VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

AUTOMATIZOVANÁ STATICKÁ ANALÝZA NAD LLVM IR ZAMĚŘENÁ NA DETEKCI MALWARE

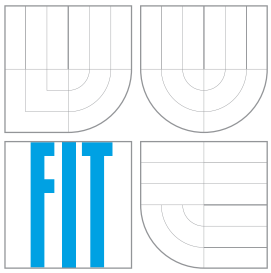
SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE
AUTHOR

Bc. MAREK SUROVIČ

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

AUTOMATIZOVANÁ STATICKÁ ANALÝZA NAD LLVM IR ZAMĚŘENÁ NA DETEKCI MALWARE

AUTOMATED STATIC ANALYSIS OVER LLVM IR FOR MALWARE DETECTION

SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE
AUTHOR

Bc. MAREK SUROVIČ

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Ing. TOMÁŠ VOJNAR, Ph.D.

BRNO 2015

Abstrakt

Tato práce přibližuje problematiku detekce malware, s důrazem na behaviorální detekci formální verifikací. Prezentován je behaviorální detektor založený na inferenci stromových automatů a plán úpravy tohoto detektoru. Původní detektor byl představen v práci Babiće, Reynauda a Songa: Malware Analysis with Tree Automata Inference a cílem úprav je náhrada dynamické taint analýzy binárního souboru malware za statickou analýzu LLVM IR kódu malwaru, který je získán zpětným překladem. Diskutováno je několik návrhů realizace této analýzy.

Abstract

This work revolves around the topic of malware detection, especially behavioral detection through formal verification. A tree automaton inferring behavioral detector introduced by Babić, Reynaud and Song in Malware Analysis with Tree Automata Inference is presented. The main goal of this work is to discuss and plan out changes to the detector, namely substituting dynamic taint analysis of binary executable files with static analysis of LLVM IR code of the same binary files obtained through decompilation. Few approaches to the static analysis are brought forth.

Klíčová slova

Behaviorální detekce malware, LLVM IR, Inference stromových automatů, Statická analýza, Formální verifikace

Keywords

Behavioral malware detection, LLVM IR, Tree automata inference, Static analysis, Formal verification

Citace

Marek Surovič: Automatizovaná statická analýza nad LLVM IR zaměřená na detekci malware, semestrální projekt, Brno, FIT VUT v Brně, 2015

Automatizovaná statická analýza nad LLVM IR zaměřená na detekci malware

Prohlášení

Prohlašuji, že jsem tuto semestrální práci vypracoval samostatně pod vedením prof. Tomáše Vojnara. Dále prohlašuji, že jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Marek Surovič
15. ledna 2015

Poděkování

Zde bych rád poděkoval vedoucímu mé semestrální práce, prof. Tomášu Vojnarovi za jeho odborné rady a vedení.

Obsah

Kapitola 1

Úvod

Informační technologie jsou dnes součástí většiny odvětví lidské činnosti. Od mezilidské komunikace a interaktivní zábavy, až po průmyslnou výrobu a poskytování služeb. S tímto výrazným průnikem do každodenního života však stále častěji vyvstává otázka bezpečnosti a důvěryhodnosti informačních technologií. Díky své složitosti je vývoj a provoz těchto technologií náchylný na chyby a tyto chyby mohou být zneužity k cílům, které nemusí být v souladu s přáními legitimního uživatele. Odpověď na tento problém je dvojí. Zlepšením procesů, které jsou uplatňovány ve vývoji je možné předejít vytvoření některých chyb. Monitorováním provozu a užívání informačních technologií je možné předejít zneužití chyb, které jsou v systému již přítomny, případně minimalizovat škodu, která vznikla úspěšným zneužitím chyby. Softvéru, který je vytvořen primárně za účelem nelegitimního užití nebo poškození informačních technologií říkáme souhrnně *malware*.

Za účelem ochrany informačních technologií a uživatelských dat byli vytvořeny systémy, které odhalují a identifikují malware. Tyto systémy souhrnně označujeme jako bezpečnostní, avšak v kontextu softwarových systémů je běžnějším označením antivirový systém neboli *antivir*. Po představení antivirů na trh nastává ve světě informačních technologií stav konfliktu, kde na jedné straně stojí autoři malware a na druhé straně společnosti, které vyvíjejí antiviry. Nové techniky na jedné straně stimulují vývoj nových technik na straně druhé. Asi nejvýznamějším katalyzátorem tohoto konfliktu je *obfuskace* malware.

Pro to aby mohl malware efektivně plnit svůj účel je vhodné aby jeho přítomnost v hostitelském systému zůstala v tajnosti. Obfuskace je jedním z prostředků jakými tohoto cíle dosáhnout. Malware, který využívá obfuskaci je cíleně upravován tak aby bylo jeho rozeznání od běžného a legitimního software co nejtěžší a aby jeho případné nalezení v jednom systému nepomohlo identifikaci v jiných systémech. Zkoumání malware v kontextu teorie složitosti a vyčíslitelnosti nám ukazuje, že absolutně spolehlivá detekce malware není možná[3] nebo může být velice výpočetně náročná[4]. Vysoké nasazení a úspěšnost antivirů však nasvědčuje tomu, že situace je v praxi o něco pozitivnější a výzkum v oblasti detekce a klasifikace malware je smysluplný.

Historicky byly techniky pro odhalování malware zaměřeny na vyhledávání syntaktických vzorů. Tedy se zaměřovali na to jak malware vypadá. Rozšířenost vysoce efektivní obfuskace na straně malware však v posledních letech činí syntakticky orientovanou detekci stále méně efektivní a je nutné stále více využívat techniky behaviorální[7]. Tyto techniky se zaměřují na to jak se malware chová v prostředí hostitelského systému. Zde však nastává problém se zajištěním bezpečnosti hostitelského systému při pozorování chování malware. Pomoc s tímto problémem by mohla poskytnout odvětví formální analýzy a verifikace, které nabízí techniky pro analýzu vlastností software bez jeho explicitního spuštění. Tako-

váto analýza se běžně v kontextu formální analýzy a verifikace nazývá *statická*. Opakem je analýza *dynamická*, která software spouští v kontrolovaném prostředí a pozoruje jeho chování v něm. Nevýhodou statické analýzy je však fakt, že mnohokrát není k dispozici k analýze vhodná reprezentace konkretního malware. Výpomocí v tomto ohledu by mohl být pokrok v oblasti překladačů programovacích jazyků a zpětného překladu[9].

