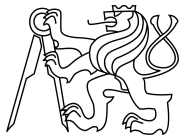
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Deminimizace a deobfuskace malware v jazyce JavaScript
Student:	Václav Hrabě
Vedoucí:	Ing. Radomír Polách
Studijní program:	Informatika
Studijní obor:	Informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce zimního semestru 2018/19

Pokyny pro vypracování

Nastudujte problematiku minimizace a obfuskace jazyka JavaScript a jejích důsledků pro bezpečnost z pohledu antivirové kontroly malware v operačních systémech.
Seznamte se s knihovnami Esprima a Escodegen pro parsování a generování jazyka JavaScript na platformě NodeJS.
Navrhněte, analyzujte a implementujte efektivní postup deminimizace a deobfuskace malware v jazyce Javascript za použití knihoven Esprima a Escodegen. Přihlédněte k tomu, že z pohledu analýzy malware není třeba, aby byl výsledný kód plně funkční.
Implementaci testujte na ukázkách malware dodaných vedoucím práce.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 27. února 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Bakalářská práce

Deminimizace a deobfuskace malware v jazyce JavaScript

Václav Hrabě

Vedoucí práce: Ing. Radomír Polách

14. května 2017

Poděkování

V první řadě bych velmi rád poděkoval mému vedoucímu práce panu Ing. Radomíru Poláchovi, který mi vždy poskytl cenné informace, abych mohl dodělat tuto bakalářskou práci. Poté bych chtěl také poděkovat mé rodině, která za mnou stála v celém průběhu celého studia na této vysoké škole. Chtěl bych i poděkovat těm, kteří jakkoliv přispěli ke vzniku této bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 14. května 2017 .....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2017 Václav Hrabě. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Hrabě, Václav. *Deminimizace a deobfuskace malware v jazyce JavaScript.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato bakalářská práce se zabývá základními praktikami při deminimizaci a deobfuskaci škodlivých programů neboli malwarů. Literární rešerše stručně pojednává o různých druzích malwaru, na které můžeme narazit při stahování programů z internetu nebo hledání informací.

Zvolený problém jsem vyřešil pomocí objektově orientovaného skriptovacího jazyka JavaScript, ve kterém jsou volně přístupné knihovny pro syntaktickou analýzu kódu.

Použité řešení poskytuje možnost základního překladu minimizovaného a obfuskovaného kódu. Díky deminimizovaného a deobfuskovaného kódu lze zjistit pomocí antivirového programu a jeho databáze malwaru, zda je program škodlivý pro naše zařízení či nikoli.

V příloze práce je možné nalézt program ve skriptovacím jazyce bash, který tuto práci testuje. Pokud měníme kód této práce, je možné díky tomuto programu zjistit, zda je překlad stále správný. Další program je také ve skriptovacím jazyce bash, který testuje přidané malwary.

Klíčová slova deminimizace kódu, deobfuskace kódu, implementace programu, vyhledávání malware, bezpečnost uživatelů, JavaScript kód

Abstract

This bachelor thesis deals with basic practies in deminimization and deobfuscation of malware in the JavaScript language. Literary recherche briefly discusses about various kinds of malware that we may encounter when downloading programs from internet or searching for information.

I have solved the problem by using object-oriented scripting language JavaScript, in which libraries for code parsing are freely accessible.

The solution used provides basic translation of minimized and obfuscated code. With deminimized and deobfuscated code, antivirus program and malware database can be detected whether the program is harmful to our device or not.

In the appendix there is a program in the scripting language bash that tests this work. If we modify the code of this work, it is possible with this program to find out whether the translation is still correct. Another program is also in the scripting language bash, which tests the added malware.

Keywords code deminimization, code deobfuscation, program implementation, malware search, user safety, JavaScript code

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Bezpečnost a ochrana je v této době velmi zásadní. Skoro každý používá internet a čelil či zrovna čelí různým škodlivým softwarům. Tyto softwary se snaží získat naše přihlašovací údaje na naše bankovní konto nebo na náš email či jiný námi zabezpečený účet.

Téma bakalářské práce je prospěšné pro širokou škálu lidí, kteří se alespoň někdy připojí na internet.

Z výše uvedených důvodů jsem se rozhodl pro volbu mého tématu, jelikož je toto téma velmi aktuální a potřebné.

V této práci se zabývám analýzou, návrhem a implementací programu pro bezpečnost proti programům typu malware napsaných v jazyce JavaScript.

V první části popíši cíl bakalářské práce.

V druhé kapitole bakalářské práce věnuji popsání základních pojmů, které je potřebné znát, aby bylo lépe porozuměno tomuto tématu bakalářské práce.

Ve třetí části zjišťuji analýzu různých druhů malware a jazyků, ve kterých mohou být napsány.

Ve čtvrté kapitole popíši, jak bakalářská práce funguje jako celek, a jak se používá.

V poslední části ukáži testovací program, který testuje naimplementovaný program, zda se v některé části něco důležitého nezměnilo. Aby výsledky byly pořád stejné a prospěšné.

Kapitola 1

Cíl práce

Cílem práce je implementovat program, který bude měnit kód, jak pro rychlou čitelnost uživatele, ale hlavně také pro antivirový program. Antivirový program pak posoudí, zda program je pro náš počítač škodlivý či nám nezpůsobí potíže.

Implementace bakalářské práce nemusí být plně funkční. Z důvodu jejího velkého rozsahu.

Jsou nastaveny základní části co musí implementace umět:

- změna identifikátorů,
- unární operátory,
- binární operátory,
- vyhodnocení polí a objektů,
- změna new array, new object,
- vyhodnocení if podmínek,
- vyhodnocení switch přepínače,
- změna negativních literálů.

Kapitola 2

Základní pojmy

V této kapitole si projdeme různé pojmy, aby jsme lépe porozuměli významu této bakalářské práce.

2.1 Antivirový program

Antivirový program je počítačový software, který je pro naše počítače zásadní v ochraně proti hackerům a jejich malwarům. Slouží k identifikaci, eliminaci a odstranění škodlivého softwaru neboli malwaru.

Každý antivirový program má svojí databázi škodlivých částí kódu, které mohou uškodit našemu zařízení. Proto každý stažený program projde antivirovou kontrolou, zda nemá v sobě tento kus škodlivého kódu. Pokud je nalezen v tomto programu škodlivý kód, pak antivirový program zahájí jednu ze tří možností:

- Pokus o opravu či vyléčení - odstraní ze souboru škodlivý kus kódu.
- Umístí nakažený soubor do karantény, aby se nemohl dále šířit.
- Smaže dotyčný nakažený soubor.

Každý antivirový program proto musí být aktualizovaný, aby měl vždy aktuální databázi s novými nalezenými kódy virů. Protože pokud v počítači nemáme aktualizovaný antivirový program, tak to je podobné jako bychom ani neměli tento antivirový software na svůj počítač nainstalovaný. Ale útočníci jsou velmi vynalézaví a snaží se být vždy o krok napřed před těmito antivirovými softwary, proto by se měl antivirový program vždy včas aktualizovat.

2.1.1 Antivirové společnosti

Zde si představíme asi ty nejznámější společnosti, jak české tak zahraniční, které se podílejí na vytváření antivirových programů a hlavně jejich databází.

2.1.1.1 Microsoft

Firma Mircosoft je nejvíce známá svým produktem Windows, ale i tato firma se na ochraně našich počítačů podílí. Ve verzích starších a to včetně i Windows 7 je možné nainstalovat produkt Microsoft Security Essentials, který chrání naše elektronické zařízení. Pro novější verze je již ochrana zabudována v základní části balíčku Windows, jmenuje se Windows Defender. Oba tyto produkty jsou připraveny chránit naše počítače před škodlivými malwary, tedy pokud bude vždy instalovaná aktuální aktualizace. [1]

2.1.1.2 Avast Software spol. s r.o.

Česká společnost Avast Software spol. s r.o. dohlíží na naše elektronická zařízení již od roku 1988. Jak již bylo řečeno, toto je česká firma, ale má velké uznání i v zahraničí. Celosvětově chrání před škodlivími útoky na 400 milionů uživatelů počítačových nebo i mobilních zařízení. [2]

2.1.1.3 AVG Technologies

AVG Technologies je opět českou firmou. Tato firma je na českém a zahraničním trhu již od roku 1991. Majitelem této společnosti je již uveřejněná společnost Avast Software spol. s r.o., proto s touto firmou velmi úzce spolupracuje na ochraně proti škodlivým malwarům. [3]

2.1.1.4 ESET, spol. s r.o.

Společnost ESET, spol. s r.o. byla založena v roce 1992 na Československu. Od té doby se snaží naše počítače, ale i mobilní telefony chránit před škodlivými softwary. Její nejznámější produkty jsou ESET NOD32 Antivirus a ESET Smart Security. Jejich produkty, každý útok na naše počítače zaznamenávají a poté společnost ESET každý měsíc informuje na svých stránkách, jaký malware byl tento měsíc nejvíce aktivní. [4]

2.1.1.5 Symantec Corporation

Tato velmi stará společnost se o naše počítače stará již od roku 1982. I v tomto roce již existovaly viry, zde zatím nepáchaly moc velké škody jako dnes, protože počítače byly zatím v rozkvětu a pomalu se dostávaly do domácností pro každodenní používání. [5]

