VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

VYLEPŠOVÁNÍ EXTRAKCE INFORMACÍ ZE SPUSTITELNÝCH SOUBORŮ

IMPROVING EXTRACTION OF INFORMATION FROM EXECUTABLE FILES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **KAREL HÁJEK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. LUKÁŠ ZOBAL**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Hájek Karel**
Program: Informační technologie
Název: **Vylepšování extrakce informací ze spustitelných souborů**
Improving Extraction of Information From Executable Files
Kategorie: Bezpečnost

Zadání:

1. Prostudujte si metody analýzy spustitelných souborů a reverzního inženýrství.
2. Seznamte se s nástroji pro extrakci informací používaných v dekompilátoru RetDec (https://github.com/avast/retdec) a jak se dané nástroje využívají ve společnosti Avast.
3. Studujte doménově specifické informace (například digitální signatury, artefakty překladačů konkrétních programovacích jazyků, artefakty programů modifikujících binární soubory), které mohou být přítomné ve spustitelných souborech a které momentálně nejsou extrahované nástroji projektu RetDec. Zaměřte se primárně na souborový formát Windows PE, v menší míře i na další formáty jako Unix ELF nebo macOS Mach-O.
4. Zhodnoťte přínos extrakce informací z bodu 3 pro praktické využití při analýze škodlivého kódu. Navrhněte metody extrakce alespoň dvou nejpřínosnějších typů z bodu 3.
5. Po konzultaci s vedoucím implementujte navržená vylepšení v bodě 4 a otestujte je na skutečných vzorcích malwaru.
6. Svoji práci zhodnoťte a uveďte výhled na další možný vývoj.

Literatura:

- Dokumentace projektu RetDec
- Sikorski, M., Honig, A. Practical Malware Analysis: a Hands-On Guide to Dissecting Malicious Software. San Francisco: No Starch Press, 2012.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění prvních čtyř bodů a rozpracování bodu pátého.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Zobal Lukáš, Ing.**
Konzultant: Matula Peter, Ing., Avast
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 26. října 2020

Abstrakt

Práce se zabývá rozšiřováním open-source projektu zpětného překladače RetDec společnosti Avast. Jedná se o vylepšování extrakce informací ze spustitelných souborů, které jsou využitelné při analýze škodlivého softwaru. Navrhuje několik možností rozšíření extrakce v projektu RetDec. Z těchto možností jsou vybrány ty nejužitečnější, které se následně implementují. Mezi vybraná rozšíření patří hlubší analýza formátu Authenticode, což je technologie společnosti Microsoft pro digitální podepisování spustitelných souborů na systémech Windows, a tvorba hashe symbolů pro spustitelné soubory na Linuxových systémech. Výstupem práce je implementace vybraných rozšíření a jejich otestování na skutečných vzorcích škodlivého softwaru.

Abstract

This thesis deals with extension of an open-source decompiler project called RetDec maintained by the Avast company. The goal is to develop an extension of data extraction from executable files for malware analysis improvement. The thesis proposes several possible improvements on data extraction in the RetDec project. The most useful of these suggested enhancements are then selected and implemented. The selected enhancements involve calculating a hash of symbol names in Linux executable files and a more extensive analysis of Authenticode format, a Microsoft technology for digital signing of executable files for Windows operating systems. The thesis implements the selected additional data extractions in the RetDec project and tests them on real-world malware samples.

Klíčová slova

RetDec, reverzní inženýrství, spustitelné soubory, PE, ELF, škodlivý kód, Authenticode, import hash, telfhash

Keywords

RetDec, reverse engineering, executable files, PE, ELF, malware, Authenticode, import hash, telfhash

Citace

HÁJEK, Karel. *Vylepšování extrakce informací ze spustitelných souborů*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Lukáš Zobal

Vylepšování extrakce informací ze spustitelných souborů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Lukáše Zobala. Další informace mi poskytl odborný konzultant Ing. Peter Matula. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Karel Hájek
12. května 2021

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Lukášovi Zobalovi za vedení práce, rady a konzultace. Dále bych chtěl poděkovat konzultantovi Ing. Peterovi Matulovi za poskytnutou pomoc a konzultace při psaní této práce.

Obsah

Kapitola 1

Úvod

Počítače jsou v dnešním moderním světě všude kolem nás. Spousta lidí má vlastní počítač ve formě telefonu neustále při sobě, někteří ho mají i doma na stole ve formě notebooku nebo stolního počítače. Všudypřítomnost těchto technologií nám usnadňuje řadu věcí v životě jako je práce, přístup k informacím nebo přístup k zábavě. Tato všudypřítomnost je ale často zneužívaná. Počítače jsou terčem různých útoků, které jsou stále více sofistikované, proto je potřeba vytvářet pokročilejší bezpečnostní systémy, které se snaží takovým útokům zabránit.

Zlomyslný útok bývá často vykonán škodlivým programem, který se ve formě spustitelného souboru dostane do počítače. Aby bezpečnostní mechanismy počítače mohly určit, zdali je program neškodný nebo potenciálně nebezpečný, je potřeba program analyzovat. Tím se snaží získat co nejvíce možných informací o jeho obsahu a chování. Protože právě na základě těchto informací si bezpečnostní systémy nebo lidé vytváří obrázek o možných záměrech takového programu.

Jedním z nástrojů pro analýzu spustitelných souborů je open-source nástroj FileInfo, který je součástí projektu RetDec[1]. Za projektem RetDec stojí česká nadnárodní společnost Avast, která se zabývá počítačovou bezpečností a s jejíž spoluprací tato práce vznikla.

Hlavním cílem této práce je právě vylepšování nástroje FileInfo za účelem získání více informací pro lepší analýzu škodlivých programů. Konkrétně se jedná o vylepšování analýzy digitálních podpisů ve formátu Authenticode, což je nejčastější způsob podepisování programů na systému Windows, a vypočítání import hashe pro spustitelné soubory na Linuxových systémech.

Motivací této práce je poznání různých možností detekce a obrany před škodlivými programy, zkoumání jaké informace o sobě tyto programy zanechávají v samotném spustitelném souboru, jak tyto informace extrahovat a jak je využít za účelem jejich detekování.

Práce je rozdělena do několika kapitol. V kapitole 2 jsou uvedené potřebné znalosti o reverzním inženýrství a spustitelných souborech pro uvedení do kontextu a pochopení dalších částí práce. Další kapitola 3 čtenáře seznámí s nástrojem FileInfo, kterému se věnuje zbytek práce. Poté se v kapitole 4 uvedou možné vylepšení nástroje FileInfo a v kapitole 5 se pro vybrané vylepšení vytvoří návrh pro následnou implementaci do nástroje FileInfo, která je popsána v kapitole 6. Způsob testování jednotlivých rozšíření je uveden v kapitole 7. V závěrečné kapitole 8 jsou shrnuté výsledky práce a výhled do budoucna, který může být užitečný pro další práce zabývající se projektem RetDec.

[1] https://retdec.com/

Kapitola 2

Reverzní inženýrství

Reverzní inženýrství je proces extrakce informací nebo návrhu čehokoliv, co bylo vytvořeno člověkem. Tento koncept existuje mnohem déle než počítače a moderní technologie. Cílem reverzního inženýrství bývá většinou zisk chybějících znalostí, návrhů a nápadů, když jsou tyto informace nedostupné. Někdy jsou tyhle informace schované někým, kdo se o ně nechce podělit, jindy jsou tyhle informace ztracené nebo zničené. Tento proces má dlouhé tradice v mnoha odvětvích, od strojírenství, přes elektroniku až po biologii a software.

Tato kapitola se ale bude dále věnovat pouze perspektivě softwarového reverzního inženýrství a je založena na knize *Secrets of Reverse Engineering* [4].

2.1 Softwarové reverzní inženýrství

Softwarové reverzní inženýrství zahrnuje techniky a nástroje pro získání informací o softwaru, například zdrojový kód, návrh aplikace nebo jeho zranitelnosti. Software je velmi komplexní předmět moderní technologie a softwarové reverzní inženýrství je pouze o tom se podívat, z čeho se takový software skládá a jak funguje.

Reverzní inženýrství je obecně velmi často využíváno za účelem kopírování existujících výrobků a tvorby jejich konkurence, ale tento účel není tak populární v informatice, protože bývá velmi neefektivní kopírovat software jako celek. Software je tak komplexní záležitost, že jeho reverzní inženýrství je velmi drahé. Softwarové reverzní inženýrství může mít řadu využití, které jsou popsána níže.

Škodlivý software

Reverzního inženýrství je hojně využíváno jak při obraně před malwarem (škodlivým softwarem), tak i při jeho tvorbě. Vývojáři antivirových programů se snaží malware rozebrat a zjistit, jak funguje. Reverzním inženýrstvím získají informace o tom, co malware dělá, jaké škody může napáchat, jakým způsobem se šíří a jak ho můžou efektivně odstranit nebo předejít infekci. Na druhé straně jsou vývojáři škodlivého softwaru, kteří využívají reverzní inženýrství k odhalení zranitelností v programech, aby se mohli dostat do systému a spustit svůj kód.

Správa digitálních práv

Od existence osobních počítačů se převedla velká část výrobků chráněných autorskými právy (hudba, filmy aj.) do digitální formy. Tato forma nese obrovské výhody pro uživatele,

ale bohužel i značné nevýhody pro distributory a autory z hlediska ochrany jejich autorských práv. Digitální výrobky jsou velmi často jednoduše kopírovatelné a pirátství takových výrobků je na denním pořádku. Autoři se snaží do svých výrobků vložit další vrstvy zabezpečení, aby je lidé nemohli jednoduše kopírovat nebo upravovat. Právě v tento moment přichází na řadu reverzní inženýrství, které slouží k získání znalostí o těchto zabezpečeních, s cílem je obejít nebo ochranu úplně odstranit.

Dosažení spolupráce s proprietárním softwarem

Interoperabilita je velmi častý důvod pro reverzní inženýrství, pokud se pracuje s proprietární API[1]. Dokumentace je často nedostatečná. Aby dostali vývojáři odpověď na otázky, na které nenašli odpověď v dokumentaci, můžou se samozřejmě zeptat tvůrce API, ale to není vždy reálné nebo efektivní řešení. Se znalostmi reverzního inženýrství se na druhou stranu lze podívat do existujícího programu a získat odpověď.

Vývoj konkurenčního produktu

Reverzování kompletního softwaru pouze za účelem vytvoření konkurenčního produktu je velmi náročné a často nedává z ekonomického hlediska smysl. Bývá totiž jednodušší vytvořit si nový produkt nebo si zakoupit licenci pro existující produkt, který je potřeba. Dává to ale smysl v případech, že konkurenční produkt obsahuje velmi unikátní design nebo algoritmus, který je obtížný vytvořit. Pokud ale něco takového konkurence má, bývá to často chráněné například patentem nebo jinými právními možnostmi.

2.2 Metody analýzy spustitelných souborů

Existuje mnoho způsobů, jak analyzovat spustitelné soubory, ale dají se rozřadit do dvou základních kategorií: dynamická analýza a statická analýza. Oba tyto přístupy mají své výhody a nevýhody. Pro efektivní analýzu je tedy potřeba kombinovat různé techniky jak dynamické, tak statické analýzy. Informace o těchto metodách jsou čerpané z knihy *Practical Malware Analysis* [27].

2.2.1 Dynamická analýza

První kategorií je dynamická analýza, která funguje na principu zkoumání chování programu za jeho běhu. Zkoumají se akce provedené programem, jako například:

- Interakce se soubory – Pozoruje se, které soubory program čte nebo do kterých zapisuje.
- Síťová komunikace – Monitorují se zprávy, které program přijímá nebo posílá po síti, a co takové zprávy obsahují.
- Systémová volání – Zkoumají se volání systémových funkcí. Dle použitých systémových volání lze poznat jaké jsou cíle programu.

Do dynamické analýzy spadá i hlubší procházení stavu programu za běhu v ladícím programu. Je možné se podívat přímo na jednotlivé instrukce programu, na stav programu při jejich vykonávání, jak se předává kontrola mezi jednotlivými funkcemi, které funkce se

[1] Application Programming Interface – sbírka funkcí, které programátor může používat.

používají nejčastěji a ve kterých se provádí hlavní chování programu (například chování popsané v předchozím seznamu).

Dynamická analýza je často rychlým způsobem, jak si vytvořit obrázek o činnostech softwaru. Umožňuje rychle zjistit možné chování a potenciální rizika programu.

Má ale také několik nevýhod. Pokud se jedná o škodlivý software, tak jeho spouštění nese možná rizika. Z tohoto důvodu se pro jeho dynamickou analýzu musí vytvořit bezpečné, izolované prostředí (často se využívá virtualizovaných prostředí), aby nebylo možné poškodit systém nebo síť. Dále bývá problém nacházení všech cest v programu, tedy vymýšlení takových vstupů, abychom pokryli všechny možné stavy nebo výstupy. Může se stát, že nějaké cesty v programu vůbec neobjevíme. Jedna z metod pro hledání různých cest v programu, například za účelem hledání chyb, se nazývá fuzz testování[2].

2.2.2 Statická analýza

Druhou kategorií je statická analýza, během které se na rozdíl od dynamické analýzy žádný kód nespouští a pouze se zkoumá samotný obsah spustitelného souboru. Z jeho obsahu lze získat řadu informací o potencionálním chování a struktuře programu. Mezi typické zkoumané informace patří:

- Architektura
- Velikost
- Kód programu
- Importované a exportované symboly
- Řetězce obsažené v souboru

Statická analýza kódu se také zabývá procházením jednotlivých instrukcí programu a skládáním obrazu o chování programu nebo jeho části.

Obtížnost takové analýzy závisí na použitých ochranách vývojáře analyzovaného spustitelného souboru a komplikovanosti programu. Někteří vývojáři ztěžují tento proces různými metodami obfuskace, za účelem ochrany svého kódu. Často šifrují nebo komprimují samotný kód v binárním souboru, aby se skryli před detekcí antivirových programů. Takové programy při spuštění dešifrují nebo dekomprimují zbytek programu, kterému poté předají řízení.

Oproti dynamické analýze je statická analýza výrazně bezpečnější a nevyžaduje speciální prostředí, protože se program nikdy nespustí. Je ale možné, že se bude analyzovat kód, který se v programu nikdy neprovede, protože není možné vždy zjistit, jestli se k němu dostane řízení.

Další části práce se budou zabývat především předměty spadajícími právě pod statickou analýzu, protože se práce bude dále věnovat extrakci statických informací ze spustitelných souborů.