Tato práce se zabývá zkoumáním možností aplikace metod formální analýzy a verifikace v statické behaviorální detekci a klasifikaci malware. Výchozím bodem je zejména implementace detekce a klasifikace malware pomocí *inference stromových automatů* představená v [1]. Vstupem detektoru v [1] je binární soubor, který je spuštěn a pozorován v kontrolovaném prostředí. Pozorované chování je zaznamenáno ve vhodné reprezentaci a z této reprezentace je odvozen nebo obohacen klasifikátor založený na stromovém automatu. Cílem této práce je nahradit přední dynamickou část zmíněného detektoru vhodnou statickou variantou. Vychozím bodem v tomto případě je mezikód používaný v populární překladačové infrastruktuře LLVM.

Zbytek práce je organizována následovně. Kapitola 2 představuje techniky, kterých využívá moderní malware k obfuskaci a skrývání. Kapitola 3 představuje techniky pro detekci a klasifikaci malware. Kapitola 4 přibližuje strukturu a návrh detektoru z [1] spolu s návrhem změn v přední části detektoru. Konečně kapitola 5 diskutuje další kroky, které je nutné podniknout k dosažení cíle této práce.

Kapitola 2

Obfuskace malware

Jedním z hlavních cílů každého malware je utajení. Hlavním nástrojem k dosažení tohoto cíle je obfuskace. Principielně existuje mnoho způsobů jakým dosáhnout utajení malware v hostitelském systému. Tato kapitola se bude zaměřovat na techniky, které jsou běžné pro velkou část malware a dostatečně obecné na to aby byli aplikovatelné ve většině případů užití malware. Kapitola čerpá zejména z článků [10, 8] a kapitol [6] věnovaným obfuskaci malware.

2.1 Packing a komprese

K tomu aby jsme pochopili co *packing* je a jakou roli hraje komprese v utajení malware je nejprve potřebné aby jsme pochopili v jaké formě se malware šíří a jaké byly první techniky pro detekci malware. Moderní malware se šíří primárně v podobě binárních souborů a ke svému šíření využívá počítačových sítí — zejména internetu. Základem detekčních technik i dnes je databáze vzorů malware. Proti syntaktickým a behaviorálním vzorům v této databázi je možné porovnávat chování procesů a strukturu souborů, které jsou přítomny v hostitelském systému.

Packování je technika, která využíva kompresních algoritmů k změně syntaktických vlastností binárního souboru. Hlavní binární soubor je komprimován a k výsledku je přidána dekomprimační procedura, která v případě spuštění výsledného souboru dekomprimuje hlavní soubor uvnitř a spustí ho. Původně byla tato technika navržena pro redukci objemu dat na přenosných mediích, kde kapacita byla omezená. Dnes má tato technika hlavní uplatnění při redukci objemu dat přenášených přes počítačové sítě a právě v obfuskaci malware.

Komprimace efektivně znemožní některé metody analýzy a detekce malware tím, že znečitelní statická data uvnitř hlavního binárního souboru. Tyto data jsou většinou znakové rětězce a číselné konstanty, které mohou prozradit identitu malware a poskytnout vhled do jeho fungování.

Dalším způsobem jak využít komprese k obfuskaci je zkomprimováním náhodných nadbytečných dat spolu s hlavním programem. Pokročilé kompresní algoritmy jsou citlivé na změnu vstupních dat a i malá změna vstupního souboru vede na dramatické rozdíly ve výstupním souboru. Tohoto jevu je možné s výhodou využít k zvýšení variability vzhledu malware mezi jednotlivými systémy napadenými stejným malwarem.

A v neposledné řadě komprese snižuje velikost výsledného malware, co vede na menší syntaktické vzory a tedy větší šanci, že se podobný vzor vyskytne uvnitř legitimního software, který bude špatně označen jako malware.

2.2 Šifrování a oligomorfismus

Vysoce efektivním a rozšířeným způsobem obfuskace je šifrování. Podobně jako u packingu a komprese je šifrování využito k změně syntaktických vlastností binárního souboru, který obsahuje hlavní tělo malware. K šifrovanému hlavnímu tělu je přidána procedura, které v případě spuštění dešifruje hlavní tělo a spustí ho. Šifrovacím klíčem je typicky údaj, který je proceduře kdykoli dispozici, například velikost souboru s hlavním tělem. Pří napadení dalších systémů je klíč změněn a hlavní tělo malware je šifrováno novým klíčem. Tímto je dosaženo toho, že na každém napadeném systému má malware stejné chování, ale vypadá jinak. Počítačový vir Cascade byl prvním malwarem využívající šifrování k obfuskaci. Svoje hlavní tělo šifroval jednoduchou `xor` šifrou s klíčem odvozeným právě z velikosti hlavního těla.

Ikdyž se hlavní tělo malware mění mezi jednotlivými instancemi napadení, tělo šifrovací procedury zůstává stejné. Tohoto může antivir využít k detekci malware. Klasifikace je však stále obtížná protože dva různé malwary mohou využívat stejnou šifrovací proceduru. Stejně tak legitimní software může využívat šifrování. Nicmnéně je v zájmu autorů malware aby přítomnost jejich výtvoru v napadeném systému zůstala úplně v tajnosti. Z tohoto důvodu byl představen *oligomorfní* malware. Takovýto malware dokáže měnit mezi instancemi napadení i svoji šifrovací proceduru.

Typickou cestou jak implementovat oligomorfismus je mít k dispozici několik nezávislých šifrovacích procedur a ty střídat a měnit šifrovací klíče. Šifrovací procedury, které nejsou využity jsou šifrovány spolu s hlavním tělem. Tímto způsobem je však možné mít „pouze“ stovky možných forem — nebo *mutací* — jednoho malware. Na této úrovni je ještě možné efektivně využívat detekce založené na vyhledávání syntaktických vzorů v binárním souboru. Příkladem oligomorfního malware byl souborový infektor Whale z roku 1990.