2.1.1.6 McAfee, Inc.

McAfee, Inc. známá americká společnost, kterou má pod svými křídly společnost Intel Corporation od roku 2010. Tato společnost byla založena již 23 let před sjednocením firem. [6]

2.1.1.7 Kaspersky Lab

Mezinárodní společnost nazývaná Kaspersky Lab, se zabývá ochranou elektronických zařízení před škodlivými malwary a útoky hackerů. Tato společnost je v pohotovosti již od roku 1997. [7]

2.2 Hacker

Člověk, který je nazývaný jako hacker, je velmi chytrý a schopný programátor, který je odborníkem na manipulaci nebo úpravy počítačových systémů a sítí. [8]

Zákeřný hacker se oproti hackerovi, snaží dostat do našich počítačů a získat přístupové kódy k našim účtům, využívat naši paměť pro svoje potřeby, údaje o platebních kartách či naše soukromé fotografie. [8]

2.3 Malware a jeho druhy

Slovem malware označujeme nějaký škodlivý software, který autorovi tohoto softwaru má dát přístup k našemu zařízení. A tím získat přístupy na naše různé uživatelské či bankovní účty nebo nějakým jiným způsobem z nás vymámit peníze. [9]

Malware se dělí do několika skupin, kde každá skupina má jiný vliv na naše zařízení či vůbec ani nemáme ponětí, že nějaký takový škodlivý software máme u nás na počítači nebo mobilním zařízení. Mezi nejrozšířenější typy malwaru patří spyware, adware, počítačový virus, trojský kůň, počítačové červy, rootkity a mnoho dalších. Proto si je zde ještě představíme, aby jsme měli představu před jakými malwary se chráníme a jak zjistíme, že jsme napadeni nějakým takovým malwarem, když nemáme antivirový program nebo ho máme, ale ještě se neaktualizoval či nebyl nalezen tento škodlivý software v databázi antivirového programu. O co vlastně můžeme i přijít, když nebudeme opatrní a nějaký takový malware se nám dostane do počítače či mobilního zařízení. [9]

2.3.1 Spyware

Malware typu spyware je software, který sbírá informace ohledně námi navštívených webových stránek nebo jaké programy máme nainstalované na našem zařízení, ale také dokáže posílat naše uložená hesla v počítači. Když se k tomuto spywaru, dokáže do našeho počítače dostat i software Keylogger (2.3.1.1), pak tento spyware posílá autorovi programu i písmena, která jsme stikli na klávesnici. [10]

Tento spyware se k nám do počítače může dostat různými způsoby. Například z infikovaného e-mailu, který jsme právě otevřeli, nebo si ho stáhneme s nějakým volným programem, který nám zlepší výkon našeho zařízení. Tím se

dostane do našeho zařízení a my o tom nic nevíme, že jsme si právě infikovali počítač či mobilní zařízení. [10]

Odhalení tohoto druhu malwaru je velmi obtížné. Pokud se nám začnou spouštět odkazy v jiném prohlížeči nebo se změní naše domovská stránka v oblíbeném internetovém prohlížeči či začnou vyskakovat nesmyslné chybové hlášky. Pak již jsme nakaženi tímto druhem malwaru. [10]

Jak odstraníme takovýto škodlivý software. Musíme mít antivirový program s funkcí na detekci spyware, který rozpozná tento druh malwaru a odstraní ho z našeho počítače či mobilního zařízení. [10]

2.3.1.1 Keylogger

Keylogger se nachází v podkupině spyware. Jelikož tento program dokáže zaznamenávat naše stisknuté znaky na klávesnici. Poté je odeslat zákeřnému hackerovi, který si již může pročítat naše napsaná hesla, osobní a e-mailové zprávy nebo čísla kreditních karet. [11]

Keylogger může být nainstalovaný na naše elektronické zařízení pomocí infikovaného e-mailu či nějakého staženého výhodného souboru. [11]

Odhalit tento škodlivý program v našem počítači je velmi obtížné, protože tento program je velmi chytře schován hluboko v útrobách operačního systému. Pokud se bude zdát, že načítání webových stránek je pomalejší než dřív nebo kurzor myši se bude zasekávat a zobrazování napsaných znaků na klávesnici je výrazně pomalejší než dříve, pak si můžeme být jistí, že jsme nakaženi tímto spywarem. Zobrazování napsaných znaků na klávesnici se ještě ukládá do systému spywaru, proto je zobrazování pomalejší. [11]

Odstranit keylogger z našeho počítače ručně by bylo velmi obtížné a neefektivní, protože nevíme jak se daný program jmenuje. Proto použijeme antivirový program, který má funkci na odhalení keyloggerů, tuto funkci budou mít antivirové programy společností, které byli výše vyjmenovány (2.1.1). [11]

2.3.2 Adware

Adware je bezplatný program, kterému pomáhá reklama. Díky reklamě se může dostat do našeho počítače. Dnes existují různé programy na koukání reklamy a tím získávání peněz, ale i s tímto programem si do počítače můžeme natáhnou tento malware. Díky kterému na nás budou různě vyskakovat reklamy ve vyskovacích oknech či v našem prohlížeči. A je tudíž velmi obtěžující, když každou chvilku na nás vyskakuje reklama a my musíme klikat a zavírat jí. [12]

Tento škodlivý software se šíří stejně jako spyware. Šíří se bezplatnými programy, které nám mají zvýšit výkon počítače nebo infikovaným e-mailem neboli spamem. Může se ale také dostat do našeho zařízení přes nějakou bezpečnostní díru, která ještě nebyla aktualizací operačního systému opravena. [12]

Pokud se nám na počítači začnou zobrazovat vyskakovací okna s reklamou je velmi pravděpodobné, že jsme napadeni tímto druhem malware. Nebo pokud se nám v internetovém prohlížeči změní domovská stránka, tak jako to bylo u spyware, pak je možné, že jsme infikováni tímto adwarem. [12]

Pokud budeme odstraňovat adware, pak si zazálohujme naše důležité soubory, aby jsme o ně nepřišli. Poté spustíme antivirový program, který bude mít funkci pro odstranění adwaru z infikovaného počítače. [12]

2.3.3 Spam

Spamem můžeme nazvat nechtěnou nebo nevyžádanou zprávu, která nám na e-mailového klienta byla zaslána. Dnes již mnoho e-mailových klientů dokáže rozlišovat spamy, které nám přijdou do e-mailové schránky. Pokud nám takový e-mail dorazí do e-mailové schránky, pak je klientem označen jako spam a nebo případně také přesunut do nějaké složky, kde se ukládají e-maily tohoto typu. [13]

2.3.4 Phishing

Phishing je možné přeložit jako rhybaření. Jde o techniku získat z oběti citlivé informace ohledně údajů o platebních kartách či našeho hesla k účtům internetového bankovnictví. [14]

Tento druh malwaru je spjat s e-mailovými schránkami či nějakými chatovacími programy. [14]

Jde o druh malwaru, který dostaneme nějakou zprávou a ta odkazuje na podvodné stránky, ale takové které vypadají velmi podobně či můžeme spíše říct, že jsou na chlup stejné, jako stránky na které se každý den přihlašujeme. Zde na těchto podvodných stránkách jsme vyzváni na zadání citlivých informací o našem účtu a tím může dojít ke krádeži identity či odcizení peněz z našeho účtu. [14]

Pokud dostaneme do našeho e-mailového klienta nějaký e-mail ve kterém jsme vyzváni, aby jsme zadali naše přihlašovací údaje, pak se na 99% bude jednat o phishing. Protože žádné společnosti po nás nebudou chtít naše přihlašovací údaje, jelikož je mají pro sebe přístupné a není potřeba je znovu sdělovat. [14]

2.3.5 Pharming

Pharming neboli farmaření je technika, při které je napaden DNS server a je přepsána IP adresa námi zvoleného serveru a jsme přesměrováni na podvodnou stránku. Zde opět po nás společnost chce přihlašovací údaje jako tomu bylo u malware typu phishing. [15]

2.3.6 Počítačový virus

Počítačovým virusem nazýváme škodlivý program nebo jen část kódu, který se spoustí v našem počítači bez našeho vědomí nebo svolení. Tento malware může někdy jen znepříjemňovat život uživatele různými akcemi, jako jsou zavírání používaných oken či otevírání náhodných aplikací. A v některých případech se počítačový virus snaží získat kontrolu nad napadeným systémem v počítači či mobilním zařízení. [16]

Počítačový virus se šíří podobně jako biologický virus, např. kašel, který se přenáší z člověka na člověka, ale tento virus se přenáší z počítače na počítač. Buďto pomocí nějakého přídavného zařízení flashdisk do kterého zkopírujete nakažený soubor s tímto virem, a poté ve zdravém počítači tento soubor pustíme a tím se pustí počítačový virus, který se pak může kopírovat do dalších souborů, aby nebylo snadné ho zneškodnit. Je možné přenášet tento malware i přes počítačové sítě, či pokud rozklidneme nějaký nakažený e-mail. [16]

Jak poznáme, že máme v počítači počítačový virus, jak bylo výše řečeno. Nějakými neobvyklými akcemi jako zapínání náhodných aplikací. Nebo se nám vypíná připojení internetu nebo se snaží vypnout náš antivirový program, aby se do našeho počítače mohlo dostat více malwaru. Změna názvů dokumentů nebo programů, ale také ztráta cenných dokumentů či fotek. [16]

