GENERIC OBFUSCATION ON BYTECODE LEVEL

Samuel Kollát

Bachelor Degree Programme (3), FIT BUT

E-mail: xkolla04@stud.fit.vutbr.cz

Supervised by: Lukáš Ďurfina

E-mail: idurfina@fit.vutbr.cz

Abstract: This document is intended to provide description of baseline obfuscation methods and effectiveness of those methods. It also aims at application of generic obfuscation on bytecode level using LLVM compiler and its toolchains. The first chapter contains definition and main domains of obfuscation. It is followed by brief introduction to LLVM system and commentary on suitability of LLVM infrastucture for realization of generic obfuscation.

Keywords: obfuscation, bytecode, compiler, LLVM, reverse engineering

1. ÚVOD

Obfuskácia, alebo zahmlievanie, zatemnenie, je cielené skrytie významu v komunikácií, čím sa komunikácia stáva náročnejšia na interpretovanie a následné pochopenie [1]. Skrývanie významu sa využíva vo viacerých oblastiach a jednou z nich je obfuskácia počítačových aplikácií. Touto metódou je možné sťažiť, v najlepšom prípade zabrániť, reverzným inžinierom analyzovať spôsob a metódy, akými aplikácia realizuje svoju činnosť.

2. OBLASTI VYUŽITIA

Pred samotným popísaním oblastí využitia je vhodné klasifikovať obfuskáciu do dvoch základných tried na základe typu kódu, nad ktorým je realizovaná.

Prvá trieda zahrňuje obfuskáciu nad zdrojovým kódom. Tieto obfuskačné transformácie sú nevratné a realizujú jednoduché zmeny nad kódom s cieľom zhoršiť jeho čitateľnosť. Z hľadiska využitia je významnejšia druhá trieda obfuskačných metód.

V druhej triede sa nachádzajú metódy pracujúce nad strojovým kódom aplikácie. Ich cieľom je zhoršiť analýzu kódu pri reverznom inžinierstve a ich využitie bude opísané v nasledujúcom texte tejto kapitoly. Podmienkou obfuskácie je zmeniť štruktúru a obsah pôvodnej aplikácie bez toho, aby sa zmenilo správanie tejto aplikácie [2].

2.1. OCHRANA INTELEKTUÁLNEHO VLASTNÍCTVA

Obfuskácia sa stala jedným z nástroj tvorcov proprietárneho software. Pomocou obfuskácie sa snažia chrániť svoje intelektuálne vlastníctvo, teda algoritmy a dátové štruktúry, ktoré tvoria základ ich aplikácií. Metódy obfuskácie ponúkajú lacný a zároveň efektívny spôsob takejto ochrany.

2.2. ZAMEDZENIE DETEKCIE MALWARE

Rovnako sa dá obfuskácia použiť na zabránenie analýzy malware antivírovými programami. Toto vo veľkej miere využívajú autori škodlivého software. Množstvo antivírových programov využíva na detekciu malware vyhľadávanie pomocou signatúr v databáze vírusov [3]. Pri použití obfuskácie sa tieto metódy stávajú neúčinnými. Medzi najdokonalejší škodlivý software patrí metamorfný malware, ktorý obsahuje obfuskačný algoritmus priamo vo svojom tele. Pri každom jeho šírení sa vytvorí nová, obfuskovaná generácia pôvodného malware.

3. LLVM

Cieľom projektu je vytvoriť obfuskátor, ktorý umožní realizovať nielen jednotlivé obfuskačné transformácie nad pôvodným programom, ale bude taktiež nezávislý na cieľovej architektúre a umožní konfiguráciu obfuskácie.

Pre tento účel bol zvolený systém LLVM a s ním spojené nástroje. Obfuskácie sú realizované ako transformačné prechody nad vnútornou reprezentáciou strojového kódu, tzv. LLVM IR alebo bytecode. To umožňuje generickú obfuskáciu vzhľadom na cieľovú platformu, pretože táto obfuskovaná vnútorná reprezentácia sa následne preloží do strojového kódu cieľovej platformy.

Infraštruktúra LLVM bola zvolená pretože je optimalizovaná na vytváranie prechodov nad LLVM IR, umožňuje preklad do strojového kódu rôznych architektúr a pre samotné vlastnosti LLVM IR. Vnútorná reprezentácia kódu je silne typovaná, čo umožňuje realizovať transformácie, ktoré by nemohli byť realizované nad trojadresným kódom bez uskutočnenia ďalších analýz [4].

4. METÓDY OBFUSKÁCIE

Existuje množstvo obfuskačných transformácií, ktorých základnou odlišnosťou je schopnosť odolať deobfuskácií, teda odstráneniu obfuskácie s cieľom získať čo najpresnejšiu podobu pôvodného software. V projekte sú implementované nižšie zmienené metódy.

Jednou zo základných metód je vkladanie mŕtveho kódu. Ide o zmenu pôvodného kódu pomocou inštrukcií, ktoré nemajú žiadny význam na stav vykonávaného programu, ako sú napríklad inštrukcie nop. Taktiež sa môže jednať o sekvencie inštrukcií, ktoré najprv zmenia stav programu a následne ho vrátia späť. Táto metóda patrí medzi menej efektívne metódy obfuskácie. Jednou zo sekvencií v projekte je výraz využívajúci exkluzívnu disjunkciu, ktorý má v LLVM IR podobu (1).

```
%load = load i32* %n                    (1)
%xor = xor i32 %load, 0
store i32 %xor, i32* %n
```

Substitúcia je jednou z metód obfuskácie, ktorá vykazuje vhodnú efektivitu voči deobfuskácií [5]. Ide o nahradenie pôvodnej inštrukcie alebo množiny inštrukcií inými, významovo rovnakými. Efektivita obfuskácie záleží na množine, slovníku, podľa ktorého sa vykonávajú obfuskácie. Ako príklad môže slúžiť tabuľka 1.

