OPTIMIZATIONS IN A RETARGETABLE DECOMPILER

Jaroslav Kollár
Bachelor Degree Programme (3), FIT BUT
E-mail: xkolla03@stud.fit.vutbr.cz

Supervised by: Petr Zemek
E-mail: izemek@fit.vutbr.cz

Abstract: An important part of information technology today is security. The most vulnerable elements at present are mobile phones, tablets and other similar devices. To create a malware protection may be used decompilation. Acquired knowledge of decompilation is used in the creation of anti-virus programs. Decompilation as a technique falls under the area of reverse engineering. Readability of the code produced by the decompiler is really important. Without this feature, the analysis of malware would be very difficult. The readability of the produced code is ensured by plenty of optimizations. This article is about a decompiler which is developed within the project Lissom and about optimizations in this decompiler.

Keywords: reverse engineering, retargetable decompiler, optimizations, readability of source code

1 ÚVOD

Dôležitým pojmom v rámci informačných technológií je v dnešnej dobe bezpečnosť. Jedným z najčastejších ohrození či už počítačov, mobilov, tabletov a iných zariadení, patrí ohrozenie rôznymi druhmi škodlivého softvéru, tzv. malvérom [1].

Medzi osvedčenú techniku obrany patrí obrana pomocou antivírusových programov. Tieto programy prehľadávajú súbory, sledujú správanie sa aplikácií a aktivitu systému na pozadí. Pri týchto činnostiach súčasne porovnávajú či napríklad chovanie niektorej aplikácie neodpovedá správaniu sa napadnutého programu. V súboroch zase hľadajú sekvenciu inštrukcií, ktorá odpovedá nejakému malvéru [2]. Jednou z techník, ktorú môžu použiť tvorcovia aplikácii proti malvéru, je softvérové reverzné inžinierstvo. Pod pojmom reverzné inžinierstvo chápeme proces, ktorého úlohou je získať chýbajúce znalosti o skúmanom predmete. Ako príklad je možné uviesť využitie spätného prekladu nad získaným binárnym súborom. O spätný preklad sa postará spätný prekladač, pričom jeho úlohou je z tohto súboru vygenerovať program vo vysokoúrovňovom jazyku [3]. Následne je možné tento kód preskúmať a vyhodnotiť jeho činnosť. Za týmto účelom je vyvíjaný spätný prekladač v rámci projektu Lissom. Tento spätný prekladač však môže byť aj použitý na transformáciu kódu z jedného programovacieho jazyka do iného [5].

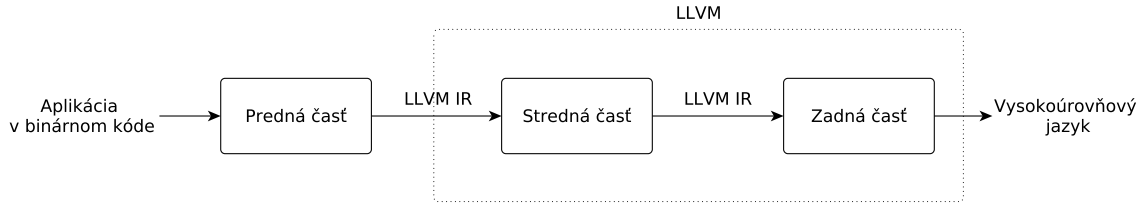
Témou tohto článku je popis tohto vyvíjaného spätného prekladača. Pozornosť bude hlavne venovaná optimalizáciám v zadnej časti spätného prekladača. Cieľom týchto optimalizácií je poskytnúť výsledný výstupný kód v čo najčitateľnejšej podobe. Takýto kód je potom jednoduchšie analyzovať.

2 SPÄTNÝ PREKLADAČ VYVÍJANÝ V RÁMCI PROJEKTU LISSOM

Táto sekcia sa venuje popisu spätného prekladača vyvíjaného v rámci projektu Lissom. Uvedený obrázok **1** demonštruje jeho štruktúru. Každá podsekcia sa venuje práve jednej jeho časti.

