Posudek oponenta disertační práce

Ing. Tomáš Zahradnický, Ph.D.

Praha 18. listopadu 2014

Předmětem posudku je disertační práce Ing. Lukáše Ďurfiny na téma „Generický zpětný překlad za účelem rozpoznání chování“, v originále „Generic Reverse Compilation to Recognize Specific Behavior“, předložená na Fakultě informačních technologií Vysokého učení technického v Brně. Tento posudek byl vypracován na základě požadavku o vystavení oponentského posudku č.j. 229/1493/2014. Práce sestává z 94 stran textu členěných do 8 kapitol, seznamu použité literatury a jedné přílohy. Posudek je členen na stručný popis práce, komentáře a dotazy k jednotlivým kapitolám disertace, a dále hodnotící sekce.

1 Stručný popis práce

Kapitola první, Introduction, krátce uvádí do čtenáře do problematiky reverzního překladu. Následující kapitola, Definitions, popisuje základní klíčové termíny používané dále v práci. Třetí kapitola, Malware, se věnuje převážně obfuskaci binárního a interpretovaného kódu znepřehledňující zpětnou analýzu. Čtvrtá kapitola, Reverse Engineering, popisuje dostupné nástroje pro reverzní inženýrství a srovnává je. Pátá kapitola, Lissom Decompiler, stručně popisuje jazyk pro popis procesorů, sadu nástrojů LLVM, návrhovou fázi retargetovatelného reverzního překladače, předzpracování vstupních souborů, a dále front-end, middle-end a back-end překladače. Následuje aplikace reverzního překladače na tři kusy škodlivého softwaru Psyb0t, Aidra a Darlloz. Kapitola šestá, Detection of Specific Behavior, se zabývá nástroji pro porovnávání podobností zdrojových kódů Moss a JPlag a srovnává je s nástrojem LfDComparator, který využívá vlastní jazyk pro provádění porovnání LfD. Kapitola sedmá, Results, uvádí výsledky získané reverzním překladačem překládajícím do programovacího jazyka C, a dále výsledky detekce podobností nástroji Moss, JPlag a LfDComparator. Kapitola osmá, Conclusion, stručně shrnuje práci.

2 Kapitola druhá

Kapitola druhá, Definitions, popisuje základní klíčová slova jako jsou abeceda, jazyk, bezkontextová gramatika, základní blok, atd. Ty jsou popisovány obvyklým formalismem. Takováto úroveň formalismu se v práci již dále nevyužívá, což je na škodu, zejména v popisovaných algoritmech.

3 Kapitola třetí

Kapitola třetí, Malware, popisuje tzv. Internet of Things jako oblast silně postiženou škodlivým softwarem. Dále se však věnuje obfuskaci kódu, a to na binární i zdrojové úrovni — popisuje jednotlivé druhy obfuskace včetně příkladů na úrovni instrukcí v assembleru i zdrojovém kódu v C. V této kapitole trochu postrádám diskuzi o nástrojích na obfuskaci, packerech a enkodérech. Kromě packerů, o kterých se student zmiňuje, v práci jako například UPX, uvádím nástroj msfvenom, součást frameworku Metasploit [1], anebo Themida [2], který je svými vlastnostmi zcela unikátní.

4 Kapitola čtvrtá

Kapitla čtvrtá, Reverse Engineering, se věnuje reverznímu inženýrství (RI) počítačového softwaru. Reverzní inženýrství je dnes velmi aktuálním oborem, který poskytuje zpětnou vazbu návrhářům té či oné věci podléhající RI. RI se vyskytuje nejen na softwarové, ale i na hardwarové úrovni (dekapitace čipů, zjišťování schemat elektronických obvodů, útoky postranními kanály, atd.). To však není předmětem této práce. Student uvádí, že se RI používá k cituji: Cryptanalysis is a discipline, where reverse engineering is used to reveal a weakness of security ciphers or algorithms (str. 11, 3. věta pod nadpisem sekce 4.1). RI v kryptoanalýze je jen jedním z prostředků, které kryptoanalýza používá a nezkoumá šifru jako takovou, ale její implementaci, ze které případně (zpětně) rozkrývá šifrovací algoritmus. Přestože se slabina může objevit na kterékoliv úrovni, relevantní ke studentově práci a reverznímu inženýrství je slabina na úrovni implementace a z ní zpětně odvozeného algoritmu. Proto považuji toto tvrzení za nepřesné.