2.3 Typy nástrojů pro reverzní inženýrství

Softwarové reverzování je hlavně o nástrojích. Následující sekce popíšou nejčastěji používané typy nástrojů pro softwarové reverzní inženýrství. Několik z nich bylo vytvořeno pro jiné účely než reverzování, ale i tak se staly jeho důležitou součástí. Často bývají populární programy, které kombinují funkcionalitu několika těchto typů nástrojů.

[2] https://en.wikipedia.org/wiki/Fuzzing

Disassembler

Disassemblery převádějí obsah spustitelného souboru nebo jeho část do assembleru (jazyka symbolických adres). Jde o převádění číselného kódu instrukce procesoru (tzv. opcode) do mnemotechnických zkratek, které většinou představují, co instrukce dělá. Tenhle proces je podstatně obtížnější, než se může zdát. Často je totiž součástí disassembleru rozeznání instrukcí programu od zbytku. Kvalitní disassembler je jeden z nejdůležitějších nástrojů statické analýzy. Mezi nejpopulárnější disassemblery patří IDA Pro[3], Ghidra[4] nebo Radare2[5].

Debugger

Debugger neboli ladící program je jeden z hlavních nástrojů při vývoji softwaru, který slouží k hledání chyb a ladění programu. Programy jsou příliš složité na to, aby si programátor mohl představit všechny možné situace, které v programu mohou nastat, a debugger nám dovoluje zkoumat stav programu během toho co se provádí. Nejzákladnější dvě funkce debuggeru jsou: 1) nastavování míst (určitá instrukce nebo řádek kódu) v programu, dále jako tzv. breakpoint, na kterých má debugger zastavit vykonávání programu, a za 2) prohlížení stavu zastaveného programu, obsahu registrů, paměti, zásobníku volání atd. Některé ladící nástroje umí i pokročilejší funkce, umožňují volat funkce programu nebo měnit stav programu. Debugger je jeden z nejdůležitějších nástrojů pro dynamickou analýzu programů. Mezi konkrétní představitelé debuggeru patří x64dbg[6], OllyDbg[7] nebo GDB[8].

Zpětný překladač

Zpětný překladač (decompiler) je program, který transformuje strojový kód zpět do vyšší abstrakce, například do pseudokódu (nejčastěji se používá formát jazyka C). Stejně jako klasický překladač překládá zdrojový kód do kódu strojového. Bohužel se ale při klasickém překladu ztrácí spousta informací jako jsou datové typy, identifikátory, komentáře nebo abstraktní programové konstrukce. Klasický překladač ale také optimalizuje, čímž výrazně zhoršuje čitelnost výsledného kódu. Optimalizace mohou některé části kódu úplně vynechat, nahradit nebo přeskládat. Ztráta všech těchto detailů znamená, že není prakticky možné získat z přeloženého kódu originální zdrojový kód. I přes tyto nevýhody je ale zpětný překladač velice užitečný nástroj. Čtení a zkoumání strojového kódu ve vyšší úrovni (například v konstrukcích jazyka C) je výrazně rychlejší než čtení strojového kódu ve formátu jazyka symbolických adres, jak bývá běžné. Mezi známé zpětné překladače patří například Hex-Rays[9], Ghidra, Hopper[10] nebo RetDec.

Nástroje pro monitorování systému

Protože skoro všechna komunikace mezi programem a okolním světem probíhá skrz funkce operačního systému, lze tyto funkce monitorovat za účelem zkoumání spuštěných programů. Nástroje pro monitorování systému můžou sledovat sítovou aktivitu, přístupy k souborům,

[3] https://www.hex-rays.com/ida-pro/
[4] https://ghidra-sre.org/
[5] https://rada.re/n/radare2.html
[6] https://x64dbg.com/#start
[7] http://www.ollydbg.de/
[8] https://www.gnu.org/software/gdb/
[9] https://www.hex-rays.com/
[10] https://www.hopperapp.com/

registrům a dalším možnostem nabízené systémem. Tyto nástroje jsou důležitou součástí dynamické analýzy. Konkrétní nástroje závisí na operačním systému a konkrétní aktivitě, která je monitorována. Mezi používané nástroje pro zkoumání procesů na systému Windows patří Process Monitor[11] nebo Process Hacker[12]. Pro monitorování sítové aktivity se využívá například Fiddler[13] nebo Wireshark[14].

2.4 Spustitelné soubory

Práce se zabývá analýzou spustitelných souborů (také jako binární soubory), proto je důležité zmínit co takový spustitelný soubor typicky obsahuje, jaké existují různé formáty, a další detaily, které jsou důležité pro pochopení následujících kapitol. Informace o obsahu binárních souborů a jednotlivých formátů byly získané především z knihy *Linkers and loaders*, která se zabývá tímto tématem [14].

Spustitelný soubor je soubor obsahující instrukce, které je počítač schopen vykonávat, popřípadě i informace, jak takové instrukce vykonat. Instrukce mohou být přímo pro procesor, ale za spustitelný soubor lze považovat i takový soubor, který obsahuje bajt-kód. Bajt-kód je přenositelná instrukční sada, která ale potřebuje další program, který ji vykoná.

Programy ve formě zdrojového kódu v programovacím jazyce se typicky transformují do strojového kódu, instrukcí procesoru, aby je procesor mohl vykonat. I když je možné, aby byl spustitelný program složen pouze z kódu (v historii je takový formát, který obsahoval pouze strojový kód – MS-DOS .COM formát), tak taková jednoduchost trpí různými omezeními.

Z důvodu optimalizací, bezpečnosti, sdílení kódu a přenášení programů mezi různými systémy se dnes využívá řada složitých formátů, které typicky obsahují následující informace:

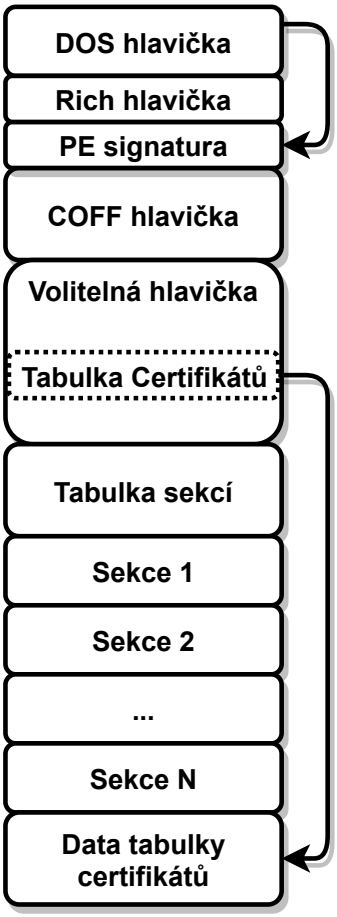
- **Hlavičkové informace** – Celkové informace o souboru, například velikost kódu, datum vytvoření, instrukční sada apod.
- **Kód** – Instrukce a data programu.
- **Relokace** – Seznam míst v kódu, které musí sestavovací program (dále jako linker) upravit při přesunu kódu na jinou základní adresu.
- **Symboly** – Symbolické odkazy neboli symboly jsou metadata o adresách funkcí a proměnných ze zdrojovém kódu.
- **Ladící informace** – Další informace o kódu, které se využívají pro ladění programu. Může obsahovat cestu ke zdrojovému souboru, názvy a typy proměnných, použité datové struktury, mapování jednotlivých instrukcí na řádky ve zdrojovém kódu.

Načtení samotného spustitelného souboru do paměti a jeho provedení má na starosti program zavaděč (anglicky loader), který při spuštění zjistí formát souboru, ověří jeho strukturu, umístí program do paměti a postará se o další věci, které jsou potřeba pro správný běh programu. Mezi nejpoužívanější formáty spustitelných souborů patří dvojice formátů PE a ELF, které jsou popsané v následujících podsekcích. Tyto formáty budou předmětem navrhovaných rozšíření v dalších kapitolách.

[11] https://docs.microsoft.com/en-us/sysinternals/downloads/procmon
[12] https://processhacker.sourceforge.io/
[13] https://www.telerik.com/fiddler
[14] https://www.wireshark.org/

2.4.1 Formát souborů PE

Formát PE (Portable Executable) je formát spustitelných souborů, který je typický pro rodinu operačních systému Windows. Formát PE vychází ze staršího formátu COFF (Common Object File), který byl používaný na systémech Unix. Formát obsahuje všechny informace, které jsou důležité pro zavaděč systému Windows. Soubory PE formátu mají typicky příponu .exe, .dll nebo .sys. Specifika o tomto formátu jsou čerpána z dokumentace společnosti Microsoft [20].



Obrázek 2.1: Struktura PE formátu spustitelného souboru

Jak lze vidět na obrázku 2.1, soubor PE začíná, z důvodů zpětné kompatibility se zavaděčem systému DOS, hlavičkou formátu pro systémy DOS. První dva bajty hlavičky DOS obsahují 2 ASCII znaky, 'M' a 'Z', které hlavičku identifikují. Hlavička DOS obsahuje ukazatel na hlavičku PE souboru, která se skládá z řetězce písmen 'P' a 'E' v jejich ASCII hodnotách, který je zakončený dvěma nulovými bajty. Prostor mezi DOS a PE hlavičkou není zdokumentovaný. Tato oblast se nazývá Rich hlavička, do které nástroje ze sady Visual Studio[15] (konkrétně linker), ukládají informace o vytváření programu. Za PE hlavičkou se nachází hlavička COFF.

COFF hlavička obsahuje typ souboru (jedná-li se o dynamickou knihovnu, spustitelný soubor aj.), velikost volitelné hlavičky, velikost kódu, adresu vstupního bodu programu

[15] https://visualstudio.microsoft.com/cs/

a další. Za ní následuje volitelná hlavička. Její název může mást, i když se nazývá volitelná, je povinná pro spustitelné soubory. Ve volitelné hlavičce může být celá řada informací, které poskytují detailnější informace o souboru jako například tabulka certifikátu, tabulka importů, tabulka relokací a další.

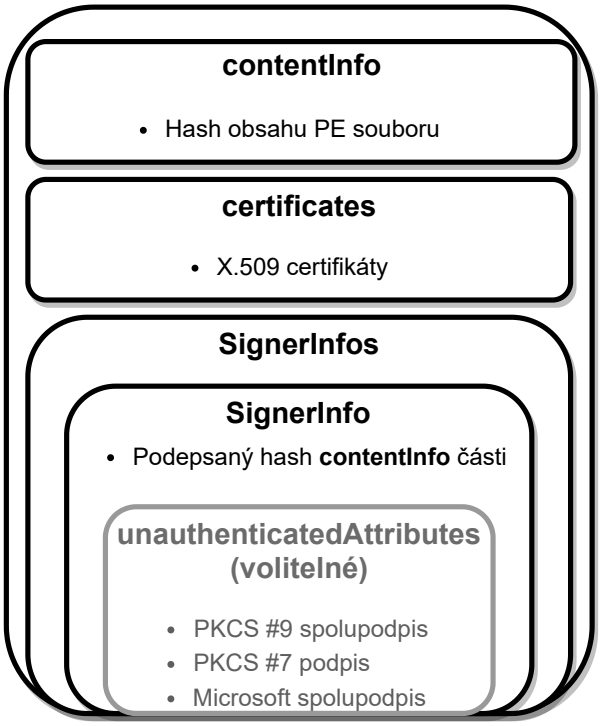
Důležitým prvkem pro spustitelné soubory je tabulka sekcí. Ta obsahuje hlavičku každé sekce, kterou se v souboru nachází. Jejich počet je uložen v COFF hlavičce. Hlavička sekce obsahuje informace o každé sekci, její jméno, pozici v souboru, velikost, relokace a jiné atributy. Existují různé typy sekcí. Mezi ty typické, které se dají v souborech PE najít, patří:

- Sekce obsahující spustitelný kód. Typicky je pojmenovaná jako .text.
- Datové sekce mezi které patří .data, .rdata (data, která jsou pouze pro čtení, tzv. readonly), .bss (rezervovaný prostor pro proměnné, které nejsou inicializované určitou hodnotou).
- Sekce obsahující zdroje, která má název .rsrc a obsahuje informace o souboru jako ikona, font, verze, manifest nebo jiná libovolná data.
- Import sekce .idata a export sekce .edata, které obsahují informace o importovaných nebo exportovaných funkcí spustitelným souborem.
- Sekce s ladícími informacemi .debug.

Tabulka certifikátů

Tabulka certifikátů je součást volitelné hlavičky PE souboru. Pokud má soubor digitální podpis, tak tabulka obsahuje adresu, která ukazuje na začátek těchto podpisů v souboru, což je zobrazeno na obrázku 2.1.

Tato struktura obsahuje velikost podpisu, číslo verze a typ podpisu. Nejčastějším typem je PKCS #7 `SignedData`, který indikuje, že struktura obsahuje podpis ve formátu Authenticode, což je formát od společnosti Microsoft pro digitální podpisování PE souborů za účelem určení původu a zajištění jejich integrity. Formát je založen právě na standardním kryptografickém formátu PKCS #7 typu `SignedData` [10] a certifikátech formátu X.509 [3]. Obsah těchto objektů je definován v notaci ASN.1 (Abstract Syntax Notation One) [7] a je serializován kódováním DER (Distinguished Encoding Rules) [8].



