBI până la nivelul sistemului de operare.

În timp ce utilitarul vine echipat cu o serie de module, noi module pot fi construite pentru a asista și extinde scenariile de analiză. Totuși, modulele curente sunt mai mult decât suficient pentru a produce rezultate consistente într-un timp scurt. O serie de hook-uri permit mediului de analiză să schimbe comportamentul unor API-uri și instrucțiuni fie, modificând intrarea, simulând efectele secundare sau modificând ieșirea. Modulele specializate pentru hook-uri descriu de asemenea un strat subțire între aplicația analizată și mediul de analiză, oferind transparență crescută, nu doar la nivelul motorului DBI ci pentru întregul sandbox utilizat. Din evaluarea noastră, performanța lui COBAI se situează între DynamoRIO (cea mai bună performanță) și PIN, și deși uneori scade sub aceasta din urmă compensează cu un consum mic de resurse. Comparând performanța cache-ului cu PIN, utilizând un joc vechi (blobby volley), COBAI forțează jocul să opereze în 35 de cadre pe secundă folosind un fișier pentru a înregistra instrucțiunile executate, fără memorie cache și atinge 57 de cadre pe secundă fără a oferi o înregistrare a intrucțiunilor executate, în timp ce în mod nativ jocul stă în 62 de cadre pe secundă, iar PIN îl ține în 40 de cadre pe secunde (comparabil cu varianta în care COBAI îl ține în 57 de cadre pe secunde).

Transparența sistemului de operare a fost evaluată pe trei aplicații, una fiind un ransomware, iar celelalte fiind luate din surse publice. Pentru al-khaser COBAI a trecut 243 din 244 teste, picând un test în care era implicat un API WMI (Windows Management Instrumentation - testul este legat de monitorizarea temperaturii procesorului, iar pentru a fi depășit COBAI ar trebui să sincronizeze trei hook-uri pentru a creea și distruge o structură de date care în mod normal nu ar trebui să existe). Pentru pafish toate cele 54 de teste au fost trecute cu succes și la fel s-a întâmplat și cu cele 39 de teste din Satan ransomware, unde acesta din urmă a stat sub analiză aproximativ doua minute, comparativ cu execuția nativă sau alte medii de analiză, unde nu a executat mai mult de câteva milisecunde, reușind să identifice mediul de analiză. Testele de transparență pentru motorul DBI au fost realizate, utilizând 12 demo-uri create de AVAST în [10]. COBAI a trecut 11 din 12 teste,

PIN a trecut 6, iar DynamoRIO 4, fiecare din ultimele două, forțând aplicația să crape pe câte un demo și picând diferența de până la 12.

Am făcut deasemenea o evaluare a 10 aplicații malițioase, majoritatea fiind familii ransoware unde COBAI a finalizat în cel mult 8 minute pe o aplicație care ar fi executat la nesfârșit. Câteva optimizări în COBAI, fac în așa fel încât latențe de timp și bucle cu execuție infinită să fie ținute sub control. PIN și DynamoRIO fie au executat într-un timp mai lung, fie au făcut aplicația să crape (DynamoRIO - Crysis și GandCrab).

5. Concluzii

În această teză aducem ca și contribuție câteva utilitare pentru rezolvat sarcini specifice în analiza aplicațiilor binare și construim peste acestea alte două aplicații mai complexe: CFGDump și COBAI. CFGDump utilizează o abordare nouă pentru a dezpacheta aplicații binare obfuscate polimorfic, operând structurile CFG într-o manieră nouă pentru a identifica momentul în care aplicația în timpul execuției a ajuns într-un stadiu dezpachetat, dar și pentru a identifica corect configurația procesului astfel încât să extragem toate aplicațiile unice care rulau în memoria procesului. CFGDump clasifică totodată mecanismele de împachetare polimorfică în trei tipuri și vine cu o soluție pentru primele două. COBAI este un utilitar construit pentru analiza aplicațiilor malițioase, și deși uneori are o performanță inferioară soluțiilor existente precum DynamoRIO ți PIN, este capabil să depășească testele de transparență la nivelul motorului de analiză împreună cu cele dedicate sistemului de operare și totdată să optimizeze execuțiile obfuscate în ideea de a accelera execuția acolo unde este posibil, ceea ce îl face un mediu de analiză optim atunci când sunt vizate aplicații malițioase. Faptul că nu sunt necesare abilități de programare, îl face să fie ușor de utilizat de către cei care testează aplicații, spre deosebire de alte soluții precum DynamoRIO, PIN, QBDI care pleacă de la premisa că utilizatorul înțelege mediul de analiză și poate să scrie un program pentru a realiza anumite sarcini.