Možností, jak odstranit počítačový virus z našeho zařízení, je taková že vypneme všechny aplikace a spustíme v antivirovém programu plnou kontrolu systému, aby jsme odhalili, kde se tento malware skrývá, a poté ho necháme antivirovým programem odstranit. [16]

2.3.7 Trojský kůň

Druh malwaru, nazývaný trojský kůň, je nazýván podle trojského koně, kterého nechali postavit řekové. A díky této lsti se dostali do nedobitného města Tróje, které předtím řekové obléhali. Tento malware typu trojského koně je také lstiví a velmi zákeřný. [17]

Trojského koně můžeme do našeho počítače dostat díky naší důvěřivosti jako to udělali trójané. Tento malware se může tvářit jako užitečná funkce, zábavná hra. Trojský kůň může být také vydáván jako bezplatný program, který dokáže odstranit malware z našeho počítače. Může tuto funkce splňovat a tím vyřadit konkurenční škodlivý software, ale stále ho budeme mít v našem počítači či mobilním zařízení a moc o tom nebudeme vědět. [17]

Pokud budeme infikování tímto malwarem, počítač bude přicházet o svůj výkon a tím se i zahřívat, jelikož většinu výkonu bude spotřebovávat právě trojský kůň. Ale tento typ malwaru se nedokáže nijak replikovat a tím být na více místech v počítači najednou. K tomu slouží další malware, který se nazývá počítačový červ, ale o něm v další podkapitole. [17]

Nejlepší způsob, jak nemít ve svém zařízení trojského koně je nevěřit neznámým a neprověřeným aplikacím a používat jen opravdu prověřené aplikace.

Pokud jsme si již stáhli tento druh malwaru do počítače, tak náš aktualizovaný antivirový program by měl dokázat detekovat a posléze odstranit tento malware. Pokud by jsme měli tento malware odstranit ručně, tak musíme odstranit všechny programy, které jsou s tímto trojským koněm propojeny. [17]

2.3.8 Počítačový červ

Počítačové červy jsou velmi snaživé „potvůrky“. Pokud se dostane do našeho počítače, usadí se na síťovém komunikačním kanále počítače a využívá ho pro své potřeby. Dokáže se sám replikovat a díky tomu, že sídlí na síťové komunikaci, tak se dokáže dostat do dalších počítačů pomocí připojení k síti. [18]

Počítačový červ se do našeho počítače může dostat přes síťovou komunikaci s nakaženým počítačem v síti nebo díky nakaženému e-mailu, který otevřeme. [18]

Červy velmi vytěžují síťovou komunikaci, proto když budeme mít podezdření, že naše internetové připojení je pomalé než dříve, pak může začít přemýšlet zda nejsme nakaženi nějakým počítačovým červem. Počítačové červy, také hledají v našem zařízení údaje na bankonvní účet a tím získá nějaký profit pro autora tohoto malwaru. Je také možné, poté co se červ rozmnoží na další zařízení v síti, začne vytěžovat paměť počítače. Například pro šifrování souborů na disku, a pak chce po nás, aby jsme zaplatili, jinak se k šifrovaným datům již nedostaneme. Pokud budeme mít infikovaný počítač, pak se jako u trojských koní velmi zpomalí reakční doba na naše povely či přestane úplně reagovat. [18]

Pokud budeme mít infikovaný počítač nebo elektronické zařízení, je možné pustit antivirový program a tím odhalíme počítačového červa. Poté následuje smazání tohoto škodlivého programu a tím budeme mít již zabezpečný počítač, pokud za pár chvil opět nedostaneme tohoto počítačového červa ze síťové komunikace. [18]

2.3.9 Rootkit

Malware typu rootkit je navržený tak, aby se zákeřný hacker dostal do našeho zařízení a měl práva administrátora. Poté již může schovávat přítomné adresáře, ve kterých jsou nainstalovány škodlivé softwary, aby jejich přítomnost nebyla odhalena běžně dostupnými systémovými prostředky. [19]

Rootkity do našeho počítače můžeme nainstalovat různými způsoby. Například instalací bezpečnostního produktu nebo možností vylepšení nějaké aplikace třetí strany. [19]

Odhalení, že jsme infikováni nějakým malwarem typu rootkit je velmi obtížné a náročné. Jelikož antivirové programy, při hledání těchto malwarů, musí sledovat všechny procesy a jejich závislosti na volání dalších procesů, kam se můžou přesunout, aby nebyli zjistitelné. Ale v dnešní době toto zvládá velká

skupina těchto antivirových programů, hlavně programy od společností uvedených v kapitole Antivirové programy (2.1.1). [19]

Antivirové programy dokážou jen deaktivovat tyto rootkity, poté již musíme tyto programy odstranit ručně. [19]

2.3.10 Ransomware

Tento druh malwaru je velmi zákeřný na dokumenty, fotky a soubory obecně. Jelikož ransomware omezuje přístup k těmto souborům a nemůžeme je použít. Snaží se z nás vymámit velké sumy peněz jako výkupné za naše soubory. [20]

Ransomwary se do našeho počítače dostávají infikovanými přílohami e-mailů, nebo díky naší návštěvě infikované webové stránky tímto malwarem. [20]

Pokud budeme mít infikovaný počítač ransomwarem, tak to zjistíme velmi snadno, tím že nebudeme mít žádný přístup k velkému množství souborů. [20]

Ochrana před tímto malwarem je taková, že máme velmi dobře vybavený antivirový software. Jeho výbava obsahuje odstranění malwaru typu ransomware. Je také dobré si velmi dobře zálohovat dokumenty, například do nějakého cloudu či vlastního serveru. A ne tím způsobem, že máte zálohováno na jednom disku, který je rozdělený na A a B a zálohu provádíme z A do B. Poté stačí jen přeinstalovat daný systém a naformátovat disk na kterém je daný malware. Poté si nahrát opět data zpátky ze zálohy. [20]

2.3.11 Neautorizované změny prohlížečů

Malware, který mění nastavení internetového prohlížeče, dělá toto bez vědomí uživatele prohlížeče. Mění důležitá nastavení a přesměrovává naší domovskou stránku na kterou jsme nikdy neměli v úmyslu navštívit. [21]

Program, který mění nastavení v internetovém prohlížeči, se k nám může dostat pomocí instalace závadného doplňku webového prohlížeče, rozšíření nebo panelu nástrojů. Většinou je nabízen jako vylepšení určitých webových stránek a jejich interaktivního obsahu. [21]

Pokud se budeme dostávat na webové stránky, které jsme nechtěli navštívit, pak si můžeme začít být jisti, že jsme infikováni tímto malwarem. [21]

Chceme-li se zbavit tohoto malwaru měli bychom odinstalovat nedávno nainstalované zásuvné moduly a doplňky webového prohlížeče. A ještě musíme otestovat náš počítač antivirovým programem, aby odstranil možné zbytky tohoto malwaru. [21]

2.3.12 Sociální inženýrství

Nejslabší článek bezpečnosti je člověk, který pracuje na počítači či mobilním zařízení. Tato technika spoléhá na zvědavost, chamtivost, strach nebo lidskou závist. Mezi nejrozšířenější techniky sociálního inženýrství patří Baiting, Phishing, Pretexting, Scareware. [22]

Jak bychom měli odhalit tuto techniku napadení našeho zařízení, je ta že pokud máme kliknout na odkaz či poslat nějaké osobní údaje na uvedenou adresu, na tuto výzvu by člověk měl přistupovat s velkou opatrností. [22]

Tuto techniku nelze odstranit z počítače, jelikož je to na naší zodpovědnost. [22]

2.3.12.1 Baiting

Pokud se chceme podívat na film, tak si musíme stáhnout tuto aplikaci, aby jsme viděli tento film. A tím stáhneme infikovaný soubor do našeho počítače. [22]

2.3.12.2 Phishing

Tento druh malwaru, již byl popsán výše v kapitole 2.3.4.

2.3.12.3 Pretexting

Člověk, který chce od nás získat informace se vydává za někoho jiného, kterému bychom měli věřit a poskytnout mu tyto údaje. [22]

2.3.12.4 Scareware

Technika, která nám řekne, že je v našem počítač či mobilní zařízení nalezen malware a pokud bychom ho chtěli odstranit, tak bychom si měli nainstalovat tento *antivirový program.* [22]

2.3.13 Internetový podvod

V dnešní době existuje mnoho typů internetových podvodů neboli scam, ale všechny mají za úkol jediné a to vylákat z nás osobní údaje nebo zaplatit za služby, kterých se nikdy nedočkáme. Podvodné e-maily jsou nejrozšířenějším typem on-line podvodů. [23]

Takovéto podvody se nejvíce šíří pomocí počítačových virů nebo pomocí spyware, které se skrytě nainstalují do našeho elektronického zařízení. Mají za cíl získat naše hesla, údaje o platebních kartách nebo naše citlivé informace. [23]

Odhalení těchto internetových podvodů je následující. Pokud nám někdo slibuje velmi rychlý zisk velkého množství peněz, jen jediné co musíme udělat je poslat malou zálohu na zprostředkování a zpracování našeho požadavku.