Obrázek 2.2: Zjednodušená struktura Authenticode formátu

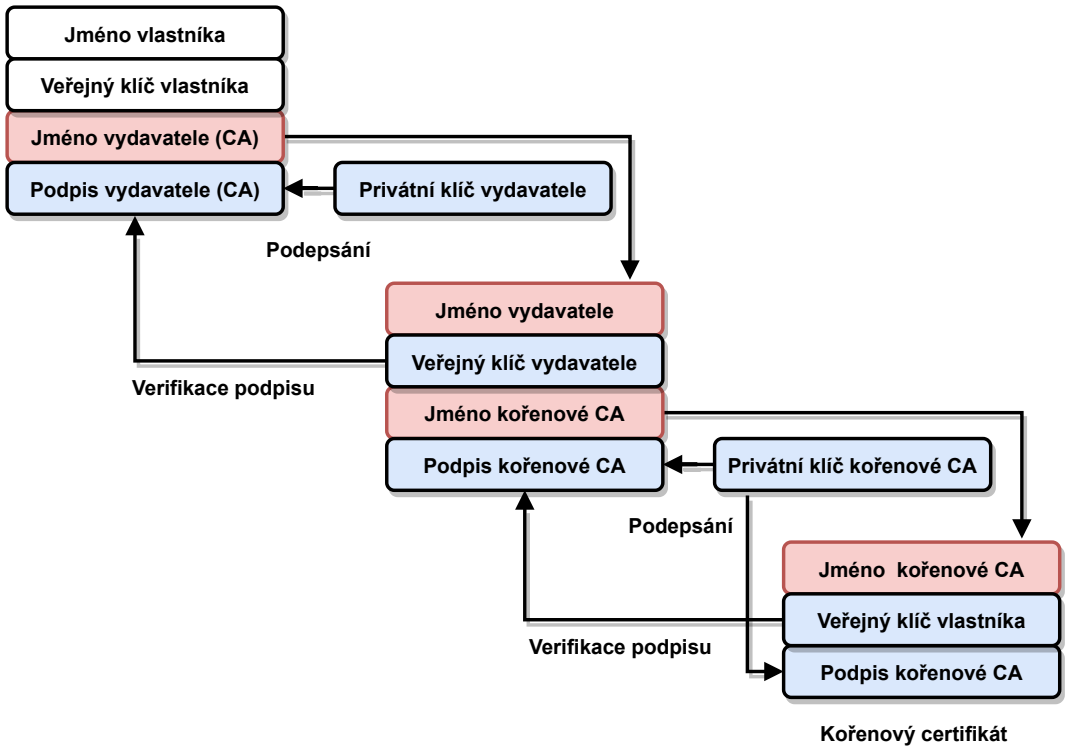
Authenticode formát

Informace o Authenticode formátu pochází především ze specifikace od společnosti Microsoft [19]. Jednotlivé důležité komponenty podpisu jsou graficky ukázané na obrázku 2.2. Hlavním obsahem Authenticode podpisu je hash PE souboru, který je obsažen v `contentInfo`. První krok k ověření originality souboru je shoda vypočítaného hashe s tím, který je uložen v podpisu. Dále je potřeba zjistit, jestli je samotný podpis důvěryhodný. K tomu slouží právě položka `SignerInfo`, která obsahuje podepsaný hash obsahu `contentInfo` části. Pro podepisování se využívá asymetrická šifra, což v tomto kontextu znamená, že existuje privátní a veřejný klíč, první z nich umí šifrovat a druhý dešifrovat obsah, který je zašifrovaný prvním klíčem. Tento podpis slouží ke zjištění, jestli s podpisem nebylo manipulováno.

K dešifrování podepsaného hashe je potřeba veřejný klíč. K tomu slouží certifikáty, které vážou majitele certifikátu (autora podpisu) s jeho veřejným klíčem. Všechny certifikáty, které se v podpisu vyskytují (i v případném PKCS #9 spolupodpisu [22]) jsou uloženy v části `certificates`. Klíčem autora podpisu obsažený v certifikátu je možno dešifrovat zašifrovaný hash v `SignerInfo`. Tím je lze zjistit, že podpis, který je uložen v `contentInfo`, nebyl upraven třetí osobou. Je potřeba ještě zkontrolovat, jestli je možné věřit obsahu samotného certifikátu.

Pro tenhle účel certifikáty také obsahují podpis (podpis jiným certifikátem) a s ním funguje přenos důvěry (chain of trust) [3]. Existují takzvané kořenové certifikáty, pro které platí předpoklad důvěryhodnosti (takové certifikáty často přichází předinstalované na systému). Přenos důvěry je proces postupného ověřování podpisů jednotlivých certifikátů, které na sebe navazují (řetězec certifikátů), který je zakončen ověřením podpisu kořenovým certifikátem, který je již brán jako důvěryhodný. Pokud se podaří takto přenést důvěru z kořenového

certifikátu do certifikátu použitého při podpisu, lze počítat s tím, že s obsahem nebylo manipulováno. Tento proces je zobrazen na diagramu 2.3.



Obrázek 2.3: Řetězec certifikátů

Authenticode umožňuje vkládat do `SignerInfo` různé neautentizované atributy, jako je například spolupodpis nebo vnořený podpis. Authenticode obsahuje dvě varianty spolupodpisu, PKCS #9 nebo Microsoft spolupodpis, který není uveden nikde v oficiální dokumentaci. Spolupodpis je používán jako časové razítko, které nám může říct kdy byl digitální podpis spolu podepsán. Toto razítko je vystavováno ověřenou autoritou časových razítek. O Microsoft spolupodpisu lze z existujících podpisů a nástrojů vydedukovat to, že je uložen ve formátu PKCS #7, který obsahuje `TSTInfo` strukturu, která obsahuje informace o časovém razítku [1].

Dále je možné vložit celý další Authenticode podpis jako atribut do existujícího podpisu. Takhle je možné mít několik různých podpisů pro jeden soubor. Častým použitím vnořeného podpisu bývá přidání podpisu s jinou hashovací funkcí. Důvodem je zrušení podpory pro podpisy využívající hashovací algoritmus SHA1[16] v nových verzích systému Windows. Vydavatelé, kteří jej používali, přidali další podpis s modernějším hashem, jako atribut v kolekci `unauthenticatedAttributes` existujícího podpisu. Tím mohou zachovat zpětnou kompatibilitu.

Rich hlavička

Rich hlavička je prostor mezi DOS a PE hlavičkou, jak jde vidět na diagramu 2.1. K obsahu této hlavičky neexistuje žádná oficiální dokumentace, ale v bezpečnostní komunitě jde již

[16] https://tools.ietf.org/html/rfc3174

o známou a komunitou zdokumentovanou záležitost. Zdrojem informací o této hlavičce je publikace od výzkumníků ze společnosti ESET [24].

Od vydání balíčku Visual Studio 97 SP3 začal linker, z této sady nástrojů, zapisovat nezdokumentovaná data mezi DOS a PE hlavičku v každém vytvořeném spustitelném souboru. Hlavička začíná slovem `DanS` (0x536E6144) a je zakončena slovem `Rich` (0x68636952), za kterým následuje XOR klíč, pomocí kterého jsou data zašifrovaná. Data reprezentují informace o prostředí, ve kterém byl program sestaven, a o velikosti sestaveného projektu. Informace jsou uloženy jako seznam položek, kde každá položka obsahuje identifikátor nástroje a počet objektů, které byly sestaveny pomoci tohoto nástroje. Identifikátor nástroje reprezentuje některý z vývojářských nástrojů od společnosti Microsoft. Mezi takové nástroje může patřit linker, C/C++ překladač, MASM (Microsoft Macro Assembler) a další.

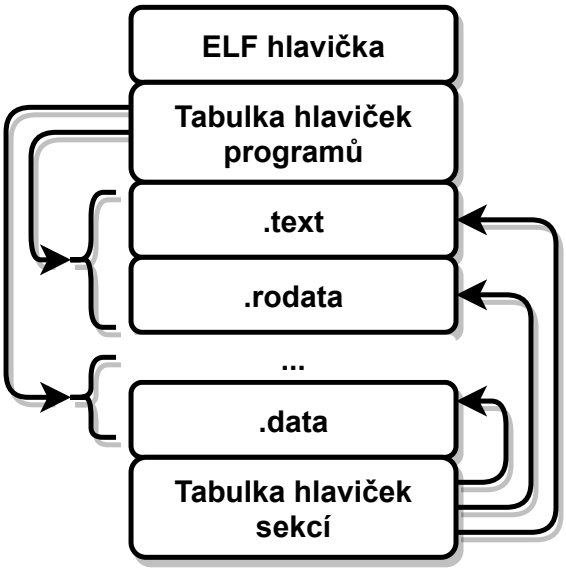
2.4.2 Formát souborů ELF

Executable and Linkable formát neboli zkráceně ELF, je formát pro spustitelné soubory, objektové soubory, sdílené knihovny a výpisy z paměti (tzv. core dumps). Od roku 1999 se používá jako standardní formát pro binární soubory na Unixových systémech. Informace o struktuře formátu jsou čerpány z ELF specifikace [29].

Soubor ve formátu ELF se snadno identifikuje podle sekvence bajtů `0x7fELF` (kde ELF představuje hodnotu těchto znaků dle tabulky ASCII) na začátku souboru.

ELF formát dělí objektové soubory na 3 kategorie.

- **Relokovatelné soubory**, které obsahují kód a data vhodné pro linkování s jinými objektovými soubory za účelem vytvořit spustitelný soubor nebo sdílený objektový soubor.
- **Spustitelné soubory**, které zahrnují program a všechny potřebné informace k jeho spuštění.
- **Sdílené objektové soubory**, které obsahují kód a data pro linkování ve dvou možných kontextech. Buď pro linkování s jinými relokovatelnými nebo sdílenými objektovými soubory, s cílem vytvořit další objektový soubor. Nebo pro dynamické linkování se spustitelným souborem nebo dalšími sdílenými objektovými soubory.



Obrázek 2.4: Příklad pohledu na ELF formát

Obsah ELF souboru se liší na základě toho, do jaké kategorie soubor patří. Obrázek 2.4 ukazuje zjednodušený pohled na spustitelný ELF soubor. ELF hlavička leží na začátku a obsahuje základní informace o struktuře a obsahu zbytku souboru.

Tabulka hlaviček sekcí obsahuje informace popisující sekce v souboru. Pro každou sekci existuje položka v této tabulce. Každá hlavička sekce má informace jako je typ sekce, jméno sekce, velikost sekce a její umístění v souboru. Samotné sekce obsahují kód, data programu nebo více speciální informace jako je tabulka symbolů, relokace a další. Na obrázku 2.4 je možné vidět tři sekce .text, .rodata a .data.

Hlavičky programů říkají zavaděči, jak má připravit program pro spuštění. Spustitelné ELF soubory musí obsahovat tabulku hlaviček programů. Ta specifikuje informace o segmentech. Jak lze vidět na obrázku 2.4, obsahem segmentu jsou sekce (jeden segment obsahuje sekce .text a .rodata). Segmenty říkají systému, které sekce má zavaděč načíst do paměti a jak. Lze specifikovat oprávnění, jako je zákaz zapisování a další.

Sekce ELF formátu, které jsou důležité pro další části práce jsou popsány níže.

Tabulka řetězců

Tabulky řetězců jsou sekce s typem `SHT_STRTAB`. Řetězec je zde sekvence znaků, zakončena bajtem s hodnotou 0x00 (tzv. null bajt). První řetězec v této tabulce je vždy prázdný (skládá se pouze z null bajtu). Typickým využitím této tabulky je uložení názvů symbolů nebo sekcí. Aby v sobě struktury reprezentující symbol nemusely ukládat celý název symbolu jako řetězec, obsahují pouze index do tabulky řetězců, čímž se lze vyhnout duplikaci informací. Může jich tak více ukazovat na jeden stejný řetězec a tím pádem nemusí docházet ke zbytečnému využití paměti.

Symboly

Co je to symbol bylo již zmíněno na začátku sekce 2.4. Tabulka symbolů obsahuje takové symboly, které jsou odkazované někde v souboru. Každý symbol obsahuje informace jako je jeho jméno, typ, viditelnost, velikost, hodnota a další. Jména symbolů jsou uložená jako

index do již zmíněné tabulky řetězců. V ELF formátu se používají dva typy tabulek symbolů `SHT_SYMTAB` a `SHT_DYNSYM`.

- **Tabulka symbolů**
  Tabulka `SHT_SYMTAB` představuje globální tabulku symbolů, protože obsahuje všechny symboly. Jak symboly pro dynamické linkování, tak i symboly, které jsou pro modul lokální a nejsou viditelné mimo daný ELF soubor. Tyto symboly jsou používané pro další linkování nebo pro ladění programu. Je možné tuto tabulku využít i pro dynamické linkování.

- **Tabulka dynamických symbolů**
  Tabulka typu `SHT_DYNSYM` obsahuje pouze symboly potřebné pro dynamické linkování. Z tohoto důvodu její používání šetří místo v paměti, protože se dynamické symboly načítají do paměti procesu. To je úspornější varianta než načítat kompletní tabulku symbolů `SHT_SYMTAB`.

Kapitola 3

Projekt RetDec a nástroj FileInfo

RetDec je open-source projekt vyvíjený společností Avast, který je znám především pro svůj zpětný překladač [13]. Zpětný překladač je napsán v jazyce C++ a je založen na projektu LLVM[1]. Kód binárních souborů převádí na nízkoúrovňový jazyk (Intermediate Representation) vytvořený projektem LLVM, který je platformně nezávislý. Právě z této reprezentace dál vytváří výstup ve formě jazyka vyšší abstrakce jako je jazyk C. RetDec podporuje širokou škálu architektur jako Intel x86, x86-64, ARM, MIPS, PPC atd.

Projekt RetDec se skládá z několika nástrojů, které jsou jak součástí zpětného překladače, tak i samostatnými programy. Jedním z těchto nástrojů je FileInfo.

3.1 Nástroj FileInfo

FileInfo je nástroj pro statickou analýzu spustitelných souborů. Nástroj FileInfo je schopen získat statické informace z několika formátů spustitelných souborů. Především se jedná o formáty ELF, PE a Mach-O. Mezi informace, které je FileInfo schopno extrahovat, patří:

- Hash souboru
- Endianita
- Formát souboru
- Třída souboru (32-bit, 64-bit)
- Typ souboru (spustitelný soubor, objektový soubor, dynamická knihovna)
- Adresa vstupního bodu
- Název sekce vstupního bodu
- Bajty vstupního bodu
- Architektura
- Tabulka sekcí
- Tabulka importovaných symbolů
- Tabulka exportovaných symbolů

[1] https://llvm.org/

- Tabulka symbolů
- Tabulka relokací
- Manifest

FileInfo obsahuje i pokročilejší extrakce jako jsou heuristiky na detekci použitých nástrojů, například použitý linker, packer nebo installer. Protože různé formáty obsahují určitá specifická data pro daný formát, je výstup z programu FileInfo odlišný mezi jednotlivými formáty. Například Authenticode podpisy jsou specifické pouze pro PE soubory.

Nástroj FileInfo umožňuje výstup všech informací jako obyčejný text nebo ve formátu JSON, který se využívá jako vstup do dalších aplikací ve společnosti Avast, které pracují s těmito získanými informacemi z binárních souborů za účelem další analýzy. Některé tyto programy jsou popsané v dalších sekcích.

Textový výstup z nástroje FileInfo lze vidět na ukázce 3.1, který obsahuje stručný přehled o souboru. Nástroj umožňuje i plný „verbose“ výstup, který obsahuje výrazně více informací.

```
Input file : samples/2249611fef630d666f667ac9dc7b665d3b9382956e41f10704e40bd900decbb8
CRC32 : f7112f82
MD5 : 565aa02a153b03e6caeb361a21c73a78
SHA256 : 2249611fef630d666f667ac9dc7b665d3b9382956e41f10704e40bd900decbb8
File format : PE
File class : 32−bit
File type : Executable file
Architecture : x86
Endianness : Little endian
Image base address : 0x400000
Entry point address : 0x40383d
Entry point offset : 0x2c3d
Entry point section name : .text
Entry point section index: 0
Bytes on entry point : 81ecd4020000535556576a2033ed5e896c2418c7442410d8914000896
Detected tool : Microsoft Linker (9.0) (linker), combined heuristic
Detected tool : Nullsoft Install System (installer), YARA rule
Detected tool : Microsoft (linker), dos header style
Rich header offset : 0x80
Rich header key : 0x3e1742a2
Rich header signature : 006dc62700000003007bc6270000001100010000000000ab008378090
Overlay offset : 0x20000
Overlay size : 0x32e2f38
Overlay entropy : 8.000
```

Výpis 3.1: Ukázka výstupu nástroje FileInfo

FileInfo je předmětem této práce, jelikož je to hlavní nástroj pro extrakci informací ze souborů v projektu RetDec. Pozdější kapitoly se budou zabývat návrhy na jeho rozšíření a později i implementací některých těchto návrhů.