2.3 Polymorfismus

Hlavním nedostatkem oligomorfního malware byl relativně malý počet možných mutací jednoho malware. Řešením tohoto „problému“ je *polymorfní* malware. Tento druh malware používá k obfuskaci šifrovací procedury transformace kódu, které mění syntaktické vlastnosti šifrovací procedury, ale nijak nemění její chování. Výsledkem je malware, který mutací mění téměř celou svoji binární podobu a není detekovatelný pomocí čistě statických syntaktických metod. Typickými transformacemi kódu, které polymorfní malware provádí jsou:

Vkládání mrtvého kódu

Do původního kódu jsou vloženy zbytečné instrukce, které nemají žádný vliv na funčnost kódu, ale mění jeho binární podobu. Příkladem může být vkládání `nop` isntrukcí nebo `inc` instrukce následovaná `dec` instrukcí nad stejným registrem.

Přerazení registrů

Kód provádí stejné instrukce nad stejnými daty, které jsou ale po každé mutaci umístěny v jiných registrech. Tato transformace samotná je však detekovatelná pomocí tzv. *wildcard* vyhledávání. Tato technika byla využita například ve viru Win95/Regswap.

Přeuspořádání podprogramů

Velmi efektivní transformace, která přeuspořádává nezávislé části kódu. Pro kód, který lze rozdělit na n nezávislých částí je možné tímto způsobem vytvořit $n!$ různých mutací. Virus Win32/Ghost se svými 10 částmi byl teda schopen vyprodukovat až $10! = 3628800$ mutací.

Substituce instrukcí

Nahrazení jedné nebo více instrukcí jinými instrukcemi se stejnou funkčností. Například `xor` za `sub` nebo `mov` za `push` a `pop` instrukce.

Transpozice kódu

Přeuspořádání instrukcí. V případě, že instrukce jsou na sobě závislé, je pořadí vykonávání obnoveno pomocí nepodmíněných skoků. Pokud přeuspořádávané instrukce nejsou na sobě závislé, obnova pořadí není nutná. Avšak nalezení nezávislých instrukcí je v kontextu malware poměrně nákladné.

Integrace kódu

Technika, která využívá k obfuskaci jiný binární soubor. Malware nejprve rozloží „hostitelský“ binární soubor na nezávislé bloky, pak vloží malwarový kód mezi tyto bloky a nakonec znovu sestaví původní binární soubor. Tato technika byla poprvé představena v populárním viru Win95/Zmist a je jednou z nejvíce sofistikovaných metod obfuskace vůbec.

2.4 Metamorfismus

S nástupem polymorfního malware bylo nutné obohatit antiviry o dynamické detekční metody. Polymorfní malware měnil mutací celou svoji binární podobu, ale po tom co provedl dešifrování bylo možné v paměti počítače objevit neobfuskované hlavní tělo malware, které vykonávalo škodlivou činnost a mutaci šifrovací procedury pro novou generaci. Tedy vyhledáváním syntaktických vzorů v paměti počítače bylo možné polymorfní malware odhalit. Odpovědí na tyto metody detekce byl malware *metamorfní*. Takovýto malware již nevyužívá primárně šifrování k obfuskaci, ale aplikuje transformační techniky polymorfního malware i na hlavní tělo. Tímto se mění syntaxe celého těla malware a jakýkoli čistě syntaktický přístup je obecně málo efektivní.

Implementace efektivního metamorfního malware je netriviální úkol. Na obrázku 2.1 je ilustrován obecný životní cyklus metamorfního malware. Jedna iterace životního cyklu koresponduje právě k jedné mutaci malware. Následuje popis jednotlivých etap.

Lokalizace vlastního kódu

Prvním krokem malware při mutaci je nalezení specifikace vlastního kódu v napadeném systému. Takovouto specifikací může být vlastní binarní soubor, pseudokód nebo přímo zdrojový kód malware. V případě malware, který využívá hostitelské soubory — například Win95/Zmist — je nutné aby byl malware schopen identifikovat napadadené hostitelské soubory a svůj vlastní kód v nich.

Dekódování

Aby metamorfní malware unikl čistě syntaktickým metodám, je nutné aby žádná jeho část nebyla konstantní, tedy ani jeho specifikace. Při dekódování je specifikace zpracována do podoby, z které je možné sestavit mutací novou generaci malware. U specifikace vlastním binárním souborem, je dekódování vlastně zpětným překladem strojového kódu do assembleru, tedy disassembling.

Analýza

Na to aby byl malware schopen provést mutaci je nutné aby měl k dispozici například informace o využití registrů pro obfuskaci jejich přeřazením, nebo informace o toku řízení pro obfuskaci přeuspořádáním podprogramů a transpozicí kódu. V tomto kroku jsou prováděny analýzy dekódované specifikace nutné pro získání těchto informací.

Transformace

V této etapě jsou prováděna obfuskace na základě výsledků předchozích etap. Aplikovány mohou být všechny transformace představené v polymorfních malwarech a některé další, jako je například obfuskace vstupního bodu vykonávání malware, která může ztížit některé druhy dynamické detekce. Součástí transformací může být také odstranění některých předchozích obfuskací a jednoduchá optimalizace kódu za účelem zmenšení velikosti výsledného malware.

Znovusestavení

Posledním krokem při mutaci metamorfního malware je znovusestavení binárního souboru malware. Tímto krokem je mutace dokončena a binární soubor malware je připraven k napadení dalšího systému nebo souboru v systému.

Obrázek 2.1: Životní cyklus metamorfního malware

Kapitola 3

Metody detekce malware

Na problematiku detekce malware je možné se dívat dvojím způsobem. Defenzivním přístupem k detekci malware jsou *intrusion detection systémy*, které monitorují využívání zdrojů a činnost procesů v chráněném systému. Pokud je zaznamenáno potenciálně škodlivé chování u některého z procesů v systému nebo podezřelé využití zdrojů, uživatel je o tom informován a případně jsou podniknuty kroky z potlačení hrozby.