Nebo další možné odhalení můžeme ovlivnit i my tím, pokud máme puštěný aktualizovaný antivirový program a stránka nám říká, že náš počítač je infikovaný nějakým škodlivým softwarem, že si urychleně máme nainstalovat jejich antivirový software, který tento malware odstraní. [23]

Tyto podvody nejdou odstranit, protože se nejedná o fyzický útok na naše počítače. Tedy pokud nás internetový podvod nenabádá k instalaci „lepšího“ softwaru. Tento program již bude pro náš počítač škodlivý. [23]

2.3.14 Makroviry

Makroviry jsou specifickým malwarem, jelikož se nešíří jako počítačové viry a trojské koně nějakým programem, který je spustí, ale přežívají v dokumentech. Dokumenty ale musí mít spustitelná makra, aby se tyto viry při otevření dokumentu mohly aktivovat. [24]

Dnes tento malware je méně vyskytující než před přibližně 15 lety, kdy začínala možnost spustit makra při otevření dokumentu. Dnes již je spouštění maker, při spuštění dokumentu zabráněno. Ale pro jistotu je lepší mít aktualizovaný antivirový program a mít zapnutou antivirovou ochranu maker. [24]

2.3.15 Zadní vrátka

Zadní vrátka neboli Backdoors, umožňuje obejít běžnou autentizaci operačního systému, která za normálních okolností brání uživateli v neoprávněnému užívání operačního systému. Tuto techniku mohou využívat některé malwary, aby zpřístupnily náš počítač či mobilní zařízení útočníkovi nebo natáhly do systému další škodlivý malware. [25]

2.4 Online hrozby

V dnešní době jsou internetové hrozby více početné než před 15 lety. Avšak v té době také každá domácnost nevlastnila počítač či dva a každý člen rodiny neměl chytrý mobilní telefon, na který se škodlivý software mohl nainstalovat.

2.4.1 Historie on-line hrozeb

Historické viry jsou na dnešní dobu pouhou banalitou, kterou by měl dnes již každý antivirový program zjistit a zamezit infikování daným softwarem. [26]

Nejznámější ale i nejprvnější vir se jmenoval Brain, který vytvořili bratři Alviové z Pákistánu a svojí kopií měnil zaváděcí sektor diskety. [26]

Program Creeper, který ze začátku sedmdesátých let 20. století se šířil po Arpanetu připomínal dnešní počítačové červy (2.3.8). Následoval program Reaper, který „červa“ Creepera ničil. [26]

Statusem první počítačový červ je označován Morrisův červ, který byl roku 1988 naprogramován panem Robertem Tappanem Morrisem. [26]

2.4.2 Dnešní on-line hrozeb

Dnešní on-line hrozby používají techniky minimizace a obfuskace, aby nebylo poznat co daný malware dělá. Nejznámější v České republice je malware Danger.

2.5 Minimizace

Minimizace je lehká změna kódu. Když spustíme minimizace nad naším kódem, pak se provedou následující kroky:

- odstranění bílých znaků takzvaných "whitespace",
- odstranění komentářů. [27]

Odstranění bílých znaků takzvaných "whitespace", znepřehledníme náš kód, jelikož se změní v jednu souvislou řádku. Do bílých znaků jsou zařazovány mezery, tabelátory a nové řádky. [27]

Odstranění komentářů je důležité, pokud chceme, aby někdo hned nevěděl co náš program dělá. Každý má ve svých komentářích napsáno, co jaká funkce dělá, aby sám nemusel zjišťovat, co funkce dělá. [27]

2.6 Deminimizace

Deminimizací kódu se snažíme dostat zdrojový kód do původního stavu, ale funguje to jen na vložení bílých znaků do kódu. Kterými změníme kód opět do stavu, který i člověk okem může přečíst a zjistit co kód dělá.

Komentáře budou nenávratně ztraceny, jelikož odstranění komentářů je jen jednostranná záležitost. Nikdy nezískáme z ničeho, co autor měl napsáno v komentářích.

2.7 Obfuskace

Obfuskační technika je ve větším případě používána i s minimizací kódu. Jde o možnost schovat svůj zdrojový kód před zraky lidí, co si buď chtějí přivlastnit náš kód nebo pomůže zákeřným hackerům se dostat do našeho počítače či mobilního zařízení. Obfuskační techniky se rychle zdokonalují.

Pokud použijeme obfuskační techniky na zdrojový kód, pak se provedou následující možnosti:

- odstranění bílých znaků takzvaných "whitespace",

- odstranění komentářů,
- přejmenování názvů,
- vložení mrtvého kódu,
- rozdělení stávajícího kódu,
- duplicita stávajícího kódu,
- substituce,
- štěpení proměnných,
- kódování nebo šifrování dat,
- složení dat za běhu programu. [27]

První dvě možnosti jsou již probrány v sekci Minimizace (2.5). Proto se zde s nimi nebudeme zabývat.

Přejmenování názvů spočívá v přejmenování funkcí, tříd, metod, proměnných na nesrozumitelné řetězce písmen a čísel, aby nebylo poznatelné z prvního pohledu co dělá tato funkce nebo metoda, jelikož většina názvů funkcí, tříd, metod či proměnných vychází z toho co vlastně mají na práci. [27]

Vložení mrtvého kódu má za hlavní úkol odvést naši pozornost od porozumění hlavního kódu. Pokud budeme volat z funkce nějaký kód, ve většině případů neudělá nic s výsledkem. Ale jen díky této funkci budeme přemýšlet co dělá. Také pokud vložíme nějakou proměnnou, která pro náš výsledek nebude mít žádný význam, ale jen si třeba do ní budeme ukládat věci, co jsme spočetli, ale již dávno použili. Člověka to odvede od pozornosti hlavních a důležitých věcí. [27]

Rozdělení stávajícího kódu na více funkcí či tříd. Jedna funkce nebo třída se například rozdělí na další 4 funkce a jsou různě rozesety po zdrojovém kódu. Toto taky člověka, který analyzuje náš kód odvede od kódu a znepříjemní mu čtení kódu. [27]

Duplicita stávajícího kódu, kde se stávající funkce zkopíruje a můžou se i přejmenovat, poté nad nimi může být uplatněna i substituce. Může se zde i uplatnit možné volání těchto funkcí, nejprve se zavolá jedna a při druhém běhu se může zavolat druhá funkce, ale mají stejný výsledek. Tato technika opět odvádí pozornost od pravého účelu obfuskovaného programu. [27]

Substituce neboli záměna, kdy se matematické a logické operace změní na jiné více složité matematické či logické operace, ale ve svém výsledku jsou úplně stejné. Příklad (1.1), který má výsledek (1.2), lze nahradit příkladem (1.3) se stejným výsledkem (1.4) jako v předchozím příkladě (1.1). [27]

$$x = \frac{(18-3)}{5} \tag{2.1}$$

$$x = 3 \tag{2.2}$$

$$x = \frac{(2*3*3*3-3*3)}{5*3} \tag{2.3}$$

$$x = 3 \tag{2.4}$$

Štěpení proměnné, kdy přiřazování hodnot do proměnné je rozdělené do více částí a je prováděno voláním matematických nebo logických funkcí. Tato technika se opět snaží odvést naši pozornost od toho co se vlastně v dané proměnné nachází. [27]

```
function alfa() {
  var x = '';
  x += 'a';
  x += 'b';
  x += 'c';
}
console.log(alfa());
```

V této funkci vidíme velmi jednoduše, že do proměnné x se přiřadí po volání funkce *alfa* hodnota 'abc'. Ale když máme velmi rozlehlou funkci, kde na každém řádku přiřazujeme dlouhý řetězec do jedné proměnné, pak je to velmi nepřehledné a opět nevíme co od dané proměnné můžeme čekat. [27]

Kódování nebo šifrování dat, tato technika se dobře používá, jelikož při této technice opravdu nevíte, co daný kód dělá. Data mohou být zakódována pomocí běžných funkcí, které jsou dostupné v daném vývojovém prostředí. Nebo můžeme zakódovat zdrojový kód jen do symbolů !, { }, [ ], ( ), +, /, pokud jsme v programovacím jazyce JavaScript. Více o tomto kódování v nadcházející kapitole 4.1.2. [27]

Složení dat za běhu programu se provádí následující metodou. Data se například poskládají do nějakého pole, a pak se z tohoto pole načítají a může se nad nimi dělat ještě nějaká transformace a ještě se může použít jen některá část těchto dat. [27]

```
var a = ["x", "j", "o", "h", "a"];
var text = "";
for( i = a.length - 1; i > 0 ; i--) {
    text += a[i];
}
```

2.8 Deobfuskace

Deobfuskace se snaží opět dát obfuskovaný kód do čitelného stavu a trochu pomoci uživateli pochopit, co daný kód dělá. Jelikož je obfuskační technika jen jednosměrná, pak tedy můžeme předpokládat, že pomocí deobfuskační techniky nedostaneme ten samý kód jako před obfuskací.

Kapitola 3

Analýza a návrh

V předchozí kapitole jsme si představili různé pojmy ohledně bezpečnosti uživatelů, a také i skupiny malwarů, které můžeme mít ve svém elektronickém zařízení.