3.2 Využití nástroje FileInfo ve společnosti Avast

V rámci společnosti je nástroj FileInfo používán několika aplikacemi, které využívají získané informace v další analýze nebo je uchovávají a zpřístupňují v určité podobě dalším inter-

ních systémům. Zástupcem první kategorie je nástroj Clusty, který se zabývá shlukováním, a zástupcem druhé kategorie je nástroj Scavenger.

3.2.1 Clusty

Každý den se objevují stovky tisíc nových vzorků programů. Některé z nich jsou škodlivé a jiné nikoliv. Analyzovat takové množství programů manuálně je silně repetitivní a časově náročné. Řada škodlivých programů si je podobná, proto existuje program Clusty, který analyzuje tyto nově příchozí vzorky a snaží se je shlukovat do skupin na základě jejich podobnosti.

Na začátku analýzy Clusty identifikuje formát souboru, který se snaží shlukovat. Identifikace pracuje na základě různých kritérií jako jsou například úvodní bajty (magic numbers) souborů, které bývají dané formátem – jako například bajty ELF a MZ pro formáty ELF a PE. Pro hlavní formáty spustitelných souborů jako ELF, PE, Mach-O se v rámci statické analýzy používá právě nástroj FileInfo a pro ostatní formáty se používají specifické programy. Další informace o souboru jsou získané dynamickou analýzou pomocí nástrojů jako je například Cuckoo.

Po získaní všech důležitých charakteristik vzorku začne probíhat hledání vhodného shluku. Vlastnosti jsou seřazené dle jejich priority, a podle této priority se vybírá shluk pro daný vzorek. Při vytvoření nebo modifikaci existujícího shluku se přepočítají vlastnosti celého shluku. Mezi takové společné charakteristiky, které jsou získávané nástrojem FileInfo, může patřit:

- Adresa vstupního bodu
- Bajty vstupního bodu
- Import hash
- Jazyk (C++)
- Rich header
- Hash tabulky sekcí
- Sekce
- Importované symboly

Dále je provedena klasifikace. Klasifikace hodnotí, o jaký typ programu se jedná, zdali je škodlivý a případně o jaký typ malwaru se jedná (PUP, ransomware, worm, aj.) a do jaké rodiny patří. Clusty také umožňuje manuální klasifikaci od analytiků malwarů, která má větší prioritu než automatická klasifikace.

3.2.2 Scavenger

Scavenger je sada aplikací, které automatizují různé procesy v rámci společnosti Avast, především týkající se zpracování vzorků. Scavenger obsahuje například uložené informace o spoustě spustitelných souborů. Tyto informace jsou získané pomocí sady nástrojů, do které patří i nástroj FileInfo, který poskytuje Scanvengeru statické charakteristiky některých typů souborů.

3.2.3 RetDec FileInfo Service

Nástroj FileInfo je dostupný i jako webová služba, která přijímá SHA256 hash souboru, o kterém následně vrací informace získané pomocí tohoto nástroje. Tato služba umožňuje vzdáleně automatizovat analýzy vzorků skrz její webové rozhraní nebo přes API pomocí HTTP GET požadavků.

3.2.4 YaraGen

YARA[2] je open-source nástroj pro identifikaci a klasifikaci vzorků malwaru. YARA umožňuje popsat malware pomocí textových nebo binárních vzorů. Tyto popisy škodlivých souborů se nazývají YARA pravidla. YaraGen je program na automatické generování YARA pravidel ze spustitelných souborů. YaraGen extrahuje informace o souboru na vstupu, tyto informace následně analyzuje a vybírá z nich ty vhodná, ze kterých bude následně generovat YARA pravidla pro daný soubor. Pro získávání informací o souboru používá právě nástroj FileInfo.

[2] https://github.com/VirusTotal/yara

Kapitola 4

Nové možnosti extrakce informací v nástroji FileInfo

Cílem této kapitoly je představit nové možnosti extrakce informací ze spustitelných programů nástroje FileInfo, případně vylepšení stávajících. Nové možnosti extrakce byly vybírané podle existujících nedostatků, které jsou zdokumentované na webové službě GitHub, kde se nachází repozitář projektu RetDec[1], nebo na základě návrhů od týmů, které stojí za nástroji ze sekce 3.2. Jednotlivé možnosti jsou zkoumány z hlediska potencionálního užitku při analýze škodlivých programů.

4.1 Analýza Authenticode podpisů

Authenticode je formát, který slouží k podepisování spustitelných souborů na systému Windows. Jeho struktura a obsah je popsána v sekci 2.4.1. Digitálně podepsaný škodlivý program je schopen obejít ochranné prvky systému, které blokují programy bez platného digitálního podpisu. Zároveň může uniknout antivirovým programům, které přeskakují podepsané programy nebo v ně vkládají větší důvěru. Mezi známe škodlivé programy, které těchto výhod zneužily, patří Stuxnet [25] nebo Flame [2].

I když malware není vždy digitálně podepsaný, jeho podepisování je poměrně běžné. Bezpečnostní společnost Trend Micro, která analyzovala přes 1 600 000 spustitelných souborů stáhnutých z webu, zjistila, že 191 450 z těchto vzorků je škodlivých a 66 % z nich obsahuje digitální podpis [28].

Celý mechanismus podpisů je založen na důvěře. Autoři malwaru jsou ale obecně velmi nedůvěryhodní, a i přes tuto nedůvěryhodnost mají možnost získat validní certifikáty. Část viny je na černém trhu pro nákup certifikátu, kde vlastníci platných certifikátu nabízejí certifikáty vývojářům a distributorům malwaru [12]. Tito prodejci uvádějí, že podepsání programu je nejefektivnější způsob, jak ochránit spustitelné soubory před neustálým blokováním a kontrolami antivirovými programy. Někteří dokonce mění svoji cenu, podle důvěryhodnosti certifikační autority, která certifikát vydala.

Podle studie z americké univerzity v Maryland je dokonce použití podpisu, který neodpovídá podepsanému souboru (tj. hash souboru a hash uložen v podpisu se neshoduje), akceptováno až 34 antivirovými programy. To je chyba, které dle studie zneužívá 31,1 % zkoumaných vzorků [11].

[1] https://github.com/avast/retdec/

Vzhledem ke zmíněným častým problémům má i aktuální analýza a verifikace Authenticode formátu v projektu RetDec podobné mezery. Mezi ty hlavní nedostatky patří neschopnost extrahovat vnořené podpisy a Microsoft spolupodpisy. Tím se přichází o důležité množství informací, které mohou pomoct s analýzou a klasifikací škodlivých programů, které tyto digitální podpisy používají.

4.2 Analýza spustitelných souborů napsaných v jazyce Go

Go je staticky typovaný, kompilovaný jazyk, který byl navržen společností Google[2]. Go se poprvé objevilo v roce 2009 s cíli být jednoduchý k používání a efektivní, především pro síťové a výpočetní aplikace. V posledních letech roste Go na popularitě, a to i mezi vývojáři škodlivých programů. V červenci 2019 bezpečnostní firma Palo Alto Networks vydala analýzu škodlivých programů napsaných v Go [5]. Tato studie zjistila, že mezi lety 2017 a 2019 se množství nalezených škodlivých Go programů zvýšilo o 1994 %. Před rokem 2019 byl malware v jazyce Go poměrně vzácnost, ale během roku 2019 se z toho stala běžná věc a tento rostoucí trend dál pokračoval až do dneška. Mezi používané škodlivé programy, které mají svoji Go variantu použitou ve světě, patří Zebrocy[3] nebo Wellmess[4].

Jazyk Go má pro vývojáře malwaru několik výhod:

- Jeden kód lze přeložit na všechny nejpoužívanější operační systémy: Windows, macOS a Linux, a nespoléhá se na knihovny přítomné v systému, což dovoluje útočníkovy snadno napadnout uživatele všech těchto platforem.
- Všechny knihovny jsou staticky sestavené. Výsledný spustitelný soubor má kvůli tomu silně nadprůměrnou velikost 4,65 MB. To má na jednu stranu jisté nevýhody – takhle velký soubor se špatně distribuuje obětem, například přes email. Na druhou stranu tato velikost zhoršuje analýzu antivirových programů, které kvůli takové velikosti nemusí procházet obsah celého souboru.
- Obsah Go binárních souborů se liší od spustitelných souborů v typických jazycích jako je C nebo C++, což dohromady s jejich velikostí přidává práci reverzním inženýrům.
- Standardní knihovna Go obsahuje sadu robustních kryptografických knihoven a podporu širokého spektra síťových protokolů. Tyto kvality jsou užitečné pro několik typů malwaru, jako RAT[5] nebo Ransomware[6], který je v posledních letech velice populární.

I když je Go pořád jeden z těch méně používanějších jazyků pro tvorbu škodlivých programů, jeho růst v popularitě nelze ignorovat a pravděpodobně bude v následujících letech mnohem častější, než je dnes. Další detaily a konkrétní příklady škodlivých programů napsaných v Go lze najít ve velmi rozsáhlé zprávě od bezpečnostní společnosti Intezer, která analyzuje použití jazyka Go pro zlomyslné účely za rok 2020 [9].

Již zmíněný atypický obsah ztěžuje analýzu Go vzorků. Go ale obsahuje širokou škálu metadat, která nejsou součástí aktuální analýzy v projektu RetDec, ale mohou být užitečná při rekonstrukci symbolů nebo pro přiblížení vzhledu původního kódu. Mezi taková

[2] https://golang.org/doc/
[3] https://us-cert.cisa.gov/ncas/analysis-reports/ar20-303b
[4] https://us-cert.cisa.gov/ncas/analysis-reports/ar20-198b
[5] Remote access trojan – Škodlivý program umožnující kontrolovat systém vzdáleně skrz síť.
[6] Malware, který zamyká soubory nebo i počítač a za odemčení vyžaduje výkupné.

metadata patří například struktura `moduledata`, která existuje od verze 1.5 jazyka Go a obsahuje data pro vytvoření užitečného zobrazení zásobníku volání, když by nastala chyba v programu. Tato data nejsou odstraněna ani když je spustitelný soubor tzv. „stripnutý“, neboli dojde k vymazání všech metadat, která nejsou potřeba pro běh programu (například informace pro ladění programu) a můžou sloužit k odhalení struktury původní kódu. Tato a případné další specifická Go metadata mohou být velmi užitečnou asistencí při analýze vzorků v tomto jazyce.

4.3 Výpočet hashe Rich hlavičky PE souborů

Rich hlavička je prostor v PE spustitelných souborech mezi DOS a PE hlavičkou, kam linker ukládá informace o použitých nástrojích a velikosti sestaveného projektu. Tato hlavička je podrobněji popsána v sekci 2.4.1.

Kromě toho, že Rich hlavička poskytuje informace o použitých typech komponentů a nástrojů při tvorbě malwaru, vytváří také pomocí těchto informací velké množství možných variací, které pomáhají propojovat podobné vzorky. Podle unikátnosti hlavičky lze také naznačit, zda jsou použité komponenty unikátní pro jednoho autora nebo jestli jsou veřejné a rozšířené pro více skupin.

Podle publikace ze společnosti ESET se Rich hlavička nachází v 73,20 % z milionu zkoumaných unikátních škodlivých PE souborů [24]. Velká část z těch, které hlavičku neobsahují, je vytvořena pomocí jiného překladače. Pokud se tyto vzorky odfiltrují, zvedne se poměr až na 83,30 %.

Pokud má nový vzorek stejnou Rich hlavičku jako již známý škodlivý vzorek, je pravděpodobné, že bude také škodlivý. Tímto způsobem lze pomocí Rich hlavičky detekovat potenciálně škodlivé soubory. V případech, kdy je soubor chráněn například protektorem[7], může být Rich hlavička původního programu nezměněna. To umožňuje předpokládat, že soubor je škodlivý, pokud existuje jiný škodlivý soubor se stejnou Rich hlavičkou, nebo lze dokonce vyhledat verzi tohoto programu, který ještě protektor nepoužíval.

Efektivní způsob, jak aplikovat zmíněna porovnání Rich hlaviček a používat tyto data pro shlukování nebo klasifikaci, je pomocí hashe, který by představoval data Rich hlavičky. Stejný hash je používán i webovou stránkou VirusTotal[8], která poskytuje služby pro analýzu malwaru, a může být přínosným vylepšení nástroje FileInfo.

4.4 Tvorba import hashe pro formát ELF

Import hash (také jako ImpHash) je hash všech importovaných symbolů spustitelného souboru. Jeho výpočet funguje obecně následovně: vezmou se všechny názvy importovaných symbolů, ty se převedou do určité definované podoby, a vloží se za sebe do jednoho řetězce. Tento finální řetězec poté projde kryptografickou hašovací funkcí, jejímž výsledkem je právě import hash. Import hash umožňuje reprezentovat všechny používané funkce aplikace z externích knihoven jedním řetězcem, což se dál používá k analýze nebo klasifikaci programu.

Import hash se poprvé začal používat především pro PE formát spustitelných souborů, protože je to metoda pro analýzu a klasifikaci škodlivého softwaru, pro které je nejčastější cíl operační systém Windows. Dnes roste i počet různých škodlivých programů pro linuxové

[7] Program, který šifruje a chrání obsah spustitelného souboru před reverzním inženýrstvím.
[8] https://support.virustotal.com/hc/en-us/categories/360000160117-About-us

systémy, kde je formát ELF nejčastějším formátem spustitelných souborů. Tento nárůst je způsoben především díky rostoucímu počtu a popularity IoT (Internet of Things) zařízení, které používají Linux. Za rok 2019 bylo zaznamenáno 26,66 miliard aktivních IoT zařízení a očekává se více než 75 miliard aktivních zařízení do roku 2025 [15].