Atât COBAI cât și CFGDump, au prins viață în 2016-2017 și sunt dezvoltate în colaborare cu Andrei Mogage. Până în acest punct COBAI a trecut prin trei refactorizări majore, având în momentul de față o arhitectura dinamică ce include

sub-module la nivelul motorului principal de analiză. În timp ce pentru alte utilitare DBI, a schimba disassembler-ul sau translatorul, este imposibil, COBAI permite acest tip de schimbări, fiind potrivit pentru cercetare academică (studenții pot lua parte la construcția unor sub-module ale motorului, îmbunătățind pe cele exitente, sau adăugând mecanisme de analiă care să îmbunătățească per total funcționalitățile mediului de analiză).

Momentan, CFGDump asistă dezpachetarea aplicațiilor binare, în timp ce COBAI are grijă de noi variante de ransomware, accelerând procedurile de inginerie inversă și extragerea unor infomrații utile precum chei criptografice, algoritmi criptopgrafici, tehnici de elevare, mecanisme de injecție a codului, în laboratoarele Bitdefender. Ambele proiecte au până la 50.000 linii de cod și sunt scrise mixând limbaje precum C/C++, Assembler și Python.

Direcții viitoare de dezvoltare

Peformanța COBAI: În această direcție, planul nostru este să depășim execuția nativă acolo unde este posibil. Astfel intenționăm să îmbunătățim și să facem mult mai dinamică memoria cache a motorului DBI și totodată să întărim mecanismele de analiză a buclelor, implicând eventual analiza taint, astfel încât să optimizăm codul obfuscat în timp real. Separat de aceasta luăm în considerare optimizarea funcțiilor callback și a mecanismului de translare per total, astfel încât să atingem o performanță maximă.

Cerințe formale: De-a lungul multiplelor analize făcute, am ajuns la concluzia că existența unui mecanism care să reducă modificarea parametrilor între analize diferite prin descrierea formală a ceea ce se dorește obținut în urma analizei, poate reduce considerabil timpul petrecut cu configurarea analizelor. Spre exemplu, parametrii pentru o analiză a unei aplicații binare utilizând COBAI, poate fi necunoscută inițial, însă să prindă contur pornind de la o definiție abstractă a modului în care se dorește ca analiza să aibă loc.

Construcția automată a unor utilitare: Pentru a construi un utilitar care să

înlăture o amenințare informatică, sau chiar un utilitar de recuperare a fișierelor criptate de un ransomware, o analiză trebuie făcută în prealabil unui fișier malițios, iar apoi concluziile formate trebuie mixate cu întregul proces de dezvoltare a unei aplicații. Plănuim să reducem și să automatizăm munca repetitivă, interpretând rezultatele obținute de COBAI pe o serie de analize și să le folosim pentru a construi template-uri pentru asemenea utilitare. Astfel un analist ar putea utiliza COBAI pentru a obține o urmă a execuției iar pe seama ei să genereze un astfel de utilitar, cu efort minim.

Clasificarea aplicațiilor malițioase: Folosind urmele lăsate de analiza execuțiilor, împreună cu o serie de meta date, am putea dezvolta mecanisme care să clasifice aplicații binare pe baza comportamentului obținut în urma execuției. Acest mecanism este de interes pentru detecția aplicațiilor malițioase, unde se utilizează obfuscare.

Îmbunătățirea execuției concolice: Având o colecție care să includă toate ramificațiile aplicației, am putea folosi tehnici de execuție concolică pentru a descoperi lanțul potrivit de condiții cu scopul de a atinge comportamentul dorit în aplicații malițioase care pot evita analiza. Astfel s-ar putea forța comportamentul dorit odată ce știm că există, în loc să apelăm la explorare exhaustivă.

Obfuscare/Deobfuscare: Considerăm că putem avansa în această zonă, implicând tehnologii DBI pentru a întări obfuscarea cât și a utiliza mecanisme de translare binară statică sau analiză taint, pentru a îmbunătăți deobfuscarea în aplicațiile binare polimorfice.

Bibliografie

[1] Pin. https://software.intel.com/content/www/us/en/develop/articles/pin-a-dynamic-binary-instrumentation-tool.html.