V první části kapitoly si projdeme rozdělení malwarů na jednotlivé programovací jazyky a něco málo si řekneme o rozložení dnešních malwarů.

V druhé části se již zaměříme na knihovny programovacího jazyku JavaScript, které velmi pomáhají s reorganizací kódu, aby mohl být přečtený i uživatelem. A co se skrývá pod zkratkou AST (Abstract syntax tree).

3.1 Rozdělení malware

Programy typu malware, které útočí na naše počítače, jsou psány v různých programovacích jazycích. Některé jsou více přístupné a lépe srozumitelné, proto je v těchto jazycích napsáno více škodlivého softwaru než v jiných programovacích jazycích.

Podle tiskových zpráv společnosti ESET, spol. s r.o., která každý měsíc vyhodnocuje útoky na majitele jejich antivirového programu v České republice. Jsou v popředí malwary napsané v těchto programovacích jazycích:

(v procentech – vybrané programovací jazyky umístění v top 10 nejnebezpečnějších malwarů za posledních 5 měsíců)

- *JavaScript (JS)* - 44 %,
- *Windows API (Win32)* - 22 %,
- *Visual Basic for Applications (VBA)* - 20 %,
- *Java* - 10 %,
- *PowerShell* – 2 %,
- *PDF* - 2 %. [28], [29], [30], [31], [32]

Nejvíce útočících malwarů na počítače či mobilní zařízení s antivirovým programem od společnosti ESET, spol. s r.o. je napsáno v programovacím jazyce JavaScript.

Společnost Avast Software spol. s r.o., která má svůj vlastní antivirový program a pod svými křídly antivirovou společnost AVG Technologies. Potvrdila, že nejvíce malwarů, které se dostávají do našeho počítače či mobilního zařízení je typu JavaScript. Nemají, ale možnost zveřejnit v jakém procentu, z důvodu interních záležitostí.

Z toho můžeme usoudit, že nejvíce malwarů je napsáno v programovacím jazyce JavaScript.

3.1.1 Programovací jazyky

Pokud nemáme představu co vlastně programovací jazyk je, pak zde se nám ho pokusíme stručně představit.

Programovací jazyk je něco podobného jako náš jazyk, kterým se snažíme vyjádřit věci, které nás napadají. Díky programovacímu jazyku se dorozumíváme s počítačem, který tomu danému jazyku rozumí. Programovacím jazykem dokážeme vytvořit algoritmy, které také nazývané jako programy. A podle nich se poté náš počítač řídí. Programovacích jazyků je nepřeberné množství.

Zde jsem vybral pár programovacích jazyků, které se dostaly do tiskových zpráv společnosti ESET, spol. s r.o. za posledních 5 měsíců.

3.1.1.1 JavaScript

Objektově orientovaný programovací jazyk, který viděl světlo světa v roce 1995. Za jeho autora je označován Brendan Eich.

V tomto programovacím jazyku, jak bylo již řečeno dříve, je nejvíce napsaných malwarů.

3.1.1.2 Win32

Tento programovací jazyk, jak již název napovídá, pochází z dílny známé firmy Microsoft. Win32 neboli exe soubory spustitelné hlavně na platformě Microsoft Windows, ale je možné i na dalších typech operačních systémech pustit exe soubory, jen musí být doinstalovány doplňující balíčky.

3.1.1.3 Visual Basic for Applications

Visual Basic for Applications neboli VBA, opět další programovací jazyk od společnosti Microsoft.

Tento programovací jazyk je především používán na psaní makrovirů pro programy Microsoft Office, ve kterých můžeme pustit tyto makra.

3.1.1.4 Java

Programovací jazyk Java je asi nejvíce používaný jazyk na světě, tedy z počítačových jazyků. Člověk, který dal podnět k vytvoření Javy, se jmenuje William Nelson Joy.

3.1.1.5 PowerShell

PowerShell je skriptovací jazyk, první možnost v tomto jazyku začít skriptovat, bylo možno roku 2013. Tento jazyk je produktem společnosti Microsoft.

Jak vyplývá z názvu mělo by jít o *shell*, ale v některých místech se od skriptovacího jazyku *shell* neboli *bash* trochu liší. Například u „roury“, která je u shellovského jazyku textová, u tohoto jazyku je tato „roura“ objektová. Objektová roura, jak již název napovídá, půjde o předávání objektů mezi funkcemi.

3.2 Dnešní malware

V dnešní době se malware velmi rychle zdokonaluje. Pokud se objeví nový typ malwaru, pak firma Avast Software spol. s r.o. se snaží zareagovat na tuto hrozbu a během 24 hodin je na světě obranný prvek, který tento malware neutralizuje.

Zde si představíme 5 největších hrozeb za posledních 5 měsíců. Každý z těchto malwarů má ještě různé podskupiny, ale těmi se zde nebudeme zabývat. Jen si ukážeme například jednu skupinu *JS/TrojanDownloader* má podskupiny *JS/TrojanDownloader.Nemucod* a *JS/TrojanDownloader.Agent.PQT*.

3.2.1 JS/Danger

Tento malware má za hlavní úkol, když se dostane do našeho počítače, stáhnout další malwary. Pokud začne stahovat další malware, tak převážně malware typu ransomware (2.3.10), který omezuje přístup k souborům v našem počítači či elektronickém zařízení. [28]

3.2.2 JS/TrojanDownloader

Jak již název tohoto malwaru napovídá půjde o stahování trojských koní (2.3.7) do našeho počítače. Tento malware je typu spyware (2.3.1) a je možné ho dostat do počítače jako *freeware* nebo *shareware* aplikaci. [33]

3.2.3 Java/QRat

Tento malware je typu trojského koně (2.3.7), který útočníkovi otevře zadní vrátka pro vzdálený přístup do našeho počítače. [32]

3.2.4 Win32/Adware

Tento již známý výraz Adware z podkapitoly 2.3.2, značí že nám „zvýší“ rychlost počítače. Pokud budeme vidět více reklam, bude na tom mít zásluhu tato skupina malwaru.

3.2.5 JS/Chromex.Submelius

Tento název vůbec nenapovídá co by příslušný malware mohl dělat. Ale jde o typ malwaru trojský kůň (2.3.7). [34]

3.3 JavaScript knihovny

Knihovny jsou algoritmy, které již jsou napsané a jejich kód je zdokumentován a můžeme ho používat pro naše větší pohodlí. Protože to co člověk dělá, již někdo před ním určitě zkoušel.

Zde si představíme knihovny, které jsou použity v této bakalářské práci.

3.3.1 Knihovna Esprima

Tato knihovna je pro JavaScript přínosná tím, že velmi pomáhá programátorovi změnit kód v přehledný AST (3.4). Zde si představíme její implementovaný modul a funkci, která se jmenuje *parse*. [35]

3.3.1.1 Esprima.Syntax

Zde nalezneme všechny možné typy uzlů. Můžeme je i podle toho identifikovat v *case* podmínce či *if* podmínce jako například Esprima.Syntax.Literal. [35]

Veškerou syntax této knihovny lze najít na dostupné stránce [36].

3.3.1.2 Esprima.parse

Použitím parse funkce v knihovně Esprima, se převede pomocí těch příkazů kód na AST (3.4). Zde můžeme vidět jak se používá a jaký výstup z této funkce je. [35]

```
var program = 'var answer = 42';
esprima.parse(program);
```

Výstup této posloupnosti příkazů jako AST (3.4):

```
{ type: 'Program',
  body:
   [ { type: 'VariableDeclaration',
       declarations: [Object],
       kind: 'var' } ],
  sourceType: 'script' }
```

3.3.2 Knihovna Estraverse

Knihovna Estraverse pomáhá, když něco chceme změnit v uzlu. Jelikož základní kód této funkce je následující. Je zde podfunkce enter, kterou si můžeme upravovat. Zda nadcházející část programu je funkce nebo ne. Pokud ano, pak tento *if* přemění *currentScope* na type *function*, který označuje, že jsme ve funkci. V podfunkci leave, je to velmi podobné jako u podfunkce enter. Mění *currentScope* z ohledem na to, kde se nacházíme. [37]

```
estraverse.traverse(ast, {
    enter: function(node, parent) {
        if (/Function/.test(node.type)) {
            currentScope = scopeManager.acquire(node);
        }
    },
    leave: function (node, parent) {
        if (/Function/.test(node.type)) {
            currentScope = currentScope.upper;
        }
    }
});
```

3.3.2.1 Estraverse.replace

Pokud použijeme funkci *replace* v knihovně Estraverse, pak můžeme nahrazovat uzly použitím *return*. A to tím způsobem, pokud nějaká podsekce podmínky *if* splňuje naše požadavky, pak tedy vrátíme celý blok kódu, a tím přepíšeme začátek tohoto uzlu. [37]

3.3.2.2 Estraverse.traverse

Funkce *traverse* v knihovně Estraverse, je podobná jako funkce *replace*. Jen s tou změnou, že zde nepoužíváme *return*. Jen s možností převedení nějakého uzlu na jiný. Jako tomu bude u binárního operátoru + v následující kapitole pod označením 4.6.2 Binární operátory. [37]

3.3.3 Knihovna Escodegen

Tato knihovna pomáhá při generování z AST (3.4) na zobrazitelný kód. [38]

3.3.3.1 Escodegen.generate

Tato funkce zařídí, aby jsme z AST (3.4) viděli kód. V níže napsané ukázce můžeme vidět AST (3.4) a z tohoto stromu vznikne jen nepatrný součet „40 + 2“. [38]

```
escodegen.generate({
    type: 'BinaryExpression',
    operator: '+',
    left: { type: 'Literal', value: 40 },
    right: { type: 'Literal', value: 2 }
});
```

3.3.4 Knihovna Esvalid

3.3.4.1 Esvalid.isValid

Jak již název knihovny a název funkce vysvětluje půjde o nějakou validitu. Tato funkce zkontroluje, zda vložený kód je v pořád či nikoli a bylo možno ho převést na AST (3.4). [39]

3.3.5 Knihovna Escope

3.3.5.1 Escope.analyze

Pomocí této knihovny Escope a její funkce *analyze*, která nalezne možné uzly v algoritmu neboli v programu. Pokud narazí na proměnnou, pak si ji uloží do příslušného *setu*. [40]

3.3.6 Knihovna Fs

Tato knihovna pomáhá s prací se soubory. Aby jsme je mohli jednoduše přečíst. Uložit mezivýpočty do souborů, zda jsme se neodchýlili od správného výsledku.