Import hash je účinný, protože autoři škodlivých programů mají tendenci používat stejné odzkoušené postupy a návyky. Zároveň rádi šetří čas a snaží se nepřidávat si více práce, než je potřeba. Pokud se autor škodlivého programu snaží změnit signaturu svého spustitelného souboru (s cílem se vyhnout detekci antivirovými programy), je šance, že nezmění používané funkce a knihovny, což znamená, že výsledný import hash aplikace také zůstane stejný. Nebo pokud se ze stejného zdrojového kódu vytváří několik spustitelných souborů s malými odchylkami nebo jen verze pro různé architektury, tak je velká šance, že jejich import hash bude stejný, i když jejich signatura může být odlišná [16].

Stejný import hash dvou různých souborů ale nemusí znamenat, že jsou soubory součástí stejné rodiny škodlivých programů nebo že jsou od stejného autora. Někteří vývojáři můžou dokonce import hash zneužít, aby napodobovali jiné vývojáře malwaru. Import hash je nenáročná metoda, která prokázala, že je schopna efektivně dodat další užitečný indikátor pro sledování a klasifikaci aplikací.

Nedávno se kvůli zmíněnému rostoucímu množství škodlivých programů ve formátu ELF začal využívat import hash i pro ně. Používaným algoritmem je telfhash od společnosti Trend Micro, který by mohl být užitečným přídavkem do projektu RetDec, který aktuálně nevytváří žádný import hash pro soubory v ELF formátu [18]. Tento algoritmus začala používat i webová služba VirusTotal [17].

4.5 Analýza přeložených automatizačních skriptů

V posledních letech se začaly objevovat skriptovací jazyky, jejichž účel je automatizace úloh, typicky v prostředí systému Windows. Mezi nejznámější představitele těchto jazyků patří AutoIt[9] a AutoHotkey[10]. Kvůli jednoduchosti a širokým možnostem těchto jazyků je dnes lze najít v repertoáru autorů škodlivých programů. Za posledních pár let lze najít několik příkladů útoků využívajících tyto jazyky. Například útok cílený na finanční instituce v Kanadě a USA, který má za účel zisk přístupových údajů [6]. Dalším příkladem je botnet Retadup, který využíval jak AutoIt, tak i AutoHotkey, a infikoval přes 850 tisíc počítačů a serverů [26].

Pro skrytí opravdové funkcionality skriptu před obětí a jeho snadnou distribuci používají autoři překladače, které jsou schopné takové skripty zapouzdřit do binárních souborů a často je i obfuskovat (skrývá skutečný cíl skriptu pomocí náhrady řetězců, komprese a šifrování). Překladače takových jazyků se liší, ale existují určité charakteristiky, které řada z nich sdílí. Obecně je přeložený spustitelný soubor složen ze dvou částí: samostatného interpretu pro daný jazyk a skriptu, který je často v přeložené formě (dále jako bajt kód), který je uložen v nějaké datové sekci souboru. Pro formát PE to bývá resource sekce. Pro ztížení dekompilace je bajt kód často zkomprimován a zašifrován. Následné dešifrování a dekomprese je vykonaná programem před spuštěním skriptu. Konkrétní příklad překladače, který takovým způsobem operuje, je Aut2Exe[11]. Kromě zapouzdření skriptu do spustitelného souboru se v kontextu těchto skriptovacích jazyků často využívá oddělení interpretu a škodlivého skriptu do oddělených souborů. Samotný interpret je totiž neškodný a nebudí

[9] https://www.autoitscript.com/site/
[10] https://www.autohotkey.com/
[11] https://www.autoitscript.com/autoit3/docs/intro/compiler.htm

pozornost bezpečnostních systémů a samotné skripty nejsou detekovatelné antivirovými systémy, což umožňuje se snadno vyhnout statické analýze.

I když jsou tyto skriptovací jazyky méně populární než například jazyk Python, nejsou v reálném světě vzácné a stávají se čím dál komplexnější. Právě díky své nízké popularitě jsou tyto jazyky potencionální výhodou pro vývojáře malwaru, protože pro analýzu takových programů existuje méně nástrojů než pro jiné populární jazyky. Na druhou stranu jsou kvůli své podstatě jednoduše dekompilovány a výsledek dekompilace obsahuje spoustu řetězců, které můžou odhalit podstatu programu. I když je možno tyto skripty obfuskovat, většina těchto obfuskací se dá rychle obejít. Autoři malwaru jsou ale čím dál kreativnější a dále vyvíjí své techniky pro skrytí svého účelu.

Zjištění, zda spustitelný soubor obsahuje přeložený skript, a případná extrakce tohoto skriptu, je přínosným rozšířením projektu RecDec, jelikož použití automatizačních skriptů při tvorbě malwaru je čím dál více populární.

Kapitola 5

Návrh vybraných možností extrakce

V předchozí kapitole 4 byly uvedeny nové možnosti rozšíření statické analýzy nástroje FileInfo. Tato kapitola se věnuje těm nepřínosnějším z nich, pro které je vytvořen návrh jejich požadované funkcionality a integrace do nástroje FileInfo.

Pro výběr z navržených extrakcí proběhla konzultace s vedoucím ve společnosti Avast a s týmy, které vyvíjejí nástroje uvedené v sekci 3.2 a využívají nástroj FileInfo. Dospělo se k závěru, že PE formát je stále nejrozšířenější a vylepšení jeho extrakce bude mít největší efekt. Zároveň má PE formát standardizovaný způsob podepisovaní, který je bohatý na kvalitní informace a je nebezpečné tyto informace zanedbávat. Proto bylo vybrané rozšíření pro extrakci Authenticode podpisů. Dalším závěrem je, že import hash je velmi využívaným prvkem v rámci formátu PE a mít jej i pro formát ELF, který roste na popularitě, je velmi užitečné. Navrhnutý algoritmus telfhash je zároveň používaný například službou VirusTotal, která agreguje obrovské množství souborů, ve kterých je možné podle tohoto hashe hledat podobné soubory. Druhým rozšířením je tedy přidání výpočtu import hashe pro formát ELF.

5.1 Rozšíření analýzy Authenticode formátu

Existující analýza Authenticode formátu se v rámci programu FileInfo nachází v knihovně `fileformat`. Před začátkem této práce uměla tato knihovna extrahovat informace o certifikátech podpisu a také PKCS #9 spolupodpisu, které jsou uložené spolu s certifikáty PKCS #7 podpisu. Kromě této extrakce byla implementace schopná provést verifikaci PKCS #7 objektu.

Authenticode obsahuje další důležité informace, jako jsou data obsažená v různých členech PKCS #7 formátu a další vnořené informace v neověřených atributech (unauthenticatedAttributes) struktury SignerInfo. Tyto informace byly již zmíněny v sekci 4.1.

Tato nová implementace bude kompletně nahrazovat původní implementaci s cílem vytvořit od základu lepší a robustnější řešení. Kromě extrakce informací, které byla schopná provést původní implementace, bude nová implementace schopná získat vnořené Authenticode podpisy a Microsoft spolupodpisy s jejich časem podpisu. Rozšíření bude verifikovat obsah podpisů a spolupodpisů, k tomu bude potřeba přidat extrakci různých atributů v těchto podpisech a spolupodpisech, které jsou důležité pro verifikaci.

Popis jednotlivých atributů lze nalézt ve specifikaci PKCS #7 formátu [10] nebo ve specifikaci Authenticode formátu [19]. Následuje podrobný seznam, který obsahuje atributy, které budou cílem extrakce:

- Verze (version)
- Hashovací algoritmus (digestAlgorithms)
- Certifikáty (certificates)
- Obsah contentInfo
  - ◇ Hash PE souboru (digest)
  - ◇ Hashovací algoritmus (digestAlgorithm)
- Obsah SignerInfo:
  - ◇ Verze (version)
  - ◇ Hashovací algoritmus (digestAlgorithm)
  - ◇ Šifrovací algoritmus (digestEncryptionAlgorithm)
  - ◇ Podpis (encryptedDigest)
  - ◇ Řetězec certifikátů
  - ◇ Ověřené atributy (authenticatedAttributes)
    - ∗ Typ obsahu (contentType)
    - ∗ Hash obsahu (messageDigest)
    - ∗ SpcSpOpusInfo
  - ◇ Neověřené atributy
    - ∗ PKCS #9 spolupodpis
    - ∗ Microsoft spolupodpis
    - ∗ Authenticode podpis
- Obsah spolupodpisů:
  - ◇ Verze (version)
  - ◇ Hashovací algoritmus (digestAlgorithm)
  - ◇ Šifrovací algoritmus (digestEncryptionAlgorithm)
  - ◇ Podpis (Encrypted Digest)
  - ◇ Řetězec certifikátů
  - ◇ Ověřené atributy (authenticatedAttributes)
    - ∗ Typ obsahu (contentType)
    - ∗ Čas podpisu (signingTime)
    - ∗ Hash podpisu SignerInfo (messageDigest)

Ne všechny tyto uvedené informace budou dostupné na výstupu nástroje FileInfo. Mnohé z nich jsou důležité především pro ověření správnosti podpisu. Analýza informací obsažených v X509 certifikátech v řetězcích bude převzata z původní implementace, která již tuto funkcionalitu obsahuje.

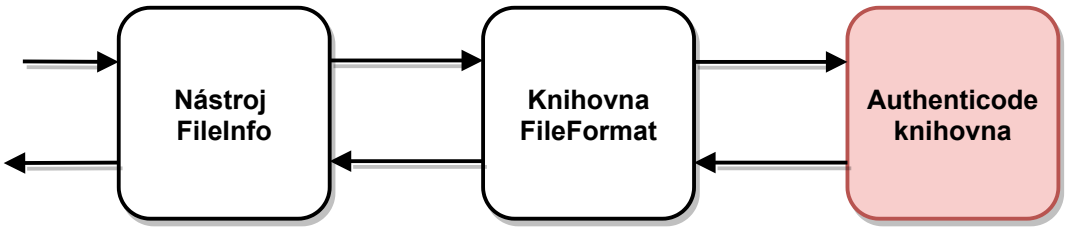
Kontrola správnosti se dá shrnout do několika pravidel, rozdělených podle jednotlivých částí Authenticode objektu. Pokud je některé z těchto pravidel porušené, celý podpis může být kompromitován. Jednotlivé části na sebe můžou odkazovat, takže se některá pravidla vyskytují vícekrát, ale pro přehlednost jsou uvedena na obou stranách vztahu. Pravidla pro ověřování podpisu jsou následující:

- Pravidla pro PKCS #7 SignedData:
  - ◇ Verze (version) má hodnotu 1.
  - ◇ Kolekce hashovacích algoritmů (digestAlgorithms) obsahuje pouze jeden algoritmus, který se shoduje s hashovacím algoritmem v SignerInfo.
  - ◇ Obsah (contentInfo) má typ SpcIndirectData a obsahuje hash PE souboru, který odpovídá reálnému hashi souboru.
  - ◇ SignerInfos obsahuje právě jednu strukturu SignerInfo.
- Pravidla pro SignerInfo:
  - ◇ Verze (version) má hodnotu 1.
  - ◇ Obsahuje sériové číslo certifikátu a jméno vydavatele (issuerAndSerialNumber), který je uložen v PKCS #7 SignedData certifikátech.
  - ◇ Hashovací algoritmus (digestAlgorithm) se shoduje s hashovacím algoritmem uvedený v PKCS #7 SignedData.
  - ◇ Ověřené atributy (authenticatedAttributes) obsahují následující položky:
    - ∗ contentType, který má hodnotu PKCS #9 MessageDigest OID.
    - ∗ messageDigest, který obsahuje hash obsahu v PKCS #7 SignedData.
    - ∗ spcSpOpusInfo objekt.
  - ◇ Dešifrovaný podpis (encryptedDigest) veřejným klíčem certifikátu podepisovatele je stejný jako výsledek hashovací funkce specifikované v položce digestAlgorithm nad ověřenými atributy.
- Pravidla pro spolupodpis:
  - ◇ Splňuje stejná pravidla jako SignerInfo kromě těch, které se týkají ověřených atributů, hashovacího algoritmu a verze.
  - ◇ Ověřené atributy (authenticatedAttributes) obsahují následující položky:
    - ∗ ContentType, který má hodnotu PKCS #7 Data.
    - ∗ SigningTime, který obsahuje čas podpisu.
    - ∗ MessageDigest, který obsahuje hash položky encryptedDigest v SignerInfo.

Při porušení jakéhokoliv pravidla z tohoto seznamu se bude implementace snažit pokračovat v další analýze, aby získala všechny možné informace z Authenticode podpisu. Není žádoucí zastavovat analýzu při prvním porušení některého pravidla, protože to může vést k extrahování jen zlomku informací. Tím by se přišlo o cenná data, jelikož neplatný

podpis se u škodlivého kódu vyskytuje poměrně často. Zároveň je specifikace Authenticode formátu zastaralá (rok 2008) a obsahuje několik neaktuálních informací a chyb. Toto vede k tomu, že implementace ověřování podpisu přímo od společnosti Microsoft bere některé podpisy jako validní, i když odporují jejich staré specifikaci. Všechny tyto porušení, které se při analýze najdou, budou výstupem z knihovny společně s analyzovanými informacemi z podpisu a bude na uživateli, jak tyto informace využije.

Celé rozšíření by mělo být koncipované jako samostatná knihovna, jak je zobrazené na obrázku 5.1. Což umožní snadněji udržovat kód, oddělit ho od zbytku nesouvisejících funkcí a struktur. Dále to umožní využívat analýzu Authenticode podpisu samostatně, odděleně od zbytku nástroje FileInfo, tím se dá vyhnout analyzování informací, které nejsou potřeba.



Obrázek 5.1: Návrh Authenticode analýzy jako samostatné knihovny

Authenticode se skládá z různých standardních kryptografických objektů. Pro jejich analýzu využívala původní implementace knihovnu OpenSSL. OpenSSL je napsaná v jazyce C a poskytuje řadu kryptografických funkcí a nástrojů pro práci s těmito objekty. OpenSSL budu využívat i já pro usnadnění analýzy Authenticode formátu.

5.2 Tvorba import hashe pro ELF formát a implementace telfhash algoritmu

Implementace výpočtu hashe importovaných symbolů existovala pouze pro PE a Mach-O souborové formáty. Tvorba import hashe je specifická pro každý souborový formát, protože obsažené informace o symbolech a způsob jejich uložení se liší mezi různými formáty. PE formát například asociuje každý importovaný symbol s určitou knihovnou. To pro ELF soubor neplatí, a proto je potřeba přidat vlastní algoritmus pro tvorbu import hashe pro tento formát.

Při uvedení této extrakce v sekci 4.4 byl ukázán algoritmus telfhash. Po hlubším zkoumání jsem zjistil, že i přes to, že je telfhash přezdíván jako import hash pro ELF formát, ve skutečnosti pracuje se všemi symboly, a ne pouze importovanými. Telfhash je již používaný v kyberbezpečnostních firmách ve světě a funguje na stejných principech jako import hash. Z tohoto důvodu je to stále chtěné rozšíření, ale bude doplněn také generickým import hashem, který bude fungovat pouze nad importovanými symboly.