Dále se v kapitole čtvrté student věnuje historii zpětného překladu. Tuto část můžeme považovat za state-of-the-art v oboru. Zpětné překladače dělí na ty překládající strojový kód, objektový kód, jazyk symbolických instrukcí a překladače jazyka virtuálních strojů (angl. virtual machines). Student sice mluví o zpětných překladačích pro jazyky založené na frameworku .NET a na javě, nezmiňuje se však o Dalvik VM, kterou používají mobilní telefony s operačním systémem Android (85 % trhu) po celém světě. Dále student uvádí volné i komerční zpětné překladače, mezi nimi i Hex-Rays Decompiler. Uvádí, že produkt nepodporuje architekturu x86_64. Současná verze produktu Hex-Rays Decompiler, ke dni posudku verze 6.6, již architekturu x86_64 podporuje. Vzhledem k tomu, že student svoji práci odevzdal před uvedením verze 6.6 na trh, nemohl tuto skutečnost vědět.

5 Kapitola pátá

Kapitola pátá, Lissom Decompiler, popisuje zpětný překladač tohoto jména. Student je jedním ze členů vývojového týmu a věnuje se výzkumu a vývoji front-endu tohoto překladače. Po popisu jazyku ISAC, sloužícím pro popis procesoru a jím vykonávaných instrukcí, popisuje základy LLVM a jeho interní reprezentace (IR). Dále následuje začlenění vyvíjeného zpětného překladače do architektury LLVM a jeho rozdělení na front-end, middle-end a back-end, a dále stručný podpis platformově nezávislého formátu pro

objektové souboru CCOFF, do něhož jsou v rámci předzpracování převedeny vstupní soubory. Vyvíjený zpětný překladač pracuje na vstupu se soubory předzpracovanými do formátu CCOFF. Dále se pátá kapitola zabývá popisem front-endu, middle-endu, back-endu a zkušenostmi se zpětným překladem tří exemplářů škodlivého softwaru.

5.1 Front-end

Dále v páté kapitole následuje popis front-endu překladače, který překládá vstup ve formě CCOFF formátu do interní reprezentace LLVM IR. Součástí tohoto procesu je detekce staticky linkovaného kódu. Zde student používá obdobné principy jako technologie Fast Library Identification and Recognition Technology (F.L.I.R.T) [3] používaná produktem Hex-Rays Interactive Disassembler (IDA) — tj. tvorba souboru se vzory obsahů funkcí, jejich slučování do souborů signatur a řešení případných konfliktů. Vzhledem k tomu, že student musel technologii F.L.I.R.T znát, protože dělal v rámci 4. kapitoly průzkum dostupných zpětných překladačů, nevím, proč se o této technologii nezmiňuje a necituje ji. Student by měl u obhajoby vysvětlit, jaká je míra podobnosti mezi jím navrhovaným postupem a technologií F.L.I.R.T. Student dále popisuje Application Binary Interface (ABI), podle mého názoru poněkud nepřesně (str. 37, odstavec pod nadpisem sekce Application Binary Interface).

Kapitola pátá se také věnuje získávání podpůrné informace z ladících a symbolických informací, dekódování instrukcí a importními a exportními tabulkami, analýze importních a exportních tabulek a analýze systémových volání. U architektury x86 uvádí, že volání jádra operačního systému (syscall) se provádí pomocí instrukce `int` s operandem `0x80`. Toto tvrzení považuji za správné pro platformu Linux, na jiných operačních systémech na platformě x86 se pro volání jádra používají i další instrukce jako např. `sysenter` anebo `syscall`. Na platformě Windows lze použít sice `int 0x2e` tak, jako na platformě Linux `int 0x80`, ale na novějších procesorech se zásadně používají instrukce `sysenter` a `syscall`. Jakým způsobem probíhá analýza volání jádra na platformě x86 pro operační systém Microsoft Windows?