3.4 Abstract Syntax Tree

Abstract Syntax Tree značíme zkratkou AST. Ve kterém můžeme vidět rozložený zadaný kód, díky již zmíněné knihovně *Esprima* a její funkce *parse*. [41]

Každý uzel v AST obsahuje typ a další věci, ale ty se mění podle toho, jaký je typ tohoto uzlu. Pokud bude základního typu *Literal*, pak bude ještě obsahovat *value* a *raw*. Ještě máme druhý základní typ a to *Identifier*, s kterým bude ještě v tomto uzlu hodnota uložená v *name*. [41]

Zde si ukážeme vybrané zástupce, jakými typy může nabývat uzel a jaké další hodnoty bude mít.

- *Literal* - value, raw,
- *Identifier* - name,
- *ArrayExpression* - elements,

- *ObjectExpression* - properties,
- *UnaryExpression* - operator, argument, prefix,
- *BinaryExpression* - operator, left, right,
- *LogicalExpression* - operator, left, right,
- *BlockStatement* - block,
- *BreakStatement* - label,
- *EmptyStatement* - nemá žádné další hodnoty,
- *IfStatement* - test, consequent, alternate,
- *SwitchStatement* - discriminant, cases,
- *VariableDeclaration* - declarations, kind. [41]

Zde si ještě předvedeme, pro lepší představu, dva kusy kódu nejvíce používaných typů a to *Literal* a *Identifier*. [41]

```
{
    "type": "ExpressionStatement",
    "expression": {
    "type": "Literal",
    "value": 10,
    "raw": "10"
    }
}
{
    "type": "ExpressionStatement",
    "expression": {
        "type": "Identifier",
        "name": "a"
    }
}
```

Kapitola 4

Realizace

V této kapitole si nastíníme funkce naimplementované práce, které pomáhají převádět malware na lépe zjistitelné škodlivé či neškodlivé programy.

Koukneme se, jak se převádí rovnice *x += 1*; (-=) na rovnici typu *x = x +/- 1*. Také potřebné je převést znaky [] na *undefined*, poté opět díky naimplementovaným unárním a binárním operátorům jdou tyto znaky převádět na čísla nebo znaky. Velmi potřebné shledávám změnit proměnné, které jsou definované, aby se opět mohlo velmi dobře pracovat s unárními a binárními operátory. A podíváme se, kde tyto proměnné vyhledávat a měnit. V této kapitole také narazíme na úryvky kódu, jak změnit existující pole na jednu hodnotu. To samé jen s vyhledáváním přes klíč v *Objectu*. Když již budeme mít změněné a vyhodnocené proměnné v podmínkách *if* a *switch*, pak vrátíme příslušný blok, to bude obsahem další sekce této kapitoly. Předposlední věc, kterou zde nalezneme je, jak změnit příkazy *new Array*, *new Object*. V poslední řadě nahlédneme, jak změnit hodnotu, která na konci bude záporná.

4.1 Změna znaků

V první řadě se podíváme na změnu operátorů +=; (-=), abychom lépe mohli pracovat s hodnotami. Pak zde budeme mít znaky jako [], s kterými můžeme skládat různá slovama a věty. Díky těmto změnám budeme moci v pozdější části, lépe pracovat s unárními a binárními operátory, které máme ve vedlejším souboru naimplementované.

4.1.1 Operátory += a -=

Pokud narazíme na operátory += nebo -=, které jsou v AST (3.4) označeny jako *AssignmentExpression*. Pomocí knihovny estraverse s funkcí traverse a dalšími připsanými požadavky, převedeme rovnice *x += a* na *x = x + a*; (*x -= a* na *x = x - a*).

Rovnice $x \mathrel{+}= a$ se zobrazí v AST (3.4) takto:

```
"type": "ExpressionStatement",
"expression": {
    "type": "AssignmentExpression",
    "operator": "+=",
    "left": {
        "type": "Identifier",
        "name": "x"
    },
    "right": {
        "type": "Literal",
        "value": "a",
        "raw": "\"a\""
    }
}
```

Přidané požadavky vypadají takto, jen se porovná, zda na pravé straně vyskytuje *Literal* nebo *Identifier* nebo *MemberExpression*. Do pravého operátoru *node.right.operator* přidáme podle základního operátoru += nebo -=, buď + nebo -.

Zde můžeme vidět úryvek kódu, ze změny těchto operátorů. Pokud pravá část se rovná typu *Literal*:

```
variable.value = node.right.value;
variable.raw = node.right.raw;
var left = {
    type: node.left.type,
    name: node.left.name
}
node.right.type = "BinaryExpression";
node.right.operator = "+";
node.right.left = left;
node.right.right = variable;
```

4.1.2 Znaky []

Znaky [] s typem *ArrayExpression*, tyto znaky se změní na hodnotu *undefined*.

Tyto znaky se velmi dobře změní na čísla. Soubor čísel +!![] se převede na číslo 1. Znaky [] se změní na *undefined*. Poté zde máme unární operátor *!*, který změní hodnotu *undefined* na hodnotu *false*. Opět tu máme unární operátor *!* a *false* se změní na *true* a díky unárnímu operátoru *+* se true převede na číslo *1*. Kdyby zde bylo *false*, pak by číslo bylo *0*. [42]

V tabulce 4.1 můžeme vidět, jak si vytvořit různé číslovky pomocí znaků +, !, [], (). Pomocí těchto znaků je také možno vytvořit i písmena. Jak se tvoří vybraná písmena, si ukážeme v tabulce 4.2. [42]

Tabulka 4.1: Vybraná čísla prezentovaná pomocí znaků [42]

Číslo	**Vybraná čísla převedená na znaky +, !, [], ()**
0:	+![]
1:	+!![]
2:	+!![]+!![]
3:	+!![]+!![]+!![]
4:	+!![]+!![]+!![]+!![]
5:	+!![]+!![]+!![]+!![]+!![]
6:	+!![]+!![]+!![]+!![]+!![]+!![]
10:	+((+!![])+[]+(+![]))
11:	+((+!![])+[]+(+!![]))
12:	+((+!![])+[]+(+!![]+!![]))
13:	+((+!![])+[]+(+!![]+!![]+!![]))
100:	+((+!![])+[]+(+![])+(+![]))
101:	+((+!![])+[]+(+![])+(+!![]))
121:	+((+!![])+[]+(+!![]+!![])+(+!![]))
1000:	+((+!![])+[]+(+![])+(+![])+(+![]))

Tabulka 4.2: Vybraná písmena prezentovaná pomocí znaků [42]

Písmeno	**Vybraná písmena převedená na znaky +, !, [], ()**
a:	(![]+[])[+!![]+[]]
e:	(!![]+[])[+!![]+!![]+!![]+[]]
f:	(![]+[])[+![]+[]]
i:	([][[]]+[])[+!![]+!![]+!![]+!![]+!![]]
l:	(![]+[])[+!![]+!![]+[]]
n:	([][[]]+[])[+!![]+!![]+!![]+!![]+!![]+!![]
r:	(!![]+[])[+!![]+[]]
s:	(![]+[])[+!![]+!![]+!![]]
t:	(!![]+[])[+![]+[]]
u:	(!![]+[])[+!![]+!![]+[]]
o:	([][([][[]]+[])[+!![]+!![]+!![]+!![]+[]]+([][[]]+[])[+!![]+!![]+!![] +!![]+!![]]+(![]+[])[+!![]+!![]]+(!![]+[])[+[]]+(![]+[])[+!![]+!![] +!![]+!![]+[]]+(!![]+[])[+!![]]]+[])[+!![]+!![]+!![]+!![]+!![]+!![]]

4.2 Změna identifikátorů

Velmi potřebná je změna identifikátorů. Pokud si deklarujeme proměnnou pomocí *var x = 10;*, pak tato proměnná se uloží sama pomocí knihovny *escope*. Kterou si poté můžeme vytáhnout ze *set* pomocí *get(node.name)*. Tento *set* nalezneme v námi zvolené proměnné, kterou si deklarujeme na začátku programu a tato proměnná nám ukazuje na hlavní uzel.

```
var currentScope = scopeManager.acquire(ast);
```

V proměnné scopeManager nalezneme zanalyzovaný AST (3.4).

```
var scopeManager = escope.analyze(ast);
```

Poté když se dostaneme do funkce, pak proměnná *currentScope* ukazuje na tuto funkci. Když budeme hledat nějakou proměnnou, která je deklarovaná v nějakém nadřazeném uzlu. Pomocí příkazu, níže uvedeném, se dostaneme do nadřazeného uzlu a tento uzel, pak prohledáme, pokud zde opět nenajdeme, pak opět ukážeme na nadřazenější uzel dokud nenajdeme tuto příslušnou proměnnou.

```
currentScope = currentScope.upper;
```

Pokud vyměňujeme proměnnou s typem *Identifier* za typ *Literal*, pak nalezneme tuto proměnnou a změníme *node.type* na *Literal* a smažeme *node.name*. Přidáme hodnoty *node.value* a *node.raw* do uzlu a nastavíme podle nalezených hodnot v *currentScope*.