Hlavným cieľom je vytvoriť spätný prekladač nezávislý na architektúre (MIPS, ARM, Intel x86, atď.) a súborových formátoch (ELF, PE, Mach-O, atď.), generujúci zvolený vysokoúrovňový jazyk.

Jeho využitie je plánované pri analýze malvéru napríklad pre mobilné zariadenia, tablety a iné podobné zariadenia. Momentálne podporované jazyky sú jazyk C a modifikovaná verzia jazyku Python.



Obrázek 1: Štruktúra spätného prekladača vyvíjaného v rámci projektu Lissom.

2.1 PREDNÁ ČASŤ

Hlavnou úlohou prednej časti je prevod binárneho kódu platformovo závislej aplikácie do sekvencie LLVM IR[1] inštrukcií. O tento účel sa stará inštrukčný dekodér. Jeho funkcionalita je podobná disassembleru, avšak výstupom nie je kód v jazyku assembler, ale sekvencia inštrukcií odpovedajúca reprezentácii LLVM IR. Predná časť spätného prekladača je jediná časť, ktorá je platformovo závislá, pretože sa automaticky generuje v závislosti na cieľovú architektúru [4].

2.2 STREDNÁ ČASŤ

Vstupom strednej časti je sekvencia inštrukcií v jazyku LLVM IR vygenerovaná prostredníctvom prednej časti. Táto sekvencia inštrukcií je však veľmi nízkoúrovňová. Vo väčšine prípadov obsahuje veľké množstvo redundantných inštrukcií. Každý základný blok predstavuje jednoduchú inštrukciu jazyka assembler. Hlavnou úlohou strednej časti je optimalizácia LLVM IR (napr. pomocou optimalizácií z LLVM[2]) z dôvodu zlepšenia produkovaného výsledku zadnou častou [4].

2.3 ZADNÁ ČASŤ

Poslednou časťou spätného prekladača je zadná časť. Jej výstupom je kód vysokoúrovňového jazyka. Tento produkovaný kód však často neobsahuje formát takého kódu, aký by programátor vytvoril. Z tohto dôvodu sú v poslednej časti navrhnuté optimalizácie, ktorých hlavnou úlohou je výsledný kód pretransformovať do čitateľnejšej podoby [4].

3 OPTIMALIZÁCIE V ZADNEJ ČASTI SPÄTNÉHO PREKLADAČA

K tomu, aby bol výsledný kód jednoduchšie analyzovateľný, je potrebné nad produkovaným kódom vykonať optimalizácie. Vytvorenie nových optimalizácií k už vytvoreným je predmetom mojej bakalárskej práce, ku ktorej je tematikou tento článok tvorený. Hlavný problém je ten, že výsledný kód produkovaný spätným prekladačom len v zriedkavých prípadoch bude odpovedať kódu napísaného programátorom. Je to z toho dôvodu, že pri preklade dochádza k strate informácií ako sú názvy premenných, komentáre, deklarácie štruktúr a iné [3]. Pri optimalizáciach pri preklade takisto dochádza k nahradeniu niektorých druhov inštrukcií za iné, napríklad z dôvodu urýchlenia programu [8]. Ako je vidieť, tak problematika spätného prekladu je naozaj náročná. Ako príklad niektorých optimalizácií, ktorými sa teraz zaoberám, môžem uviesť odstránenie redundantných zátvoriek z výrazov. Cieľom

[1] LLVM IR predstavuje základnú prechodnú reprezentáciu, ktorá poskytuje typovú bezpečnosť, nízkoúrovňové operácie, flexibilitu a schopnosť prevodu do takmer všetkých vysokoúrovňových jazykov [6].