Následuje popis detekce funkce `main()`. Student správně uvádí, že vstupní bod (main entry point) je odlišný od funkce `main` a popisuje metody, jakými lze adresu funkce `main` získat. Tato funkce však nemusí být první „uživatelskou“ funkcí, kterou runtime zavolá. Před voláním `main` jsou mj. volány konstruktory staticky alokovaných objektů a v případě GCC také funkce označené atributem `__attribute__((constructor))`, anebo `__declspec(allocate)` u překladače z Microsoft Platform SDK. Obdobně tomu je také po opuštění funkce `main`. O tomto však v práci diskuzi nenacházím. Jak se bude analyzátor chovat v případě, že veškerý škodlivý kód bude proveden ještě před vstupem, anebo po opuštění funkce `main`?

Dále následují popis tzv. delay slotu a způsob převodu zpožděných instrukcí do LLVM IR a popis analýzy kódu pro x86 FPU architekturu. FPU architektura sdílí registry `st0-st7` spolu s vektorovým rozšířením MMX registry `mm0-mm7`, v textu se o tomto nic nepíše, a proto se ptám: Jak je řešena směs FPU a MMX instrukcí?

Následují analýza toku a skokových instrukcí spolu s detekcí funkcí. Detekce funkcí kombinuje přístup shora-dolů s přístupem sdola-nahoru. Kapitola pokračuje analýzou

datových toků a analýza argumentů funkcí. Zde se uvádí cituji: „For the Intel x86 architecture, a detection by a prologue and epilogue is normally used.“ Znamená to, že pro jiné platformy nejsou prology a epilogy funkcí rozpoznávány? Sekce pokračuje analýzou zásobníku, která se zaměřuje na přístupy instrukcí do paměti. V této sekci a v sekci Local Varibles Detection na str. 52 by se mělo mluvit o rámci zásobníku. Lokální proměnné jsou přece alokovány na zásobníku a přistupuje se k nim normálně, pokud není generování rámců vypnuto při překladu, prostřednictvím vyhrazeného registru (IA-32 platforma používá registr `ebp`). O rámcích však žádnou zmínku nenacházím. Proč? Následuje analýza globálních proměnných. Student uvádí, že se ukládají obvykle do `.data` sekce. Jak se bude analyzátor chovat ke globálním proměnným uloženým v kódové `.text` (řetězce) anebo v `.bss` sekci (globální proměnné inicializované na nulu)? V závěru front-endu se mluví o analýze mrtvého kódu (dead-code analysis) a analýze typů. Pokud je zdrojový kód v C++ anebo Obj-C, RTTI bude obsahovat množství informace. Provádí se analýza informací obsažených v RTTI? Sekce je završena krátkým popisem generátoru vnitřní formy LLVM IR.

5.2 Middle-end

Účelem middle-endu je (de)optimalizuje vnitřní formy LLVM IR. Student uvádí, že tato část je v jeho práci uvedena kompletnost. Sekce popisuje různé optimalizační techniky jako např. analýzu ukazatelů (alias analysis), eliminaci mrtvého kódu, propagaci konstant, atd.

5.3 Back-end

V této sekci se popisuje převod optimalizované LLVM IR do další vnitřní formy Back-end IR (BIR). BIR pak slouží jako vstupní forma pro generátory kódu vyšších programovacích jazuků, konkrétně v C a v jazyce LfD, který student zavádí. Jako součást této sekce je rekonstrukce příkazů vyšších programovacích jazyků (if/then/else, smyčky, atd.), určování znaménkovosti datových typů. Student zde píše (str. 59, sekce 5.7.2, odst. 4) že LLVM IR nerozlišuje mezi integrálními typy se znaménkem a bez něj. Informaci o znaménkovosti lze vyčíst z použitých instrukcí např. instrukcí podmíněného skoku. Jakým způsobem front-end propaguje z CCOFF souboru informace o znaménkovosti typu do LLVM IR? Dále se v sekci hovoří o používání proměnných při volání funkcí a optimalizacích back-endu jako např. zjednodušování aritmetických výrazů, propagace kopií proměnných, převody mezi globálními a lokálními proměnnými, přejmenovávání proměnných a eliminaci nadbytečných závorek. Tím popis back-endu končí.