Pokud proměnná, kterou chceme vyměnit bude, mít type *ArrayExpression*, pak změníme *node.elements* na opět nalezenou hodnotu v *currentScope*. Velmi podobně toto platí i u typu *ObjectExpression*, kde změníme *node.properties*.

4.3 Přiřazení do proměnné

Pokud máme deklarovanou proměnnou pomocí *var*, pak knihovna *Escope* s funkcí *analyze* toto zjistí a deklaruje si tuto proměnnou s příslušnou hodnotou. Můžeme, ale také narazit na přiřazení do proměnné. Zde je potřeba zjistit, kde se tato proměnná nachází a přiřadit do ní.

Pokud jsme ve funkci a narazíme na přiřazení pomocí operátoru =. Pak si nalezneme v *currentScope* příslušnou proměnnou a vložíme tuto hodnotu do jejího pole nazvaného *defs*. Jelikož nemusí být tato proměnná v této funkci deklarována, musíme poté prohledat i nadřazené uzly.

Zde můžeme vidět přiřazení do pole *defs*. Kde variable je vyhledaná proměnná v příslušném *currentScope*. Zvolená proměnná *vari* je nastavena podle struktury proměnné v poli *defs*.

```
variable.defs.push(vari);
```

4.4 JavaScript pole

JavaScript pole se převádí ve funkci *traverse* z knihovny *estraverse*, z *[0, 1, 58, 3, 4][1]* na *1*. Pokud výsledná složka je typ *Literal*, hodnota se uloží do *node.value*, složka typu *Identifier* se ukládá do *node.name*. Zde můžeme vidět, jak toto probíhá.

Do proměnné *variable* se uloží položka, která je hledaná v poli.

```
var variable = node.object.elements[node.property.value];
```

Poté jak bylo již řečeno podle typu nalezené hodnoty, se hodnota uloží do příslušné proměnné. Zde můžeme vidět, když typ je *Literal*.

```
node.value = variable.value;
node.raw = variable.raw;
```

4.5 JavaScript object

JavaScript object také jako JavaScript pole ve funkci *traverse* z knihovny *estraverse*, kde je připsáno vyhledávání v JavaScript object podle zvoleného klíče. Pokud budeme mít object *{'a': '0', 'b': '1'}['a']*, který se převede na *0*. Klíče nalezneme v *node.object.properties[X]*, kde X měníme v cyklu for, který projde přes všechny *properties*. Poté musíme zjistit, zda typ klíče nalezený v *node.object.properties[X].keys.type* je roven *Literal* nebo *Identifier* a porovná buď *node.object.properties[X].keys.value* či *node.object.properties[X].keys.name* s vyhledávaným klíčem. Zde můžeme vidět jedno porovnání:

```
if( node.object.properties[i].key.type
      === Syntax.Identifier
&& node.object.properties[i].key.name
      === node.property.name ){
```

Pokud nalezená hodnota je *Literal*, pak se již použije tento if:

```
if( node.object.properties[i].value.type
      === Syntax.Literal ){
    node.type = node.object.properties[i].value.type;
    node.value = node.object.properties[i].value.value;
    node.raw = node.object.properties[i].value.raw;
}
```

4.6 Operátory

Hlavní a nejpoužívanější znaky, které můžeme najít jsou různé operátory.

Zde si ukážeme, jak v této bakalářské práci řešíme unární a binární operátory na které můžeme v některých malware narazit.

Tyto operátory nalezneme v souboru operatory.js, kde nalezneme oba druhy operátorů.

4.6.1 Unární operátory

Unární operátory, které jsou v této bakalářské práce naimplementované, nalezneme v tomto přehledu:

- operátor +,
- operátor -,
- operátor ~,
- operátor !,
- operátor ++,
- operátor --,
- operátor typeof.

Operátor + v matematických rovnicích nepředstavuje velké zamyšlení. Pokud se objeví je číslo kladné. Velký důraz spíš přikládáme k tomu, že tento operátor dokáže převést hodnotu *false* na číslo *0*, a také hodnotu *true* na číslo *1*, což je velmi potřebná vlastnost k rozluštění malwaru, který se skládá ze znaků, které jsme si představili v kapitole 4.1.2.

Záporný unární operátor - nejvíce využijeme v posledním tématu této kapitoly 4.10. Kde si řekneme proč je potřebné převést číslo do kladné hodnoty, právě když narazíme na negativní číslo.

Operátor ~ mění hodnotu. Pokud máme hodnotu *0* převrátí jí na *-1*. Pracuje na principu N=x a rovnice -(N+1). Takže pokud si dosadíme N=2 pak rovnice vyjde *-3*.

Tento operátor ! je jako unární operátor nejvíce potřebný u podmínek. Také tento unární operátor můžeme vidět u znaků [], které překládá na hodnotu *false* nebo *true*.

Inkrementační operátor ++, který k hodnotě přičítá číslo jedna. Je to velmi zkrácená funkce k rovnici $x = x + 1$, stačí jen napsat $++x$ a výsledek je stejný.

Dekrementační operátor --, velmi podobný operátor jako operátor ++. Jen s tím rozdílem, že tento operátor odebírá jednotku z příslušné hodnoty. Rovnice $x = x - 1$, stačí jen napsat $--x$.

Unární operátor typeof nemá stejnou vlastnost jako předešlé unární operátory. Tento operátor dokáže zjistit typ zadané hodnoty. Návratové hodnoty tohoto operátoru jsou *number*, *string*, *boolean* a mnoho dalších.

4.6.2 Binární operátory

Binární operátory v souboru operatory.js jsou v objektu *boperators*, kde podle klíče nalezneme hledaný operátor. Zde si představíme vybrané binární operátory, které nalezneme ve škodlivých programech zvaných malware:

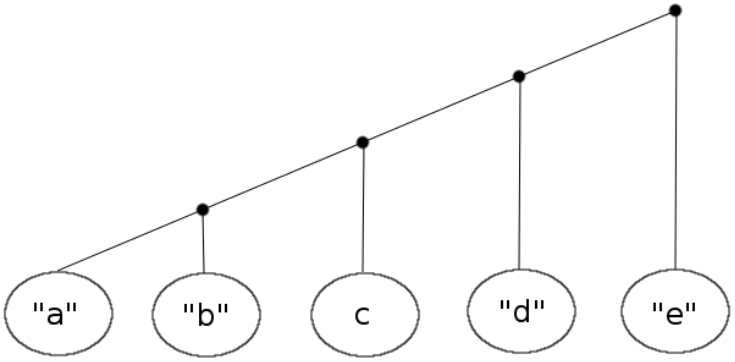
- operátor +,
- operátor -,
- operátor *,
- operátor /,
- operátor %.

Binární operátor + je možné použít jak u čísel na sčítání, pak ale také u písmen na zřetězení. U hodnot si každý dokáže představit, jak sčítání probíhá. Ale co dělat, když máme dvě písmena? Je to velmi jednoduché. Sečtením dvou písmen vznikne řetězec těchto písmen. Například "a" + "b" se zřetězí jako "ab". Ale co když budeme mít dvě písmena a u toho nějakou číslici. Tak se převede číslovka na *string* a připojí se na požadované místo v řetězci. "a" + 10 + "b" se převede na řetěz "a10b". Když budeme mít více písmen a mezi tím neznámou, která nemá žádnou hodnotu, provedeme zřetězení takto: "a" + "b" + c + "d" + "e" na řetězec "ab" + c + "de". Velmi si musíme dát pozor na zřetězení pokud máme "a" + "b" + c + "d" + "e", můžeme vidět na obrázku 4.1, jak se toto zřetězení rozloží. Jelikož na pravé straně převedeme "d" + "e" na "de", pak je potřeba odstranit uzel, který si drží hodnotu "d" a nahradit ho uzlem, který odkazuje na poduzle na levé straně, můžeme toto převedení vidět na obrázku 4.2 a ukázku kódu můžeme vidět zde.