Antiviry představují ofenzivní přístup k detekci malware. Podezřelé soubory nebo procesy jsou aktivně prohledávány a jejich vzhled nebo chování je porovnáváno vůči databázi, která obsahuje vzory známého malware. Metody, které se orientují na vzhled podezřelých souborů a procesů nazýváme *vzhledově–orientované* nebo *syntaktické*. Metody vyšetřující chování nazýváme *behaviorální*.

Tato kapitola přibližuje historický vývoj antivirových detekčních metod, metody detekce používané v moderních antivirech a nové výzkumní trendy v oblasti detekce malware. Část pojednávající o syntaktických detektorech čerpá primárně z kapitol o antivirových skenerech v [6]. Část představující behaviorální detektory čerpá z [5].

3.1 Syntaktická detekce

Historicky první přístup k detekci malware využívaný antivirovým software. Syntaktické detektory se orientují na hledání vzorů malware ve vzhledu — zejména binárních — souborů. K tomuto účelu bylo vyvinuto mnoho detekčních metod, z kterých většina je založena na *vyhledávání vzorů* nebo *emulace kódu* nebo na kombinaci zmíněných.

3.1.1 Vyhledávání vzorů

První a nejjednodušší formou detekce malware je vyhledávání posloupností bajtů v binárních souborech. Detektor je tedy složen z databáze posloupností bajtů, které jsou specifické pro jednotlivé vzorky malware a metody, která tyto posloupnosti hledá v binárních souborech. Velikost hledané posloupnosti a struktura kódu malware má velký vliv na úspěšnsot těchto metod. Následuje několik variací prostého vyhledávání, které vylepšují vlastnosti této metody.

Wildcard vyhledávání

Modifikace prostého vyhledávání posloupnosti bajtů. Mějme posloupnosti bajtů na obrázku 3.1. Horní posloupnost je vybrána z binárního souboru napadeného virem W32/Beast. Dolní

posloupnost je vzor pro wildcard vyhledávání. Značka `?` znamená, že tato část posloupnosti může být vynechána. Značka `%2` zase znamená, že další bajt — `03` v uvedeném příkladě — se může vyskytovat na následujících dvou pozicích za značkou. Zajímavostí této metody je fakt, že i navzdory své jednoduchosti byla schopna odhalit virus Win95/Regswap, který vyžíval obfuskaci pomocí přeřazení registrů.

```
83EB 0274 1683 EB0E 740A 81EB 0301 0000
83EB 0274 ??83 EB0? 740A 81EB %2 0301 0000
```

Obrázek 3.1: Posloupnost bajtů z W32/Beast a wildcard vzor pro jeho detekci

Mismatch vyhledávání

Metoda, která povoluje při vyhledávání vzoru chyby. Pokud nastane při vyhledávání chyba, vyhledávací algoritmus to zaznamená a snaží se najít stejný bajt na další pozici v binárním souboru. Na obrázku 3.2 je na prvním řádku vyhledávaný vzor. Další řádky ilustrují posloupnosti, které vzoru vyhovují pro mismatch, který připouští tři chyby.

```
     vzor: 11 22 33 44 55 66 77 88
pr1:  A3 11 22 33 C9 44 55 66 0A 77 88
pr2:  11 34 22 33 C4 44 55 66 67 77 88
pr3:  11 22 33 44 D4 DD E5 55 66 77 88
```

Obrázek 3.2: Příklad vyhledávání povolující 3–mismatch

Top–and–Tail optimalizace

Jednoduchá optimalizace, která omezuje hledání vzoru na prvních a posledních dvou, čtyřech nebo osmi kilobajtech. Tato technika byla populární v době kdy malware umísťoval svůj kód na začátek a konec souborů.

Smart Scanning

Pokročilá modifikace prostého vyhledávání, která se dokáže vypořádat s některými druhy obfuskace. Při vyhledávání jsou ignorovány zbytečné instrukce, například `nop` a hledání je prováděno v partiích kódu, které neobsahují skoky nebo volání podprogramů.

Algoritmická detekce

Nebo také *malwarově specifická detekce* je rodina metod, které se orientují na specializované vyhledávání vzorů specifických pro jednotlivé malwary. Pro efektivní implementaci těchto metod jsou častokrát vytvořeny specializované jazyky pro popis malware a jeho vyhledávání. Specifikace detekčních algoritmů v takovýchto jazycích jsou přeloženy do mezikódu, který je interpretován antivirem. Typické pro tyto metody je *filtrování*, tedy vyhledávání vzorů v specifických souborech, které napadá jenom malware daného druhu. Pokud daný mawalre využívá šifrování k obfuskaci, je možné provést *x-ray vyhledávání*, které provádí útok na šifru malware pomocí znalosti dešifrovaného těla. Protože malware často využívá slabé šifrovací klíče a jednoduché šifry, je tato metoda poměrně úspěšná.

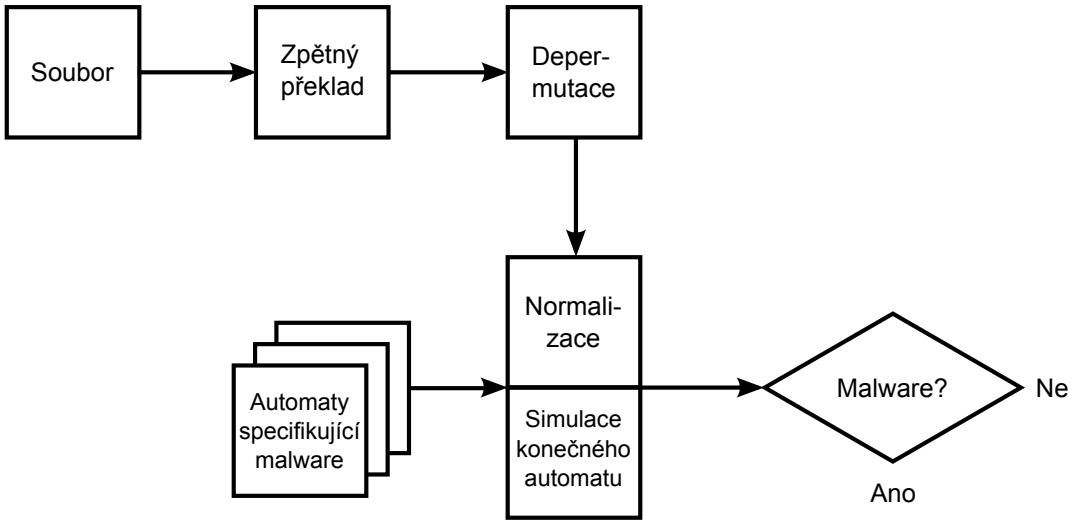
3.1.2 Emulace kódu

Velice efektivní metoda pro detekci jakéhokoli malware, který využívá šifrování k obfuskaci. Kód malware je spuštěn v kontrolovaném prostředí virtuálního stroje, který napodobuje podmínky reálného systému. Malware v tomto prostředí dešifruje svoje konstantní hlavní tělo a prezentuje tak kód, v kterém je možné vyhledat syntaktické vzory již zmíněnými metodami.