Originální implementace algoritmu telfhash (od autora tohoto algoritmu) je napsaná v jazyce Python, open-source a dostupná na platformě GitHub[1]. Cílem je, aby naše implementace algoritmu telfhash generovala identický výsledek jako originální implementace z důvodu kompatibility.

Koncept telfhash algoritmu vypadá následovně:

1. Provede se extrakce všech symbolů.

[1] https://github.com/trendmicro/telfhash

2. Z těchto symbolů se některé vyfiltrují.

3. Jména symbolů se převedou na malá písmena a seřadí, aby hash nebyl závislý na pořadí symbolů.

4. Jména se umístí za sebe, přičemž jsou oddělena čárkami, a předají TLSH hashovací funkci.

Pro extrakci symbolů z ELF formátu používá RetDec open-source knihovnu ELFIO[2]. ELFIO je knihovna složená pouze z hlavičkových souborů v jazyce C++ a slouží k jednoduchému čtení nebo vytváření spustitelných souborů v ELF formátu.

Mezi extrahované symboly patří ty, které jsou uložené v sekci typu `SHT_DYNSYM` nebo případně v `SHT_SYMTAB`, s tím, že pokud jsou oba typy sekcí k dispozici, má přednost typ `SHT_DYNSYM`. Dále symbol musí představovat funkci (tudíž mít typ `STT_FUNC`), mít globální vazbu, která je určena atributem `STB_GLOBAL`, a nakonec ještě mít základní viditelnost `STV_DEFAULT`.

Protože linuxových systémů, které ELF formát využívají, je spousta, existují na velkém množství architektur a využívají široké spektrum překladačů, je potřeba se vyhnout co nejvíce specifickým symbolům. Důvodem je, aby hash programu, který je přeložen různými překladači na různé systémy a různé architektury, byl ideálně stejný ve všech těchto variantách. Proto se z extrahovaných symbolů odstraňují symboly splňující následující podmínky, které jsou dané v originální implementaci telfhash algoritmu:

- Jejich název začíná na „mem“ nebo „str“, protože tyto řetězcové funkce jsou často odlišné pro různé architektury.
- Jejich název končí na „64“, což značí, že se jedná o funkci, která je specifická pro architekturu x86-64.
- Jejich název začíná na podtržítko nebo tečku, což implikuje interní funkci knihovny nebo funkci specifickou pro překladač.
- Jejich název je jeden z následujícího seznamu symbolů:
  - `__libc_start_main`
  - `main`
  - `abort`
  - `cachectl`
  - `cacheflush`
  - `puts`
  - `atol`
  - `malloc_trim`

Telfhash používá hashovací funkci TLSH, která se liší od ostatních používaných hashovacích funkcí tím, že zachovává podobnost (Locality-sensitive hashing) [23]. TLSH hashe dvou podobných řetězců si budou také podobné, na rozdíl od kryptografických funkcí jako jsou algoritmy typu SHA nebo MD, které mají pro změnu jediného bytu na vstupu naprosto

[2] https://github.com/serge1/ELFIO

odlišný výsledný hash. TLSH má existující open-source implementaci v jazyce C++[3]. Díky možnosti open-source licence bude integrována tato existující implementace do projektu RetDec kvůli jednoduchosti a zaručení korektních výsledků.

Import hash bude fungovat velmi obdobně jako telfhash:

- Provede se extrakce všech importovaných symbolů.
- Jména symbolů se převedou na malá písmena a seřadí, aby hash nebyl závislý na pořadí symbolů.
- Názvy, které jsou oddělené čárkami, se vloží za sebe do jednoho řetězce a předají hashovací funkci.

Hlavní rozdíl mezi telfhashem a import hashem tedy je, že import hash funguje pouze nad importovanými symboly, neprovádí žádné speciální filtrování symbolů a výsledný řetězec se předá více hashovacím funkcím: CRC32, SHA256, MD5 a TLSH.

[3] https://github.com/trendmicro/tlsh

Kapitola 6

Implementace rozšíření extrakcí informací

Obsahem kapitoly je implementace navrhnutých rozšíření. Jedná se o rozšíření analýzy Authenticode podpisů a přidání tvorby import hashe pro soubory v ELF formátu a implementaci algoritmu telfhash.

Knihovna `fileformat`, pod kterou spadají všechny tyto rozšíření, je napsaná v jazyce C++, dle specifikace ISO C++ 17. Z tohoto důvodu popisované rozšíření budou používat možnosti této specifikace.

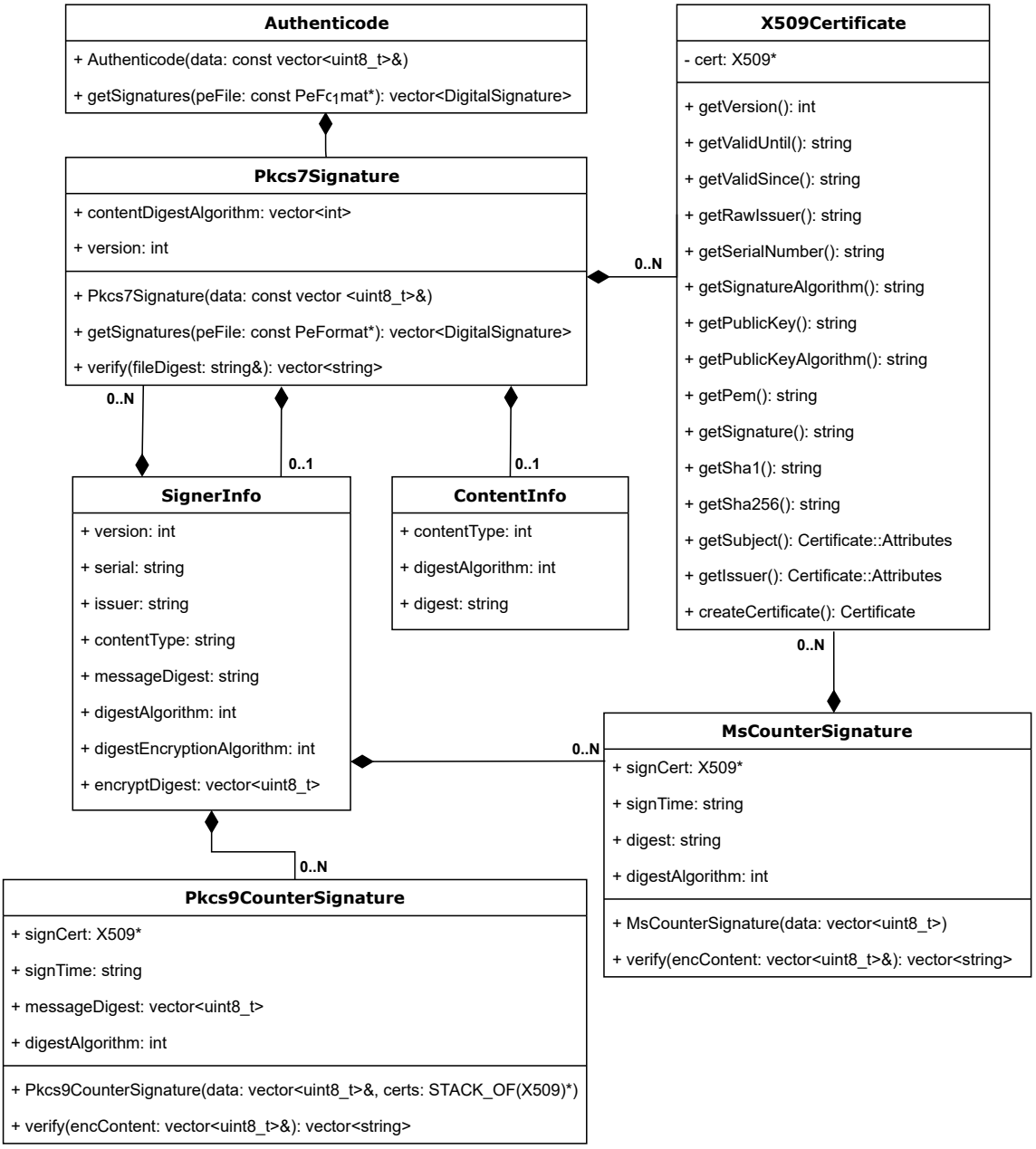
Další sekce popisují způsob implementace jednotlivých rozšíření, jejichž návrh se nachází v předchozí kapitole 5.

6.1 Rozšíření analýzy Authenticode podpisů

Původní implementace analýzy Authenticode podpisů se nacházela v souboru `pe_format.cpp`. Ta obsahovala funkci `loadCertificates()`, která z PE souboru získala informace v tabulce certifikátů a z ní případné data obsahující Authenticode podpis, které následně analyzovala. Výsledkem této funkce bylo vytvoření instance třídy `CertificateTable`, která obsahovala extrahované informace o podpisu.

Zmíněná implementace funkce `loadCertificates()` je nahrazena voláním funkcí Authenticode knihovny. Z původní implementace je zachována analýza informací z certifikátů X509 a přesunuta do Authenticode knihovny. Data obsahující Authenticode podpis jsou předána knihovně jako kolekce bajtů do konstruktoru třídy `Authenticode`, která se stará o kompletní extrakci informací z podpisu a jeho validaci ve formě konstrukce instance této třídy. Způsob, jakým se tyto informace získají zpět z instance třídy, je skrz metodu `Authenticode::getSignatures()`. Ta vrací kolekci instancí třídy `DigitalSignature`, která obsahuje všechna zajímavá data z podpisu a seznam chyb, které se během analýzy nebo verifikace vyskytly. Třída `CertificateTable` byla upravena, aby držela kolekci `DigitalSignature` objektů.

Další části sekce dále přibližují proces extrakce a verifikace informací z Authenticode podpisu.



Obrázek 6.1: Třídní diagram Authenticode knihovny

6.1.1 Analýza informací z podpisu

Na obrázku 6.1 lze vidět zjednodušený třídní diagram celé Authenticode knihovny. Způsob zpracování jednotlivých tříd bude předmětem této sekce.

Vstupní bod pro analýzu je třída `Authenticode`. Tato třída vytváří abstrakci nad celým Authenticode podpisem a umožňuje získat všechny důležité informace o podpisech a spolupodpisech. Uvnitř konstruktoru probíhá transformace těchto bajtů na interní reprezentaci PKCS #7 podpisu. Při transformaci se používají funkce OpenSSL knihovny. Funkce s předponou „d2i“ se využívají pro převod DER dat na interní reprezentaci. Zkratka „d2i“ znamená „DER to internal“ neboli konverze serializovaných DER dat na OpenSSL struk-

turu. Z vytvořené interní reprezentace PKCS7 objektu lze lehce získat atributy podpisu. Je důležité zmínit, že tvorba těchto OpenSSL objektů alokuje paměť. Pro zaručení korektního uvolnění se využívá tzv. chytrých ukazatelů.

Zpracování jednotlivých částí PKCS #7 SignedData objektu probíhá následovně:

- Verze (version) obsažena v PKCS7 objektu je uložena jako celé číslo.
- Hashovací algoritmy, které mají typ `X509_ALGOR` jsou uložené v PKCS7 objektu jako OpenSSL kolekce. Jednotlivé `X509_ALGOR` objekty se vyjmou z kolekce a uloží ve formě jejich NID (unikátní identifikátor v rámci OpenSSL knihovny), což je pouhé celé číslo. V této podobě se s nimi v OpenSSL snadno pracuje.
- Certifikáty jsou uložené v `PKCS7` objektu, jako kolekce `X509` objektů. Jednotlivé X509 objekty se převedou do knihovní abstrakce certifikátu `X509Certificate` a uloží do vektoru těchto objektů. Třída `X509Certificate` se skládá z tohoto OpenSSL X509 objektu a metod, které z tohoto OpenSSL objektu umožňují získat všechny potřebné informace. Jak bylo zmíněno v úvodu sekce, implementace těchto metod byly převzaty z velké části z původní implementace, která je popsána v diplomové práci Marka Milkoviče [21].
- ContentInfo, které má ve formátu Authenticode typ SpcIndirectDataContent, je převedeno z binární reprezentace do interní reprezentace. Tento typ ASN.1 objektu není úplně typický, proto pro něj neexistují funkce v knihovně OpenSSL. Knihovna OpenSSL ale umožňuje vytvořit vlastní struktury pomocí knihovních maker, pro které pomocí dalších maker umožňuje vygenerovat funkce pro serializaci a deserializaci. Tento objekt a další speciální Authenticode objekty, které nejsou součástí samotné OpenSSL, jsou vytvořeny pomocí těchto maker. Z tohoto zpracovaného objektu je získán hash PE souboru a hashovací algoritmus. PE hash je převeden do hexadecimální podoby a uložen do řetězce. Hashovací algoritmus se uloží ve formě jeho NID.
- SignerInfo se získá z OpenSSL `PKCS7` objektu přes funkci. Z tohoto SignerInfo objektu se získá verze, hashovací a šifrovací algoritmus stejným způsobem jako v předchozích případech. Certifikát autora podpisu se získá funkcí `PKCS7_get0_signers`, která ho vyhledá podle jména vydavatele a sériového čísla v kolekci PKCS7 certifikátů. Podpis (encryptedDigest) se uloží v podobě vektoru bajtů. Sériové číslo a jméno vydavatele je uloženo ve formě řetězců. Dále následuje procházení kolekcí všech ověřených a neověřených atributů. Ty jsou uložené jako kolekce generických objektů. Atributy v těchto kolekcích jsou vyhledávané podle jejich ASN.1 OID (object identifier). Z ověřených atributů se získává hash, SpcSpOpusInfo a typ obsahu. Z neověřených atributů lze extrahovat vnořené Authenticode podpisy, PKCS #9 spolupodpis a Microsoft spolupodpis. Případné vnořené podpisy jsou rekurzivně zpracované.

Další komplexní objekty analýzy jsou PKCS #9 a Microsoft spolupodpisy. Informace obsažené v obou z nich jsou velmi podobné, ale způsob, jakým jsou uložené, je velmi odlišný.

- PKCS #9 formát má stejnou strukturu jako objekt SignerInfo, jehož analýza je popsaná výše, ale liší se v obsažených ověřených atributech. Místo objektu SpcSpOpusInfo obsahuje UTC čas podpisu. Uložený čas je převeden do textové podoby a uložen do řetězce. Dle specifikace formátu je možné, aby obsahoval další PKCS #9 spolupodpisy jako neověřený atribut. I když je tato možnost implementovaná, nepodařilo se mi najít žádný Authenticode podpis, který by spolupodpis spolupodpisu využíval.