5.4 Malware Decompilation Experience

Tato sekce popisuje zkušenosti, které byly dosaženy aplikaci zpětného překladače na škodlivý software Psyb0t, Aidra a Dallock. Celá sekce 5.8.1 bez úvodního odstavce je doslovnou kopií odkazu č. 69, totéž u sekce 5.8.2, která je kopií odkazu č. 67. Tvrzení na staně 67, že cituji: „...of the conditional branch instruction `bal`“, je instrukce

podmíněného skoku považuji za chybné. Instrukce `bal` je zkratkou „branch absolute and link“, nejde tedy o podmíněný, ale o nepodmíněný skok.

6 Detection of Specific Behavior

Tato kapitola popisuje metodu srovnávání podobností zdrojového kódu, která se aplikuje na zdrojové kódy ve vyšším programovacím jazyce pořízené prostřednictvím zpětného překladu. Student zde nejdříve popisuje dva analyzátory zdrojových kódů v C, a to Moss a JPlag. Na straně 81 je použit téměř identický výňatek z citovaného dokumentu č. 50 ([4]), a to:

> *JPlag's algorithm computes similarity in two phases:*
>
> 1. *All programs are parsed and converted to token strings.*
> 2. *These tokens are compared in pairs for determining the similarity of each pair. The used method is Greedy String Tiling. During each comparison, JPlag attempts to cover one token (string) with substrings (tiles) taken from the other as well as possible. The similarity value is given by the percentage of token strings that can be covered.*

Výňatky obdobného charakteru bývá zvykem viditelně odlišit od ostatního textu tak, jako v tomto posudku. Dále následuje popis studentova vlastního jazyka LfD, jehož gramatika tvoří přílohu A. Soubory v tomto jazyce jsou zpracovávány následně studentovým nástrojem LfDComparator, který vyhodnocuje míru podobnosti. Výsledky jsou pak srovnávány s nástroji Moss a JPlag. Student musel evidentně navrhnout a implementovat generátor pro jazyk LfD z vnitřní reprezentace back-endu (BIR) (viz obrázek 5.16 na str. 57). V práci tuto informaci a jeho popis nenacházím, proč? Očekával bych, že algoritmy, které jsou používány v nástroji LfDComparator budou popsány více formálně a bude řečeno, jakým způsobem se podobnost vyhodnocuje. Student uvádí, že podobnost se měří v procentech, ale jakým způsobem se tato procenta počítají?

7 Results

Tato sekce popisuje výsledky, kterých student dosáhl. První druh je testování vlastního zpětného překladače a srovnání kódu generovaného zpětným překladem s originály zdrojového kódu v programovacím jazyce C. Druhý druh výsledků je z testování shod pomocí nástrojů Moss, JPlag a LfDComparator. Jako testovací množinu student používá celkem 10 zdrojových kódů v C, které však blíže nespecifikuje. Z těchto zdrojových kódů vytvoří kombinací platformy, úrovně optimalizace, souborového formátu a ladících informací celkem 30 testovacích souborů na 1 zdrojový kód v C. Tyto soubory pak zpracovává zpětným překladačem a srovnává je s originály prostřednictvím jmenovaných tří nástrojů. U každého nástroje uvádí počty porovnání a míru shody v procentech. Student by měl u obhajoby upřesnit, jaký byl charakter těchto dat a způsob porovnávání v LfD. Dále student porovnává verze škodlivého softwaru Aidra a Darlloz pro různé platformy,

které prezentuje v tabulce 7.1 na straně 92. Jsou-li výsledky skutečně takové, pak to znamená, že způsob srovnávání je velice účinný. Ze známých vzorků na jedné platformě budeme schopni účinně rozpoznat neznámý škodlivý software na platformě jiné.

8 Conclusion

Tato sekce shrnuje obsah práce a dosažené výsledky.