Efektivnost této metody je však závislá od implementace virtuálního prostředí, v kterém je malware spuštěn. Malware může použít zbytečných výpočtů k tomu aby učinil emulaci drahou a neefektivní a tím celou metodu znehodnotil. K tomuto byly typicky využívány smyčky se zbytečným kódem a operace nad čísly v plovoucí řádové čárce. Obranou proti tomuto je dynamická detekce zbytečných smyček a emulace operací v plovoucí řádové čárce. Toto však dává malware možnosti detekce emulátoru a adekvátní odpovědi.

3.1.3 Detekce pomocí konečných automatů

Metoda založená na teorii formálních jazyků a konečných automatů, která byla navržena pro analýzu obfuskovaného, zejména polymorfního a metamorfního malware. Malware je přeložen ze strojového jazyka do assembleru, transformován a výsledný kód je brán jako řetězec symbolů, kde symboly jsou jednotlivé instrukce assembleru. Na obrázku 3.3 je uveden schematický popis detektoru založeném na simulaci konečných automatů.



Obrázek 3.3: Schéma detektoru, který využívá konečné automaty

Depermutace má za úkol přeskládat kód obfuskovaný přeuspořádaním podprogramů a transpozicí kódu. Původní pořadí vykonávání instrukcí je částečně obnoveno analýzou instrukcí skoků a voláním podprogramů. Tento proces však ze svého principu není exaktní. Normalizace je prováděna zároveň se simulací konečného automatu a slouží k zahazování zbytečného kódu a reverzi obfuskace provedené substitucí instrukcí. Samotná simulace konečného automatu může být obohacena o zpětné navracení, nebo přechody se speciální sémantikou.

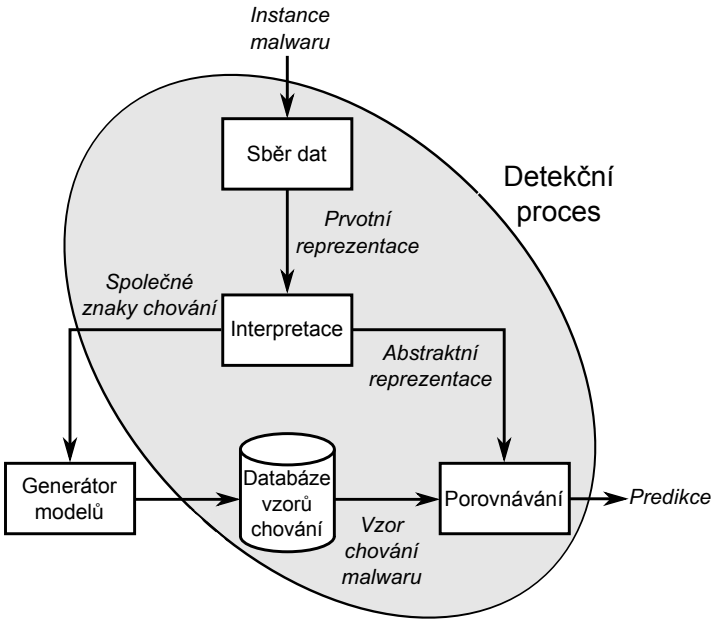
Polymorfní malware je detekován na základě kódu šifrovací procedury, případně na základě dešifrovaného hlavního těla pokud je metoda kombinována s emulací kódu. Metamorfní malware je detekován na základě celého těla, avšak pro detekci pokročilého metamorfního malware jako je Win95/Zmist je zase vhodné použít i emulaci kódu.