Pôvodný výraz	Obfuskovaný výraz
x = 0	x = x - x
x == 0	(x || x) == 0
x = <fp> (ľubovoľné desatinné číslo)	x = rand(); x = (x-<fp>-x) / (x-1-x)

Tabuľka 1: Príklad slovníku pre substitúciu.

Základom substitúcie je aby jednotlivé dvojice v slovníku boli sémanticky ekvivalentné. Obfuskátor, ktorý je obsahom práce obsahuje aj jeden experimentálny prechod, ktorý má za cieľ nahradiť slovník iným spôsobom na získavanie obfuskovaných výrazov. Ide o algoritmus genetického programovania. Ten si na základe pôvodného výrazu vytvorí tabuľku výsledkov pre náhodné vstupy, ktorá bude slúžiť ako vstup pre jeho učenie a následne v niekoľkých generáciách má za úlohu vytvoriť matematický výraz, ktorý zodpovedá pôvodnému na základe vytvorenej tabuľky. Základným problémom ostáva, že nie je možné dokázať, že sa tento výraz bude chovať rovnako ako pôvodný pre všetky možné vstupy [6]. V projekte je tento problém riešený spôsobom, že za vhodný vygenerovaný výraz sa považuje iba taký výraz, ktorý dáva zhodné výsledky pre všetky testované vstupy, ako pôvodný výraz. Zároveň sa využíva dostatočné množstvo takýchto vstupov, aby sa minimalizovala možnosť zanesenia chyby výberom nevhodného výrazu.

Vytváranie umelých cyklov je metóda, ktorá vkladá iteráciu na miesta, kde pôvodne táto riadiaca konštrukcia neexistovala. Môže sa jednať o priradenie celého čísla do premennej. V obfuskovanom programe bude na miesto tohoto priradenia cyklus, v ktorom sa bude postupne hodnota tejto premennej meniť (napr. inkrementáciou, dekrementáciou), až kým nedosiahne cieľovú hodnotu.

Jednou z najznámejších metód obfuskácie je premiestnenie kódu. Ten je v projekte riešený dvoma samostatnými prechodmi. Prvý realizuje rozdelenie blokov kódu na množinu menších blokov a ich prepojenie pomocou nepodmienených skokov, pričom sa volí vhodná veľkosť blokov vzhľadom k pôvodnému a obfuskovanému programu. Druhý prechod realizuje premiestnenie jednotlivých blokov, ktoré sú prepojené nepodmienenými aj podmienenými skokmi. Následne premiestnené bloky prepojí, aby sa zachovala pôvodná sémantika programu. Táto metóda má za následok výraznú zmenu pôvodného kódu.

Klonovanie funkcií je zaujímavou technikou, implementovanou ako samostatný prechod. Ide o vytvorenie viacerých klonov funkcie, v prípade, že je volaná na viacerých miestach. Po následnej obfuskácií tiel jednotlivých funkcií sa jednotlivé klony javia ako funkcie s odlišným chovaním a efektom. Táto metóda pretvára priamu rekurziu na nepriamu, pričom výsledok ostáva rovnaký.

Poslednou spomenutou metódou budú nepriehľadné predikáty (opaque predicates). Nepriehľadný predikát je definovaný ako konštrukcia, ktorej hodnota je známa obfuskátoru, ale náročná na odvodenie pre deobfuskátor [5]. Príkladom je rovnica (2) používaná pre celočíselnú aritmetiku, ktorá sa vždy vyhodnotí ako hodnota *true*.

$$(x^2*(x+1)^2)\%4 == 0 \qquad (2)$$

5. ZÁVER

Takto realizovaný obfuskátor, ktorý je možné riadiť pomocou parametrov, nájde využitie v oblasti testovania deobfuskátorov a spätných prekladačov, ktorých kvalitná realizácia je nutná na odhaľovanie škodlivého software. Výhodou je tiež možnosť spustiť danú metódu viackrát, prípadne viacnásobne aplikovať určitú sériu obfuskácií. Týmto však môže narásť veľkosť výsledného programu za prijateľnú hranicu. Je nutné brať do úvahy, že obfuskovaný program môže podľa úrovne obfuskácie využívať viac operačnej pamäte ako neobfuskovaný program, alebo môže potrebovať viac času na svoj beh.

POĎAKOVANIE

Tento príspevok vznikol za podpory grantu TAČR TA01010667 a výskumného zámeru MSM0021630528.

REFERENCIE

[1] Ďurfina, L.; Kolář, D.: C Source Code Obfuscator, In: Kybernetika, roč. 48, č. 3, 2012, CZ, s. 8, ISSN 0023-5954

[2] Collberg, C.; Thomborson, C.; Low, D.: A Taxonomy of Obfuscating Transformations. Technical Report 148, University of Auckland, New Zealand, 1997.

[3] You, I.; Yim, K.: Malware Obfuscation Techniques: A Brief Survey. BWCCA, IEEE, 2010, s. 297-300.

[4] LLVM Language Reference Manual. http://llvm.org/docs/LangRef.html#introduction, 12-02-2013.

[5] Balakrishnan, A.; Schulze, C.: Code Obfuscation Literature Survey. http://pages.cs.wisc.edu/~arinib/writeup.pdf, 2005.

[6] Schwarz, J.; Sekanina, L.: VUT BRNO, FIT. Aplikované evoluční algoritmy: Studijní opora. Brno, 2006.