[2] LLVM systém bol pôvodne navrhnutý ako inovatívny framework pre prekladače. Obsahuje množinu jazykovo nezávislých inštrukcií, veľké množstvo vstavaných optimalizačných algoritmov a prechodnú reprezentáciu LLVM IR [7].

tejto optimalizácie je odstrániť redundantné zátvorky za pomoci matematických pravidiel ako je asociativita a komutativita operátorov. Pôvodný výstupný kód pred touto optimalizáciou obsahoval zátvorky ohraničujúce každú jednu operáciu vykonávanú nad dvoma operandami. Ďalšia optimalizácia je zameraná na zjednodušenie aritmetických výrazov. Pôvodný kód často obsahoval výrazy, ktoré je možné zjednodušiť. Napríklad násobenie/delenie číslom jedna, kombinácia znamienok plus a mínus, ktorá je nahraditeľná len za mínus, ďalej odčítanie dvoch rovnakých čísel, ktoré je možno nahradiť za číslo nula a iné. Pri preklade dochádza často k náhrade operácie delenia a násobenia za bitové posuvy doľava a doprava [8]. Ich spiatočná náhrada za násobenie a delenie opäť zlepšuje čitateľnosť kódu. Kód často obsahoval prípady, že deklarácia premennej a následné priradenie hodnoty do tejto premennej bola na dvoch rôznych miestach kódu. Toto bolo tiež cieľom ďalšej optimalizácie, ktorej úlohou je zlúčiť tieto dve činnosti rovno do deklarácie premennej.

4 ZÁVER

Ako je vidieť, tak prínos spätného prekladu pre bezpečnosť v rámci informačných technológií je evidentný. Vytvorenie spätného prekladača je možné považovať za náročnú úlohu. Kvalita produkovaného kódu závisí vo veľkej miere na optimalizáciách. Uvedené optimalizácie v tejto práci predstavujú len krátky úvod do tejto problematiky. Optimalizácie v zadnej časti spätného prekladača sú však závislé na vyspelosti navrhnutého spätného prekladača. Zavádzaním nových jazykových konštrukcií, ktoré dokáže spätný prekladač vytvárať, dochádza k objavovaniu nových možných optimalizácií.

POĎAKOVANIE

Tento príspevok vznikol za podpory grantu TAČR TA01010667, FIT-S-11-2 a výskumného zámeru MSM0021630528.

REFERENCE

[1] TechTerms.com: Malware [online]. `http://www.techterms.com/definition/malware`, [cit. 2013-2-24]

[2] Gerçek, B.: Jak funguje antivirový program? [online]. `http://www.symantec.com/region/cz/resources/antivirus.html`, [cit.2013-02-27]

[3] Eilam, E.: Reversing: Secrets of Reverse Engineering. Wiley Publishing, Inc., 2005, ISBN-10: 0-7645-7481-7

[4] Ďurfina, L.; Křoustek, J.; Zemek, P.; aj.: Design of a Retargetable Decompiler for a Static Platform-Independent Malware Analysis. International Journal of Security and Its Applications, 2011, s. 91–106, ISSN 1738-9976

[5] Ďurfina, L.; Křoustek, J.; Zemek, P.: Generic Source Code Migration Using Decompilation. In 10th Annual Industrial Simulation Conference (ISC'2012), EUROSIS, 2012, ISBN 978-90-77381-71-7, s. 38–42

[6] LLVM Project: LLVM Language Reference Manual [online]. `http://llvm.org/docs/LangRef.html`, Posledné úpravy 10.1.2013 [cit. 2013-02-24]

[7] Ďurfina, L.; Křoustek, J.; Zemek, P.; aj.: Design of an Automatically Generated Retargetable Decompiler. In 2nd European Conference of COMPUTER SCIENCE (ECCS'11), North Atlantic University Union, 2011, ISBN 978-1-61804-056-5, s. 199–204

[8] harvestsoft: Bit Shifting Technique [online]. `http://harvestsoft.net/bitshift.htm`, [cit. 2013-02-24]