3.2 Behaviorální detekce

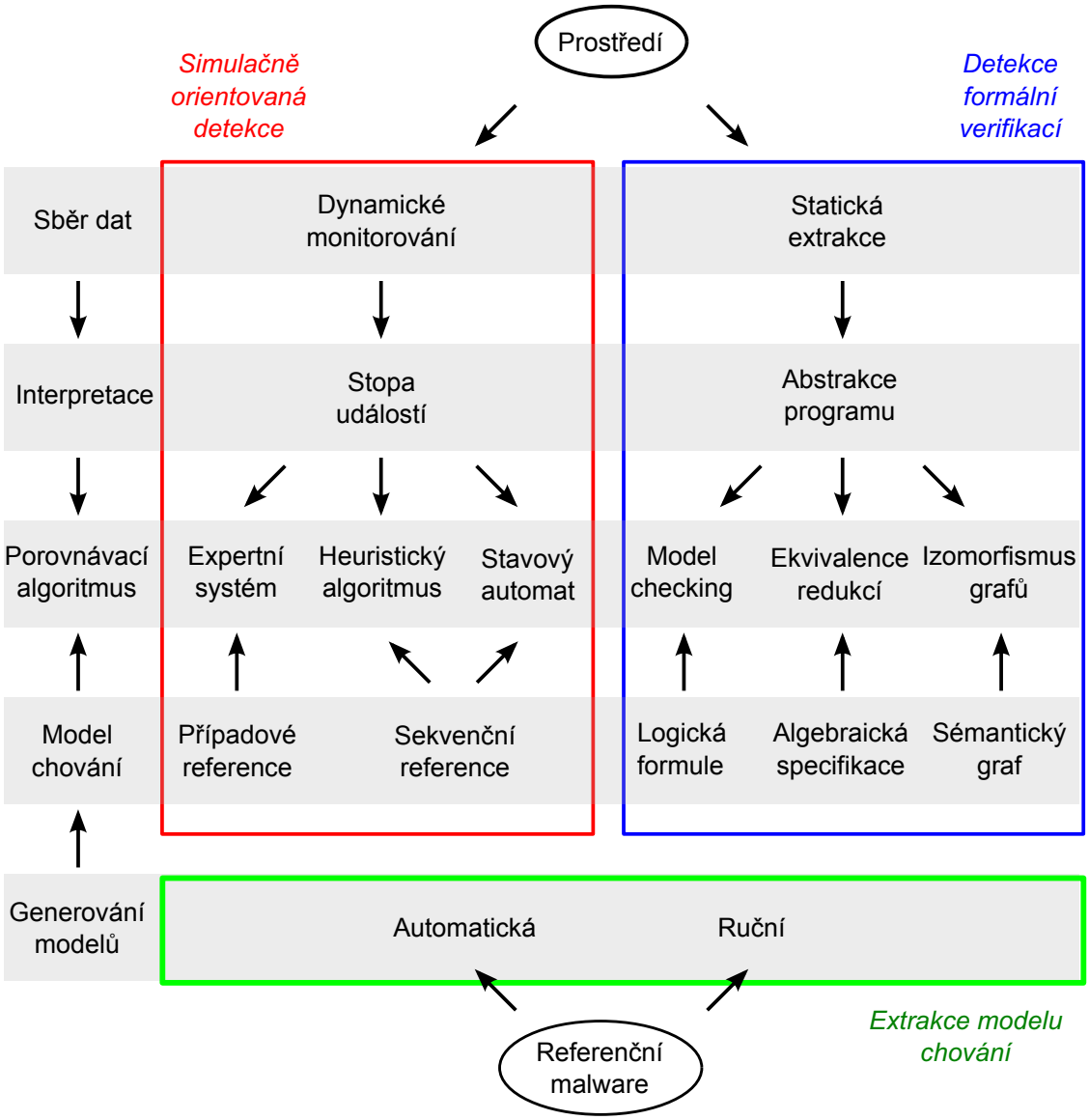
Fundamentálním nedostatkem syntaktických detekorů je jejich závislost na vzorech malware, které jsou odvozeny z vzhledu malware. Pokročilý metamorfický malware jako je Win95/Zmist a {Win32, Linux}/Simile dokazuje, že čistě syntaktický přístup už není dostačující. Prohloubení tohoto problému demonstroval v roce 2007 malware Storm Worm. Autoři nevybavili samotný malware polymorfismem, ale místo toho vytvářeli nové generace malware předem a v pravidelných intervalech je vypouštěli do veřejného internetu. Tímto způsobem tvůrci antivirů neměli přístup k mutačním procedurám, které malware obfuskovali, nemohli efektivně vytvořit syntaktické vzory pro detekci a jejich detektory byly zahlceny velkým množstvím syntakticky různých variant stejného malware.

Tyto a další důvody motivovali vývoj behaviorálních metod, které se orientují na funkčnost a chování malware. Výhodou tohoto přístupu je jeho generičnost, kdy pomocí jednoho vzoru podezřelého chování je možné detekovat celou rodinu malware a odolnost vůči mutacím metamorfního a polymorfního malware. Společnou nevýhodou behaviorálních detektorů je výpočetní složitost detekčních metod a nároky, které kladou na informační systém a jeho uživatele.

Na obrázku 3.4 je ilustrována dekompozice behaviorálního detektoru. Detekční proces je rozdělen do tří podprocesů. Výstupem každého z dílčích podprocesů je popis souboru nebo procesu podezřelého z malwarového chování na vyšší úrovni abstrakce. Proces porovnávání nakonec porovná abstraktní reprezentaci chování vůči databázi vzorů známych malwarových chování. Proces generování modelů obohacuje databázi známých malwarových chování o nové vzory, případně upravuje staré vzory za účelem zvýšení důvěryhodnosti detekce.



Obrázek 3.4: Koncepční schéma behaviorálního detektoru



Obrázek 3.5: Taxonomie stávajících behaviorálních detektorů

Obrázek 3.5 znázorňuje taxonomii behaviorálních detektorů, tak jak je prezentována v [5]. Diagram je rozdělen do dvou částí dle přístupu k detekci: *simulačně orientovaná detekce* a *detekce formální verifikací*. Uvnitř těchto částí jsou do úrovní rozděleny metody a struktury, které detekory využívají při detekci a tvorbě modelu chování. Hlavní inspirací při tvorbě této taxonomie byla oblast testování a formální verifikace software. Také mnoho metod, které jsou v diagramu zahrnuty jsou v těchto oblastech hojně využívány.

3.2.1 Simulačně orientovaná detekce

Simulačně orientovaná detekce je paralelou k black box testování a teda pracuje z principu dynamicky. Analyzováno je chování podezřelého procesu po spuštění, buď v realném systému, nebo v kontrolovaném prostředí, podobně jako je tomu u emulace kódu. Detektory tohoto typu pracují s událostmi, které podezřelý proces vyvolá v systému. Posloupnosti

takovýchto událostí nazýváme *stopy*. Příkladem takovéto stopy je posloupnost systémových volání, které proces vykonal, nebo posloupnost odeslaných síťových paketů.

Stopa je následně interpretována do abstraktní reprezentace. Často využívanou reprezentací jsou *atomickická chování* a jejich posloupnosti. Příkladem takovéhoto chování je například otevření souboru, nebo odeslání paketu. Vzorem podezřelého chování jsou pak různé posloupnosti atomických chování, které byly pozorovány u malware. Tyto podezřelé posloupnosti pak mohou být organizovány do grafů, tabulek pravidel s body nebo stavových automatů. Algoritmy pro porovnávaní mohou pak být heuristické grafové algoritmy, sumy bodů s definovaným prahem nebo simulace stavových automatů.

Hlavním nedostatkem simulačně orientované detekce je fakt, že najednou je možné analyzovat pouze jedno z mnoha možných chování podezřelého procesu. Velká část malware pracuje s jistou mírou nedeterminismu, nebo je jejich chování silně ovlivněno prostředím, ve kterém byl spuštěn. Také malware může své chování upravit v případě, že detekuje spuštění v prostředí, které je monitorováno antivirem.