[2] Themida. https://www.oreans.com/themida.php.

[3] BRUENING, D., AND AMARASINGHE, S. *Efficient, transparent, and comprehensive runtime code manipulation*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering . . . , 2004.

[4] CHEN, Q., AND BRIDGES, R. A. Automated behavioral analysis of malware: A case study of wannacry ransomware. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)* (2017), IEEE, pp. 454–460.

[5] CRACIUN, V., FELBER, P. A., MOGAGE, A., ONICA, E., AND PIRES, R. Malware in the sgx supply chain: Be careful when signing enclaves! *IEEE Transactions on Dependable and Secure Computing* (2020).

[6] DALLA PREDA, M. Code obfuscation and malware detection by abstract interpretation. *PhD diss.), http://profs. sci. univr. it/dallapre/MilaDallaPreda_PhD. pdf* (2007).

[7] D'ELIA, D. C., COPPA, E., NICCHI, S., PALMARO, F., AND CAVALLARO, L. Sok: Using dynamic binary instrumentation for security (and how you may get caught red handed). In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security* (2019), pp. 15–27.

[8] DOLAN-GAVITT, B., HODOSH, J., HULIN, P., LEEK, T., AND WHELAN, R. Repeatable reverse engineering with panda. In *Proceedings of the 5th Program Protection and Reverse Engineering Workshop* (2015), pp. 1–11.

[9] GUILLOT, Y., AND GAZET, A. Automatic binary deobfuscation. *Journal in computer virology 6*, 3 (2010), 261–276.

[10] HRON, M., AND JERMÁŘ, J. Safemachine: malware needs love, too. https://www.virusbulletin.com/conference/vb2014/

abstracts/safemachine-malware-needs-love-too/, 2014.

[11] JAMALPUR, S., NAVYA, Y. S., RAJA, P., TAGORE, G., AND RAO, G. R. K. Dynamic malware analysis using cuckoo sandbox. In *2018 Second international conference on inventive communication and computational technologies (ICICCT)* (2018), IEEE, pp. 1056–1060.

[12] KIRSCH, J., ZHECHEV, Z., BIERBAUMER, B., AND KITTEL, T. Pwin–pwning intel pin: Why dbi is unsuitable for security applications. In *European Symposium on Research in Computer Security* (2018), Springer, pp. 363–382.

[13] LIM, C., KOTUALUBUN, Y. S., RAMLI, K., ET AL. Mal-xtract: Hidden code extraction using memory analysis. In *Journal of Physics: Conference Series* (2017), vol. 801, IOP Publishing, p. 012058.

[14] LIM, C., RAMLI, K., KOTUALUBUN, Y. S., ET AL. Mal-flux: Rendering hidden code of packed binary executable. *Digital Investigation 28* (2019), 83–95.

[15] MARION, J.-Y., AND REYNAUD, D. Dynamic binary instrumentation for deobfuscation and unpacking. In *Depth Security Conference* (2009).

[16] ROYAL, P., HALPIN, M., DAGON, D., EDMONDS, R., AND LEE, W. Polyunpack: Automating the hidden-code extraction of unpack-executing malware. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)* (2006), IEEE, pp. 289–300.

[17] SAIDI, H., PORRAS, P., AND YEGNESWARAN, V. Experiences in malware binary deobfuscation. *Virus Bulletin* (2010).

[18] SHIRAZI, M. T. *Analysis of obfuscation transformations on binary code*. PhD thesis, Université Grenoble Alpes, 2019.

[19] SUK, J. H., LEE, J.-Y., JIN, H., KIM, I. S., AND LEE, D. H. Unthemida: Commercial obfuscation technique analysis with a fully obfuscated program. *Software: Practice and Experience 48*, 12 (2018), 2331–2349.

[20] UDUPA, S. K., DEBRAY, S. K., AND MADOU, M. Deobfuscation: Reverse engineering obfuscated code. In *12th Working Conference on Reverse Engineering (WCRE'05)* (2005), IEEE, pp. 10–pp.

[21] YADEGARI, B., JOHANNESMEYER, B., WHITELY, B., AND DEBRAY, S. A generic approach to automatic deobfuscation of executable code. In *2015 IEEE Symposium on Security and Privacy* (2015), IEEE, pp. 674–691.

[22] YASON, M. V. The art of unpacking. *Retrieved Feb 12* (2007), 2008.