9 Obsah CD

Součástí práce je CD, na kterém nacházím zdrojové kódy studentovy disertace ve formátu LaTeX. Kromě nich již na CD nic dalšího nenacházím. Očekával jsem, že naleznu popisy procesoru, nástroj LfDComparator, jeho zdrojové kódy, generátor LfD z BIR, a další studentem v práci popsaný a vytvořený software.

10 Jazyková a typografiká úroveň

Jazykovou a typografickou úroveň práce považuji za lehce podprůměrnou. Práce obsahuje velké množství chybějících členů, občas i nějaký překlep. Typografická úroveň práce by mohla být také lepší. V tištěné verzi jsou zdrojové kódy špatně čitelné a často se stává, že obrázek je i o stránku, dvě od místa, kde by měl být.

11 Vědecká aktuálnost tématu

Pojem obecného zpětného překladu a jeho použití pro detekci (neznámého) škodlivého softwaru a problematika retargetovatelného zpětného překladače jsou aktuálním vědeckým tématem. Reverzní analýza poskytuje cennou zpětnou vazbu v návrhu (nejen) počítačového hardwaru a softwaru a je v dnešní době silně aktuálním tématem. Téma proto považuji za silně aktuální.

12 Přínos práce

Práci považuji za přínosnou. Skutečnost, že ze vzorků škodlivého softwaru pro jednu platformu jsme schopni detekovat podobný škodlivý software na jiné platformě, pro který dosud není k dispozici vzorek, je velmi povzbudivá. Jako přínosy spatřuji:

1. Návrh metod a implementace front-endu pro framework LLVM transformující binární kód do platformově nezávislé vnitřní formy.

2. Návrh jazyka LfD a metody pro srovnávání zdrojových kódů a její implementace v nástroji LfDComparator umožňující srovnávání známých vzorků škodlivého softwaru pro jednu platformu na platformě jiné.

13 Publikační činnost

Student ve svém životopise uvádí celkem 2 časopisecké publikace, 10 konferenčních příspěvků, 3 posterové prezentace a 1 přísvěvek v soutěži. Časopisecké publikace považuji za kvalitní. Časopis Kybernetika je časopisem s impakt faktorem. Článek se zabývá obfuskací zdrojového kódu. Deobfuskace je předmětem studentovy disertační práce, proto považuji tento článek za přímo souvislý s tématem práce. Článek do časopisu The International Journal of Security and Its Applications má více než 3 autory. Jaký je studentův podíl na tomto článku? Mezi konferenčními publikacemi nacházím několik příspěvků na konferenci WCRE, kterou považuji za prestižní konferenci v oboru reverzního inženýrství. Jádro práce považuji za dostatečně publikované. Publikační činnost za 4 roky studentova studia považuji za nadprůměrnou.

14 Závěr

Práci Ing. Lukáše Ďurfiny považuji za přínosnou. Student navrhl, implementoval o ověřil metody ve front-endu reverzního překladače, a dále porovnávač kódu. Výsledky práce jsou přínosné a za ně doporučuji práci k obhajobě. Student prokázal svoji samostatnost při návrhu metod a samostatné vědecké činnosti. Co se úrovně práce týče, měla by před publikací na síti Internet projít jazykovou korekturou spojenou s opravou drobných nepřesností a, pokud možno, formalizací navržených a použitých algoritmů.

Ing. Tomáš Zahradnický, Ph.D.
FIT ČVUT v Praze

Reference

[1] Rapid7, Inc.: Metasploit. `https://www.metasploit.com/`. 2014.

[2] Oreans Technologies: Themida — Advanced Windows Protection System. `http://www.oreans.com/themida.php`. 2014.

[3] Hex-Rays SA: IDA F.L.I.R.T. Technology: In-Depth. Dostupné online na adrese: `https://www.hex-rays.com/products/ida/tech/flirt/in_depth.shtml`. 2014.

[4] Prechelt L., Malpohl G. a Phillipsen M.: Finding plagiarism among a set of programs with JPlag, Journal of Computer and Science, vol. 8, no. 11, `http://www.jucs.org/jucs_8_11/finding_plagiarisms_among_a/Prechelt_L.pdf`. 2002.