3.2.2 Detekce formální verifikací

Behaviorální detekce je tradičně spojována s dynamickou analýzou, kdy je podezřelému souboru nebo procesu povoleno vykonávat svoji činnost a realizovat svoje chování. Tento přístup se však zdá krátkozraký když vezmeme v úvahu, že chování malware je plně specifikováno svým kódem. Formální verifikace se zabývá ověřováním zda verifikovaný program nebo systém vyhovuje jisté specifikaci. V kontextu detekce malware je verifikován program podezřelý z malwarového chování a specifikací je vzor chování známých malwarů.

Vstupem detektoru je tedy kód podezřelý z malwarového chování. Z tohoto kódu je odvozena abstraktní reprezentace pokrývající obecně všechna možná chování popsána podezřelým kódem. Toto představuje nespornou výhodu proti jakékoli dynamické analýze. Překážkou v tomto procesu je fakt, že pracujeme s kódem, který lze obfuskovat a tím komplikovat proces získání abstraktní reprezentace kódu. V případě binárních souborů je navíc nutné řešit problém packingu a zpětného překladu.

Volba abstraktní reprezentace podezřelého kódu a specifikace, která bude sloužit jako vzor malwarového chování je úzce spjata s volbou algoritmu pro jejich porovanávání. Zavedenou formou abstraktní rezprezentace kódu jsou *sémantické grafy*. Příkladem takovýchto grafů jsou grafy toků dat a řízení. Vzorem malwarového chování je pak graf stejného typu a porovnávacím algoritmem je algoritmus řešící problém izomorfismu podgrafů.

Jiným přístupem je využití algebraické abstraktní reprezentace a specifikace. Podezřelý kód a vzor chování známého malware je popsán pomocí specializované algebry a porovnávací algoritmus zkoumá jejich ekvivalenci. Podezřelý kód je však nejprve redukován pomocí aplikace ekvivalencí specifikovaných v příslušné algebře. Tímto způsobem je možné efektivně řešit problém obfuskace u metamorfního malware. Různě obfuskované části kódu, které mají stejnou funkčnost vedou redukcí na stejné nebo ekvivaletní výrazy v příslušné algebře.

Posledním přítupem prezentovaným v [5] je porovnávání model checkingem. Podezřelý kód je reprezentován množinou všech stavů, v kterých se může program popsaný kódem vyskytnout a přechody mezi těmito stavy. Tuto strukturu nazýváme stavovým prostorem podezřelého programu. Vzorem malwarového chování je formule v temporální logice, která klade podmínku na vlastnosti stavového prostoru podezřelého programu. Porovnávácí algoritmus pak zkoumá platnost této formule na daném stavovém prostoru.

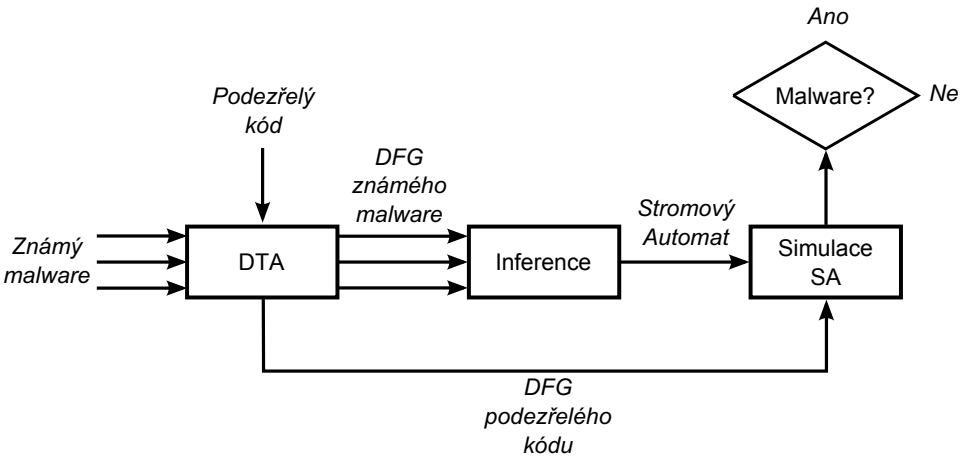
Kapitola 4

Detekce malware inferencí stromových automatů

Nový přístup k behaviorální detekci malware byl představen v [1]. V článku je prezentován přístup kombinující dynamické monitorování podezřelého procesu s metodami formální verifikace, konkrétně inference stromových automatů, které tvoří jádro detektoru. Na obrázku 4.1 je uvedeno schéma tohoto detektoru. Konečným cílem této práce je navrhnout náhradu dynamického monitorování v tomto detektoru statickou extrakcí a odvozením vhodné abstraktní reprezentace.

Statická extrakce kódu z binárních souborů obfuskovaných packingem a šifrováním je však problematika daleko přesahující rozsah této práce. Z tohoto důvodu jsou v této práci uvažovány pouze podezřelé binární soubory, které tímto způsobem obfuskovány nejsou a je možné na ně bez problémů aplikovat zpětný překlad do vyššího formy reprezentace kódu.

V této práci je touto formou interní reprezentace překladačové infrastruktury LLVM. Tato reprezentace byla zvolena díky své schopnosti postihnout strukturální vlastnosti kódu, které lze využít při formální analýze a odvození abstraktní reprezentace. V době psaní této práce existuje několik výzkumných projektů, které se zabývají zpětným překladem a analýzou binárních souborů pomocí této reprezentace. Příkladem nechť jsou [9, 2].