- Microsoft spolupodpis je o něco složitější než PKCS #9. Jak již bylo popsáno v sekci 2.4.1, tento typ spolupodpisu je uložen jako PKCS #7 struktura, která má vlastní kolekci certifikátů. Na rozdíl od PKCS #9 podpisu, který má své certifikáty uložené v nadřazeném PKCS #7 objektu. Dále obsahuje `TSTInfo` strukturu, které představuje informace o časovém razítku a SignerInfo, který je analyzováno stejně jako v předchozích odstavcích. Z `TSTInfo` jsou získané dvě důležité informace: hash spolupodepisovaného podpisu a čas podpisu.

Jak bylo řečeno, všechna výše uvedena analýza probíhá během konstrukce instance třídy `Authenticode`. Tyto analyzované informace jsou dostupné skrz funkci `getSignatures()`. Ta postupuje následovně:

Vytvoří objekt typu `DigitalSignature`, který představuje Authenticode podpis. Do něho jsou doplněné dostupné informace o podpisu. Je možné, že jakákoliv z informací bude chybět, pokud se tak stane, jsou korespondující atributy objektu prázdné. Podpis je verifikován a případná upozornění jsou uloženy do výstupního objektu. Dále se prochází všechny spolupodpisy obsažené v podpisu, které jsou také následně verifikované a jejichž informace jsou doplněné do výstupní struktury. Ještě probíhá průchod všemi vnořenými podpisy. Nad každým vnořeným podpisem se volá rekurzivně funkce `getSignatures()`. Výsledek tohoto rekurzivního volání je sloučen s výstupním objektem, který je poté vrácen z funkce.

6.1.2 Verifikace

Obsahem verifikace bude kontrola, zda jsou dodržena pravidla, která jsou sepsána v jednotlivých bodech v sekci 5.1. Všechna porušení pravidel v podpisu se objeví na výstupu jako kolekce vět, které popisují jaká chyba se vyskytla při verifikaci podpisu.

Verifikace je implementována v metodách `verify()` jednotlivých typů podpisů:

- `Pkcs7Signature`
- `Pkcs9CounterSignature`
- `MsCounterSignature`

Jak již bylo zmíněno, verifikace se provádí nad jednotlivými objekty při exportování informací o podpisu ve funkci `getSignatures()`.

Verifikace PKCS #7 podpisu

Při verifikaci je ověřena existence a hodnota jednotlivých atributů dle zmíněných pravidel. Zajímavější částí verifikace je ověření podpisu. K tomu se používá OpenSSL funkce `PKCS7_signatureVerify()`. Ta porovná hash PKCS #7 objektu s hashem uloženým v SignerInfo a pokud jsou tyto dva hashe stejné, pokračuje ověřením integrity samotného SignerInfo. Zašifrovaný hash (encryptDigest) v SignerInfo se dešifruje pomocí veřejného klíče v certifikátu autora. Získaný hash porovná s vypočítaným hashem ověřených atributů. Pokud jsou i tyto dva hashe stejné, podpis je úspěšně ověřen. Funkce `PKCS7_signatureVerify()` vyžaduje jako argument vypočítaný hash PKCS #7 objektu ve formě struktury `BIO`. Tato funkce není zdokumentovaná, pro pochopení a ověření její funkcionality jsem čerpal ze zdrojového kódu OpenSSL knihovny, který je možný nalézt na webové službě Github[1].

[1] https://github.com/openssl/openssl/

Verifikace PKCS #9 spolupodpisu

Verifikace PKCS #9 spolupodpisu je o něco složitější, protože OpenSSL nenabízí funkce pro verifikaci PKCS #9 formátu, jako nabízí pro PKCS #7 formát. Princip je ale stejný jako pro PKCS #7 a dá se inspirovat z implementace funkce `PKCS7_signatureVerify()`[2].

Prvně je ověřena existence potřebných atributů. Dále se ověří, zda sedí podpis (encryptedDigest) autentizovaných dat ve spolupodpisu. Pro toto ověření je nutné vypočítat hash ověřených atributů. Ten je porovnán s podpisem, který je dešifrován pomocí veřejného klíče v certifikátu autora podpisu. Během implementace se objevila zajímavá výjimka u několika spolupodpisů. Některé spolupodpisy měly podpis jako zašifrovaný hash ověřených atributů a některé obsahovaly zašifrovanou celou strukturu `DigestInfo` v DER kódovaní, která tento hash obsahuje. Tato situace je řešena porovnáním délky dešifrovaného podpisu. Pokud je délka jiná, než by měl výsledný hash mít, předpokládá se, že jde o celý `DigestInfo` objekt, jinak že jde o samotný hash.

Nakonec je potřeba ještě ověřit zda je spolupodpis správný a tedy, že hash obsahu (messageDigest), který je uložen v ověřených atributech, je shodný s hashem podpisu (encryptedDigest) ve spolu-podepsaném podpisu.

Verifikace Microsoft spolupodpisu

Verifikace Microsoft spolupodpisu obsahuje určité nejistoty, protože k tomuto formátu chybí jakákoliv oficiální dokumentace. Ale jak bylo popsané v předchozí sekci o analýze, spolupodpis je složen z PKCS #7 podpisu, který obsahuje TSTInfo strukturu. Pro ověření spolupodpisu se používá stejný proces jako pro PKCS #9. Získaný messageDigest atribut je porovnán s hashem encryptedDigest atributu nadřazeného podpisu. Pro ověření obsahu TSTInfo se používají existující OpenSSL funkce.

6.2 Přidaní tvorby import hashe pro ELF formát a telfhashe

Knihovna `fileformat` používá pro zpracování formátu ELF knihovnu `ELFIO`[3]. Existující extrakce symbolů ze sekce se nachází ve funkci `ElfFormat::loadSymbols()`, která využívá třídu `symbol_section_accessor` z ELFIO knihovny pro získání všech symbolů z ELF souboru. Tato funkce je volaná nad každou sekcí obsahující symboly, včetně těch obsažených v dynamických segmentech. Funkce třídí symboly do tří skupin:

- Importované symboly
- Exportované symboly
- Všechny symboly

Všechny importované symboly jsou uloženy do instance třídy `ImportTable`, která představuje abstrakci tabulky importovaných symbolů. Tato abstrakce se využívá pro formáty PE, Mach-O a ELF. Třída `ImportTable` se také stará o výpočet import hashe nad všemi symboly, které obsahuje. Protože algoritmus pro výpočet Mach-O a PE import hashe se nedá uplatnit nad ELF formátem kvůli chybějící asociaci symbolu s knihovnou, bylo potřeba změnit tento výpočet pro ELF formát. Z možností jak změnit předchozí implementaci jsem vybral využití polymorfismu, protože dovoluje změnit tento výpočet bez velkého zásahu do

[2] https://github.com/openssl/openssl/blob/master/crypto/pkcs7/pk7_doit.c#L1064
[3] https://github.com/serge1/ELFIO

aktuální struktury knihovny a dá se jednoduše použít v jazyce C++. Metoda `computeHashes()`, která se ve třídě `ImportTable` používá pro výpočet import hashe, se definovala jako virtuální kvůli možnosti její redefinice v nově přidané třídě `ElfImportTable`, která dědí z třídy `ImportTable`. Ve funkci `ElfFormat::loadSymbols()` se nahradila instance třídy `ImportTable` za instanci nové třídy `ElfImportTable`.

Ve třídě `ElfImportTable` se redefinovala již zmíněna virtuální metoda pro výpočet hashe `computeHashes()`. Ve funkci se projdou všechny importované symboly v tabulce, všechna písmena v jejich názvu se převedou na nižší variantu a tento název se uloží do kolekce. Tato kolekce názvů se poté lexikograficky seřadí. V tomto seřazeném pořadí se následně symboly přidávají do jednoho řetězce, kde je každý symbol oddělen od ostatních čárkou. Z řetězce je na konci funkce vypočten import hash ve variantách CRC32, MD5, SHA256 a TLSH hashe. Tyto varianty jsou uloženy do objektu `ElfImportTable`, ze kterého se poté předají tyto informace na výstup.

Výpočet telfhashe je zakomponován do samotné třídy `ElfFormat`. Ve funkci `ElfFormat::loadSymbols()` jsou vybrány všechny symboly pro algoritmus telfhash. Mezi tyto symboly patří ty, které odpovídají funkci, mají globální vazbu a základní viditelnost. Tyto informace o symbolu jsou získané z knihovny ELFIO. Všechny vybrané symboly jsou uloženy do kolekce instance třídy `ElfFormat`.

Způsob volání funkce `ElfFormat::loadSymbols()` komplikuje získávání telfhash symbolů. Funkce je volaná nad každou sekcí, která obsahuje symboly, ale také i nad dynamickými segmenty, které obsahují informace o symbolech. Se symboly ze segmentů původní telfhash implementace ale nepočítá, a proto je potřeba tyto symboly ignorovat. Toho je dosaženo pomocí vyhledání podřetězce „dynamic_“ v názvu sekce, který do jeho názvu vkládá knihovna `fileformat`, jestliže se jedná o sekci získanou z dynamického segmentu. Pokud je podřetězec nalezen, symboly jsou ignorovány.

Dalším problémem je zachování priority pro sekci typu `DYNSYM`. Zde jsou 3 možné situace:

- Pokud byla již funkce `ElfFormat::loadSymbols()` volána nad sekcí typu `DYNSYM`, jsou další sekce ignorovány.
- Pokud byly načteny symboly ze sekce typu `SYMTAB`, ale poté byla funkce zavolána nad další sekcí typu `SYMTAB`, nebo nejsou načteny ještě žádné symboly, jsou případné předchozí načtené symboly zahozeny a jsou načteny symboly z nové sekce.
- Posledním případem je, že se načte sekce typu `DYNSYM`. V tom případě se zahodí všechny předtím načtené symboly a nastaví se člen `ElfFormat::telfhashDynsym` na `true`. Člen `telfhashDynsym` uchovává stav přes několik možných volání této funkce a říká jestli byly již načteny symboly ze sekce typu `DYNSYM`.

Po získání symbolů se volá funkce pro výpočet telfhashe. Funkce prochází všechny symboly a vylučuje ty, které odpovídají filtrovacím pravidlům sepsané v návrhu 5.2.

Filtrovací logiku implementuje funkce `isSymbolExcluded()`, která říká, zda by měl být symbol vyloučen. Pro filtrování na základě předpon nebo přípon se používají regulární výrazy, které jsou v C++ dostupné ve standardní knihovně `regex`. Pokud se našla shoda mezi regulárním výrazem a názvem, symbol je vyloučen z výpočtu. Dále je vytvořena množina všech konkrétních symbolů, které jsou odstraněny z výpočtu. Pro rychlé vyhledávání, které probíhá nad každý symbolem, jsou symboly uložené do instance třídy `std::unordered_set`, což je kolekce, která umožňuje rychlé vyhledávání. Pokud je symbol nalezen v této kolekci, je také vyřazen z výpočtu. Všechny symboly pro kalkulaci hashe jsou po filtrování

převedené na malé písmo a uložené do kolekce názvů, která je následně lexikograficky seřazena. V tomto seřazeném pořadí se symboly přidávají do jednoho řetězce, kde je každý symbol oddělen od ostatních čárkou. Z řetězce je na konci funkce vypočten telfhash pomocí hashovací funkce TLSH.

Implementace TLSH algoritmu byla do projektu RetDec přidána z existujícího open-source projektu[4]. Všechny důležité hlavičkové a zdrojové soubory byly nakopírované do projektu RetDec a integrované pomocí konfiguračních souborů programu CMake[5], který RetDec používá pro překlad. To zpřístupnilo třídu Tlsh, která obsahuje všechny metody potřebné pro výpočet TLSH hashe. Metoda `Tlsh::update()` slouží k přidání bajtů dat k hashování, metoda `Tlsh::final()` k výpočtu hashe. K získaní hashe je použita metoda `Tlsh::getHash()`.

[4] https://github.com/trendmicro/tlsh
[5] https://cmake.org/

Kapitola 7

Testování

Tato kapitola se zabývá testováním jednotlivých rozšíření. Kvůli své podobě, kde se vstupy obtížně simulují a je mnohem příjemnější je testovat nad binárními soubory, se testují jako celek pomocí integračních testů. Integrační testování má za účel zjistit, zda naprogramovaná funkcionalita odpovídá požadavkům, a že se software jako celek chová tak, jak se od něj očekává. Tyto integrační testy budou sloužit v budoucnosti i jako regresní testy pro případné odhalení chyb, které mohou nastat při dalším vývoji v budoucnosti, a stanou se součástí automatických testů, které běží při každé úpravě kódu v projektu RetDec.

Pro psaní integračních testů má RetDec vlastní aplikační rámec napsaný v jazyce Python[1], který ulehčuje a sjednocuje způsob psaní testů. Zpřístupňuje jednoduché spouštění nástrojů v projektu RetDec a otestování jejich výstupu. Konkrétní příklad testu lze vidět v následujících sekcích.

7.1 Testování analýzy Authenticode podpisů

Sekce popisuje způsob tvorby integračních testů a vyhodnocování přínosu rozšíření Authenticode extrakce ve srovnání s předchozí implementací.

7.1.1 Integrační testy

Integrační testy jsou napsány s použitím již zmíněného existujícího aplikačního rámce. Cílem integračních testů je zajistit správnost fungování rozšíření testováním programu jako celku. Konkrétně tím, že pro vybrané vstupní soubory tiskne program na výstup správné hodnoty.

Protože analýza Authenticode formátu již v určité podobě existovala před začátkem práce, projekt obsahoval několik testů pro předchozí implementaci. Tyto původní testy pokrývaly již široké spektrum situací, krajních případů a dokonce i prvky, které je schopna extrahovat teprve nová implementace. Formát výstupu předchozí implementace je ale naprosto odlišný od nového a bylo pro něj potřeba upravit všechny tyto předchozí testy. Původní testy pokrývaly široké spektrum situací. Existující testy ulehčily práci s psaním dalších testů a také odhalily chyby v nové implementaci.

Správnost informací v testovaných případech jsem ověřil pomocí knihovny LIEF[2] a webové stránky VirusTotal[3]. Na ukázce 7.1.1 lze vidět příklad jednoho testovacího příkladu.

[1] https://github.com/avast/retdec-regression-tests-framework
[2] https://lief.quarkslab.com/
[3] https://www.virustotal.com

```
class Test2(Test):
    settings = TestSettings(
        tool='fileinfo',
        input='avgcfgex.ex',
        args='--json --verbose'
    )

    def test_certificates(self):
        assert self.fileinfo.succeeded

        assert self.fileinfo.output
            ['certificateTable']['numberOfSignatures'] == 2

        self.assertEqual(
            len(self.fileinfo.output["certificateTable"]
            ['signatures'][0]["allCertificates"]), 5)
        self.assertEqual(self.fileinfo.output["certificateTable"]
            ['signatures'][0]['signer']['chain'][0]
            ["sha256"],
            "3B0ABE047D7E84F3BBD12B5E399BED55E4D7E9FCC3F629B8953A8C060EF6D746")
```

7.1.2 Porovnání vylepšení s předchozí implementací

Porovnání s předchozí implementací probíhalo analýzou obrovského množství souborů a měření dvou základních faktorů:

- Počet získaných podpisů.
- Počet získaných spolupodpisů.

Pro toto vyhodnocení bylo získáno přes dva miliony různých PE souborů, které obsahují data v tabulce certifikátů, kde jsou uloženy Authenticode podpisy. Přes sto tisíc z těchto dvou milionů vzorků je vyhodnoceno jako škodlivé. Pro účely tohoto testování byl napsán skript, který spustí předchozí verzi programu FileInfo a aktuální verzi s rozšířenou extrakcí Authenticode podpisů nad všemi spustitelnými soubory. Cílem bylo zjistit, kolik dalších informací rozšíření extrakce zachytilo. Toto porovnání bylo rozděleno na všechny programy a na malware. Výsledky nad malwarovými vzorky se nachází v tabulce 7.1 a výsledky nad všemi vzorky v tabulce 7.2. Jde vidět, že narůst získaných podpisů a spolupodpisů se nijak výrazně neliší mezi škodlivými a ostatními vzorky. Množství získaných podpisů narostlo o více než 18 % a počet spolupodpisů dokonce kolem 70 %. Tak velký nárůst u spolupodpisů je z důvodů, že rozšíření dokáže analyzovat další typ spolupodpisů, a také že spolupodpisy jsou častou součástí vnořených signatur.

	Před rozšířením	Po rozšíření	Nárůst o
Počet podpisů	110 130	130 182	18,2 %
Počet spolupodpisů	42 806	73 122	70 %

Tabulka 7.1: Výsledky vyhodnocení přínosu nad vzorky škodlivých programů.

	Před rozšířením	Po rozšíření	Nárůst o
Počet podpisů	1 999 651	2 377 174	18,8 %
Počet spolupodpisů	941 024	1 556 893	65,4 %

Tabulka 7.2: Výsledky vyhodnocení přínosu nad všemi podepsanými programy.

7.2 Testování tvorby telfhashe

Tato sekce se zaobírá testováním implementace telfhash algoritmu. Především porovnáváním originální implementace algoritmu telfhash[4] a vytvořené RetDec implementace, protože z důvodu kompatibility s existujícími systémy je důležité mít identické výsledky.

7.2.1 Testování správnosti implementace

Správnost zde znamená, že RetDec implementace tvoří stejný výsledek jako originální implementace algoritmu telfhash. Pro otestování správnosti byla provedena analýza nad skoro půl milionu vzorků ELF formátu. Toto testování nad velkým množstvím souborů odhalilo řadu krajních případů, pro které chování implementace nebylo správné a umožnilo tyto krajní případy opravit. Výsledek po opravení chyb v implementaci lze vidět v tabulce 7.3.

V tabulce jsou uvedené čtyři různé případy:

- Výsledky obou programů se shodují.
- Výsledky obou programů se neshodují, což je způsobeno tím, jaký název symbolu získává nástroj FileInfo z knihovny ELFIO. Občas se stává, že v názvu symbolu chybí takzvaný řetězec verze (například „foo@VER_1“, kde „foo“ je název symbolu a znaky za zavináčem je řetězec verze „VER_1“), zatímco originální implementace má symbol s tímto řetězcem verze. Tenhle případ se vyskytuje vzácně.
- Originální implementace nebyla schopná zpracovat soubor nebo se zasekla a došlo k vypršení časového limitu 20 sekund pro analýzu.
- RetDec implementace nebyla schopná zanalyzovat soubor nebo došlo k vypršení časového limitu.

Počet ELF souborů	464 527
Počet správných výsledků	463 795
Počet chybných výsledků	71
Chyba při analýze souboru v originální implementaci	590
Chyba při analýze souboru v nástroji FileInfo	71

Tabulka 7.3: Výsledek porovnání RetDec a originální telfhash implementace.

7.2.2 Integrační testy

Pro psaní integračních testů telfhash algoritmu se, stejně jako pro testování Authenticode rozšíření, použil zmíněný aplikační rámec[5]. Forma těchto testů je velmi triviální, pouze

[4] https://github.com/trendmicro/telfhash
[5] https://github.com/avast/retdec-regression-tests-framework

kontroluje, zda je vygenerovaný telfhash správný vůči původní implementaci. ELF soubory, které jsou testovány, byly vybrány z těch, které během testování správnosti, zmíněné v předchozí sekci 7.2.1, odhalili krajní případ způsobující chybu v implementaci.

Kapitola 8

Závěr

Tato práce se zabývala vylepšováním projektu RetDec, přesněji nástroje FileInfo, který se používá pro statickou analýzu spustitelných souborů.

V této práci byly rozebrány různé formáty spustitelných souborů (ELF, PE), jakým způsobem se podepisují binární soubory, jakých formátů se k tomuto účelů používá (Authenticode) a jak probíhá ověření takových podpisů. Také bylo uvedeno využití hashe symbolů ve spustitelných souborech, který je i přes svoji jednoduchost často používaným prvkem v rámci analýzy a shlukování škodlivých programů. Zkoumala se i možná analýza artefaktů různých programovacích jazyků (Go, AutoIT nebo AutoHotkey) nebo analýza takzvané Rich hlavičky.

Práce splnila nastavené cíle. Zkoumala různé možnosti rozšíření tohoto nástroje s cílem zdokonalit extrakci informací pro lepší analýzu škodlivých programů. Především z pohledu užitku pro společnost Avast, s jejíž spoluprací byla tato práce vytvořena. Každá z těchto možností byla prozkoumaná pro možný přínos při analýze škodlivého kódu. Po konzultaci se společností Avast byly vybrány podle největšího možného přínosu následující dvě rozšíření: vylepšení analýzy digitálních podpisů ve formátu Authenticode a výpočet hashe symbolů v souborech formátu ELF (import hash a telfhash).

Pro tyto dvě vybrané rozšíření extrakce informací byl vytvořen návrh pro implementaci. Rozšíření byla implementovaná v jazyce C++ do projektu RetDec a staly se již i součástí tohoto projektu. Rozšíření byla otestovaná na velkých množstvích škodlivých i čistých vzorků pro ověření stability jejich implementace a správných výsledků. Pro každé rozšíření bylo napsáno několik integračních testů, které se bude dále využívat i pro testování regresí těchto nových funkcí během budoucího vývoje.

Všechny nástroje, které v rámci společnosti Avast využívají nástroj FileInfo, budou těžit z vytvořených rozšíření. Kromě toho je projekt RetDec open-source a proto z těchto vylepšení může těžit celá komunita. Z těchto důvodů má moje práce dopad nejen na samotný nástroj FileInfo, ale i všechny ostatní nástroje, které ho používají. Přidané vylepšení umožní získat více informací nástrojem FileInfo, lepší shlukování programem Clusty, poskytovat více informací nástrojem Scavenger a tvorbu kvalitnějších pravidel programem YaraGen.

V budoucnosti je možné využít rozbor různých zkoumaných možností rozšíření nástroje FileInfo, které nebyly implementované v rámci této práce, jako inspiraci pro další vylepšování tohoto nástroje. Mezi takové možnosti patří: analýza automatizačních skriptů, artefakty jazyka Go a výpočet hashe Rich hlavičky.

Literatura

[1] Adams, C., Cain, P., Pinkas, D. a Zuccherato, R. *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)* [Internet Requests for Comments]. RFC 3161. RFC Editor, Srpen 2001. http://www.rfc-editor.org/rfc/rfc3161.txt. Dostupné z: http://www.rfc-editor.org/rfc/rfc3161.txt.

[2] Center, M. S. R. *Microsoft releases Security Advisory 2718704*. Microsoft, Červen 2012 [cit. 2021-03-17]. Dostupné z: https://msrc-blog.microsoft.com/2012/06/03/microsoft-releases-security-advisory-2718704/.

[3] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R. et al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile* [Internet Requests for Comments]. RFC 5280. RFC Editor, Květen 2008. Dostupné z: http://www.rfc-editor.org/rfc/rfc5280.txt.

[4] Eilam, E. *Reversing : Secrets of reverse engineering*. Indianapolis, IN: Wiley, 2005. ISBN 978-0764574818.

[5] Grunzweig, J. *The Gopher in the Room: Analysis of GoLang Malware in the Wild*. Červenec 2019 [cit. 2021-03-13]. Dostupné z: https://unit42.paloaltonetworks.com/the-gopher-in-the-room-analysis-of-golang-malware-in-the-wild/.

[6] Hiroyuki, K. a Fujisawa, K. *Potential Targeted Attack Uses AutoHotkey and Excel*. Trend Micro, Duben 2019 [cit. 2021-03-17]. Dostupné z: https://www.trendmicro.com/en_us/research/19/d/potential-targeted-attack-uses-autohotkey-and-malicious-script-embedded-in-excel-file-to-avoid-detection.html.

[7] International Telecommunication Union. *Information Technology — Abstract Syntax Notation One (ASN.1): Specification of Basic Notation* [ITU-T Recommendation X.680]. Červenec 2002.

[8] International Telecommunication Union. *Information Technology — ASN.1 Encoding Rules — Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER)* [ITU-T Recommendation X.690]. Červenec 2002.

[9] Intezer. *Year of the Gopher: 2020 Go Malware Round-Up*. Únor 2021 [cit. 2021-03-14]. Dostupné z: https://www.intezer.com/wp-content/uploads/2021/02/Intezer-2020-Go-Malware-Round-Up.pdf.

[10] Kaliski, B. *PKCS #7: Cryptographic Message Syntax Version 1.5* [RFC 2315]. RFC Editor, Březen 1998. DOI: 10.17487/RFC2315. Dostupné z: https://rfc-editor.org/rfc/rfc2315.txt.

[11] Kim, D., Kwon, B. J. a Dumitraş, T. Certified Malware: Measuring Breaches of Trust in the Windows Code-Signing PKI. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2017, s. 1435–1448. CCS '17. DOI: 10.1145/3133956.3133958. ISBN 9781450349468. Dostupné z: https://doi.org/10.1145/3133956.3133958.

[12] Kozák, K., Kwon, B. J., Kim, D., Gates, C. a Dumitras, T. Issued for Abuse: Measuring the Underground Trade in Code Signing Certificate. *CoRR*. 2018, abs/1803.02931. Dostupné z: http://arxiv.org/abs/1803.02931.

[13] Křoustek, J., Matula, P. a Zemek, P. *RetDec: An Open-Source Machine-Code Decompiler*. 2017. Dostupné z: https://retdec.com/static/publications/retdec-slides-botconf-2017.pdf.

[14] Levine, J. *Linkers and loaders*. San Francisco: Morgan Kaufmann, 2000. ISBN 978-1-55860-496-4.

[15] Maayan, G. D. *The IoT Rundown For 2020: Stats, Risks, and Solutions*. Leden 2020. Dostupné z: https://securitytoday.com/articles/2020/01/13/the-iot-rundown-for-2020.aspx.

[16] Madiant. *Tracking Malware with Import Hashing*. Leden 2014. Dostupné z: https://www.fireeye.com/blog/threat-research/2014/01/tracking-malware-import-hashing.html.

[17] Merces, F. *VirusTotal Now Supports Trend Micro ELF Hash*. Trend Micro, Říjen 2020. Dostupné z: http://www.trendmicro.com/en_us/research/20/j/virustotal-now-supports-trend-micro-elf-hash.html.

[18] Mercês, F. *Telfhash: An Algorithm That Finds Similar Malicious ELF Files Used in Linux IoT Malware*. Trend Micro, Duben 2019 [cit. 2021-03-18]. Dostupné z: https://documents.trendmicro.com/assets/pdf/TB_Telfhash-%20An%20Algorithm%20That%20Finds%20Similar%20Malicious%20ELF%20Files%20Used%20in%20Linux%20IoT%20Malware.pdf.

[19] Microsoft Corporation. *Windows Authenticode Portable Executable Signature Format*. 2008. Dostupné z: https://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/Authenticode_PE.docx.

[20] Microsoft Corporation. *PE Format specification*. 2020. Dostupné z: https://docs.microsoft.com/windows/win32/debug/pe-format.

[21] Milkovič, M. *Systém pro detekci vzorů v binárních souborech*. Brno, CZ, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/20063/.

[22] Nystrom, M. a Kaliski, B. *PKCS #9: Selected Object Classes and Attribute Types Version 2.0* [Internet Requests for Comments]. RFC 2985. RFC Editor, Listopad 2000.

[23] Oliver, J., Cheng, C. a Chen, Y. TLSH – A Locality Sensitive Hash. In: *2013 Fourth Cybercrime and Trustworthy Computing Workshop*. 2013, s. 7–13. DOI: 10.1109/CTC.2013.9.

[24] Poslušný, M. a Kálnai, P. *Rich Headers: leveraging this mysterious artifact of the PE format for threat hunting*. ESET, Listopad 2019 [cit. 2021-04-23]. Dostupné z: https://www.avar2019.org/files/AVAR2019_Rich_Headers_Kalnai_Poslusny_ESET_.pdf.

[25] Raiu, C. *Stuxnet and stolen certificates*. Kaspersky, Červenec 2010. Dostupné z: https://securelist.com/stuxnet-and-stolen-certificates/29724/.

[26] Schwartz, M. J. *Police Trick Malware Gang Into Disinfecting 850,000 Systems*. Information Security Media Group, Corp., Srpen 2019 [cit. 2021-03-17]. Dostupné z: https://www.databreachtoday.com/police-trick-malware-gang-into-disinfecting-850000-systems-a-12989.

[27] Sikorski, M. *Practical Malware Analysis : a Hands-On Guide to Dissecting Malicious Software*. San Francisco: No Starch Press, 2012. ISBN 1593272901.

[28] Team, F.-L. T. R. *Understanding Code Signing Abuse in Malware Campaigns*. Trend Micro, Duben 2018 [cit. 2021-03-17]. Dostupné z: https://www.trendmicro.com/en_us/research/18/d/understanding-code-signing-abuse-in-malware-campaigns.html.

[29] Tool Interface Standards. Executable and Linkable Format (ELF). *Specification, Unix System Laboratories*. 1.2. 1995. Dostupné z: https://refspecs.linuxfoundation.org/elf/elf.pdf.