Obrázek 4.1: Schéma detektoru s DTA

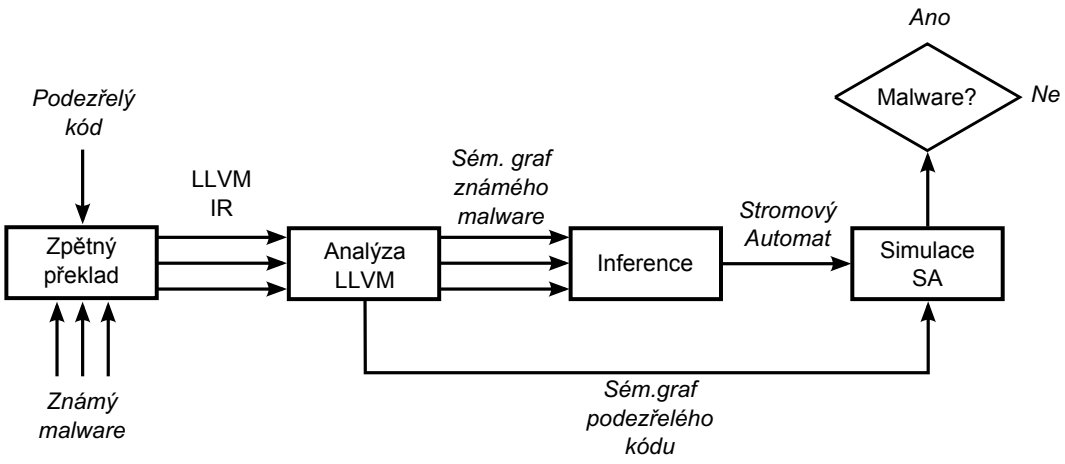
4.1 Struktura detektoru

Detektor prezentovaný v [1] je složen z tří hlavních částí, které lze vidět na obrázku 4.1. Vstupem detektoru jsou podezřelé binární soubory, které spuštěny v kontrolovaném prostředí, kde je zkoumáno jejich chování pomocí *dynamické taint analýzy* nad systémovými voláními. Dynamická taint analýza je technika využívaná k sledování toku dat v programech nebo celých systémech. Data, kterých tok sledujeme nazýváme *tainty* a tok sledujeme od *zdroje taintu* do *cíle taintu*. V našem kontextu jsou tainty výsledky jednotlivých systémových volání, zdroje taintů jsou systémová volání, která výsledky vyprodukovala a cíle taintů jsou systémová volání, která tyto výsledky využívají. Vykonávání programu je řízeno pomocí *propagačních pravidel* na úrovni assemblerových instrukcí. Výstupem DTA je orientovaný graf, kde uzly jsou jednotlivá systémová volání a hrany vedou ze zdrojů taintů k cílům. Na takto definovaný graf lze pohlížet jako na jistou formu grafu toku dat.

Inferencí budujeme jádro detektoru — stromový automat. Stromové automaty jsou stavové automaty, které zpracovávájí grafové struktury, konkrétně stromy. Vstupem inferenčního algoritmu je množina grafů toků dat, které jsou získány aplikací DTA na binární soubory známých malwarů. Vybudovaný stromový automat pak definuje jazyk grafů toků dat malware — jinými slovy definuje chování specifické pro malware. Posledním krokem v detekčním procesu je samotná simulace odvozeného stromového automatu nad grafem toků dat podezřelého binárního souboru.

4.2 Plánované změny detektoru

Dle samotných autorů detektoru je dynamická taint analýza jedním z nejslabších míst detektoru. Grafy toků dat, které analýza poskytuje přímo ovlivňují úspěšnost detekce a v rámci detektoru je to část, která je nejvíce výpočetně náročná. Na obrázku 4.2 jsou znázorněny plánované změny původního detektoru. Hlavním cílem budoucí práce je návrh analýzy LLVM interní reprezentace za účelem vytvoření sémantického grafu, který bude vstupem algoritmu pro inferenci stromového automatu. Jedním z možných řešení je vybudování stejného grafu toků dat jaký buduje dynamická taint analýza v původním návrhu detektoru. Jiným řešením může být vybudování grafu toků dat mezi základními bloky podezřelého kódu.



Obrázek 4.2: Schéma detektoru s analýzou LLVM IR

Kapitola 5

Závěr

Práce prezentuje úvodní část diplomové práce autora, která pojednává o metodách behaviorální detekce malware pomocí metod formální analýzy a verifikace. Prezentován je úvod do problematiky konfliktu mezi autory malware a vývojáři antivirového software. Ze strany autorů malware jsou prezentovány metody obfuskace malware za účelem utajení v napadeném systému. Ze strany druhé jsou prezentovány metody detekce obfuskovaného malware. V rámci detekčních metod jsou stručně popsány metody syntaktické, které se orientují na to jak kód „vypadá“. Hlavním zájmem jsou však metody behaviorální, které zkoumají funkčnost kódu a chování malware. Nakonec je prezentován behaviorální detektor založený na inferenci stromových automatů. Cílem diplomové práce autora je implementace úprav detektoru prezentované v 4.2.

Literatura

[1] Babić, D.; Reynaud, D.; Song, D.: Malware Analysis with Tree Automata Inference. In *CAV'11: Proceedings of the 23rd Int. Conference on Computer Aided Verification*, Lecture Notes in Computer Science, Springer, 2011, s. 116–131.

[2] Chipounov, V.; Candea, G.: Enabling sophisticated analyses of x86 binaries with RevGen. In *International Conference on Dependable Systems and Networks Workshops*, 2011.

[3] Cohen, F.: *Computer Viruses*. Dizertační práce, University of Southern California, 1986.

[4] Filiol, E.: Malicious cryptology and mathematics. *Cryptography and Security in Computing*, 2012, iSBN 978-953-51-0179-6.

[5] Jacob, G.; Debar, H.; Filiol, E.: Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in Computer Virology*, ročník 4, č. 3, 2008: s. 251–266, ISSN 1772-9890.

[6] Konstantinou, E.: *Metamorphic Virus: Analysis and Detection*. Dizertační práce, Royal Holloway, University of London, 2008.

[7] Moser, A.; Kruegel, C.; Kirda, E.: Limits of Static Analysis for Malware Detection. *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, 2007: s. 421 – 430, ISSN 1063-9527.

[8] Schiffman, M.: A Brief History of Malware Obfuscation. [online], Naposledy navštíveno 14. 1. 2015.
URL http://blogs.cisco.com/security/a_brief_history_of_malware_obfuscation_part_1_of_2

[9] 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*, ročník 5, č. 4, 2011: s. 91–106.

[10] You, I.; Yim, K.: Malware Obfuscation Techniques: A Brief Survey. In *BWCCA*, IEEE, 2010, s. 297 – 300.