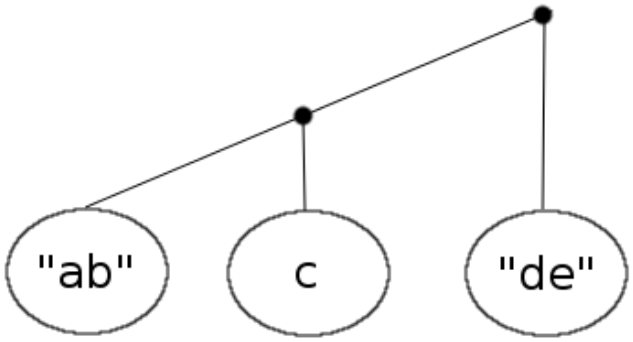
```
var ret = Operatory.boperators[node.operator]
        (node.left.right.value, node.right.value );
node.right.type = "Literal";
node.right.value = ret;
node.right.raw = ""+ret+"";

if( node.left.left ){
    node.left = node.left.left;
}
```

Operátor mínus funguje jen u číselných hodnot. Není tomu jako u binárního operátoru plus, který funguje jak u *stringů*, tak také u číslic.



Obrázek 4.1: Rozložení zřetězení



Obrázek 4.2: Převedené zřetězení

Násobení je binární operátor s kterým se rychle dostame do vysokých číselných hodnot. Jiné využití nemá.

Operátor dělení je opačnou funkcí operátoru násobení. Všichni známe operaci dělení. Tak si ji nemusíme představovat.

Pěkná operace je modulo, která se snaží dostat vždy pod zavedenou hodnotu modula. Ukážeme si na příkladu $1001\%2 = 1$, což je velmi rychlé nalezení jednoduché hodnoty.

4.7 JavaScript If

Pokud narazíme na podmínku *If* značenou pod typem *IfStatement*.

Začneme porovnávat jejich podmínku, která je již vyhodnocena z dřívějšího možného dosazení proměnných, a také vyhodnocena z unárních či binárních operátorů. Když tato podmínka je vyhodnocena jako *true*, je vrácen uzel *node.consequent*, který přepíše hlavní uzel se začátkem *IfStatement*.

```
if( node.test && node.test.value === true ){
    return node.consequent;
}
```

Pokud nenarazíme na žádnou podmínku, která je vyhodnocena jako *true*, je vrácen uzel označovaný jako *else* a označen v AST (3.4) jako *node.alternate*.

```
else if( node.test && node.test.value === false
    && node.alternate
    && node.alternate.type != Syntax.IfStatement ){
            return node.alternate;
}
```

Jelikož v podmínkách *if* nemusí existovat žádná podmínka s vyhodnocením *true* ani žádný uzel s označení *else*. Pak se bude vracet prázdný uzel, který vymaže *if*, který nemá žádný význam pro výsledek.

Toto je první možnost vrácení prázdného uzlu:

```
else{
    var ret = {
       "type": "EmptyStatement"
    }
    return ret;
}
```

Zde můžeme vidět i druhou možnost vrácení prázdného uzlu, která je o nějaký ten řádek zkrácena:

```
else{
    return {
       "type": "EmptyStatement"
    };
}
```

Ale toto v žádném případě nepoužijeme, jelikož podmínka nemusí být *true*, ale může volat nějakou funkci z počítače, z které pak vyhodnotí zda je nebo není podmínka *true* nebo *false*. Jako můžeme vidět zde.

```
if ( new Function('var ubsir = sadtoxydro.GetDrive(\'C\');
     if(typeof ubsir.FreeSpace == \'number\') return true;
     else return false;')()
   ) {
```

4.7.1 JavaScript Conditional

Hojně využívaný je zkrácený *if*, který můžeme vidět v následujícím kódu. Tato podmínka je utvořena takto: (podmínka) ? {splněna podmínka} : {NEsplněna podmínka}. V AST (3.4) nalezneme tento uzel označený jako *ConditionalExpression*.

```
a==1 ? console.log('if') : console.log('else');
```

Pokud podmínka je *true*, poté je vrácen první uzel pomocí *node.consequent*. Když podmínka není vyhodnocena jako *true* je vrácen druhý uzel pomocí *node.alternate*. V následujícím kódu můžeme vidět porovnání a vrácení jednotlivých uzlů.

```
if( node.test && node.test.value === true ){
    return node.consequent;
}
else {
    return node.alternate;
}
```

4.8 JavaScript Switch

Pokud narazíme na uzel, který je typu *SwitchStatement*, pak se bude jednat o *switch*. Při implementaci si musíme dát pozor na to, že nemusí být v každém *case* ukončovací příkaz *break*. Proto vždy se musíme koukat, zda jsme našli nebo nenašli tento příkaz.

V této bakalářské práci budeme změnu *switch* podmínky nahrazovat tímto způsobem. Vždy si nahradíme podmínku podle, které budeme hledat. Pak projíždíme v cyklu všechny *case* položky. Pokud nalezneme požadovanou položku, pak její vnitřek také projedeme cyklem. Pokud narazíme na ukončovací příkaz *break*, pak vrátíme blok do kterého si ukládáme položky, které splnily kritéria. A to že byly v požadovaném *case* a jejich poloha byla před ukončovacím příkazem.

Pokud v příslušném *case* nenalezneme ukončovací příkaz *break*, pak se přesuneme na další *case*, který je v pořadí. A opět si ukládáme položky do bloku, který pak vrátíme. Zde můžeme vidět blok do kterého si ukládáme položky, které splnily kritéria, a které pak vracíme.

```
var block = {
    type: "BlockStatement",
    body: []
}
```

Poté si zde ukážeme, jak se vkládá do těla tohoto bloku. Vždy musíme vložit na konec tohoto těla, aby jsme poté nezměnili výsledek.

```
var vel = block.body.length;
block.body[vel] = node.cases[i].consequent[y];
```

4.9 JavaScript New

Příkaz *new* vytvoří novou instanci, kterou uživatel definuje. V této bakalářské práci je naimplementováno *new Array*, *new Object*. Všechny tyto uzly mají typ *NewExpression*. Zde si obě po jedné představíme.

4.9.1 New Array

Jako první si představíme *new Array* neboli *nové pole*. Jak bylo řečeno tento typ nalezneme podle typu, který je *NewExpression*, ale k tomu ještě přísluší podmínka taková, že *node.callee.name* se musí rovnat *Array*. Tato podmínka musí být splněna, aby se jednalo o pole.

V následujícím kódu můžeme vidět, jak se *new Array* převádí do *Array*. Nejprve změníme typ na *ArrayExpression*, a poté převedeme všechny argumenty, které byly obsaženy v *new Array*, do elementů nového *Array* neboli pole. A odstraníme nepotřebné uzly, aby se nám nijak nepletly do AST (3.4).

```
node.type = "ArrayExpression";
node.elements = node.arguments;
delete( node.arguments );
delete( node.callee );
```

4.9.2 New Object

Následující odstavec bude pojednávat o vytvoření *nového objektu* neboli *new Object*. Jak zjistit, že se jedná o nový objekt. Musí splňovat podobnou podmínku jako již dříve zmíněné *nové pole*. Kdy *node.callee.name* se musí rovnat *Object*.

Opět, zde máme ukázku, která pojednává o převádění z *new Object* na *Object*. Je to velmi stejný převod jako u pole. Pokud v *new Object* má nějaké argumenty, pak vrátíme celý uzel. Pokud je bez argumentů, pak jen změníme typ uzlu na *ObjectExpression* a vyplníme *node.properties* prázdným objektem.

```
if( node.arguments[0] ){
    return node.arguments[0];
}
else{
    node.type = "ObjectExpression";
    node.properties = [];
}
delete( node.arguments );
delete( node.callee );
```

4.10 Literal negative

V poslední části naimplementovaného programu se nachází změna záporných *Literálů* na kladné. Jelikož žádné záporné literály nejdou vypsat pokud jsou dané jako čísla (-1). Pokud má *Literál* nastavenou hodnotu jako *string*, který je v záporu („-1“), tak tento *string* se vypíše.

V kódu níže můžeme vidět, jak funkce na změnu negativního *Literálu* funguje. Zkontrolujeme zda daná hodnota je záporná. Pokud ano, pošleme tuto hodnotu do připravené funkce v unárních operátorech, které jsme si představili v 4.6.1 v této kapitole.

```
if( typeof(node.value) != "string" && node.value < 0 ){
    var ret = Operatory.uoperators["-"](node.value);
    node.value = ret;
    node.raw = ""+ret+"";
}
```

Kapitola 5

Testování

V této kapitole si představíme programy, které mi vždy pomáhaly s testováním mého bakalářského programu.

5.1 Testovací programy

Testovací programy jsou vytvořeny v důsledku toho, že nikdo nechce pořád sám zkoušet testovat program, který implementoval.

Tyto programy jsou napsané ve skriptovacím jazyce bash.

5.1.1 Testovací program na základní části

Tento testovací program má za úkol mě vždy upozornit, když něco předělám ve své práci, zda jsem se neodchýlil od požadovaného výsledku. Můžeme si tento program prohlédnout v příloze B.

Funguje na principu: „pusť program a porovnej s požadovaným výsledkem“. Výsledek jsem si vždy před prvním spuštěním zkontroloval, zda je výsledek správný. Pokud nesouhlasí porovnávaný soubor se souborem, který má být dobře, pak se tedy: „ukonči a ukaž u kterého testu si skončil“. Do proměnné Test se vkládají názvy souborů jako pole, které chceme zkontrolovat, ale musí být vždy vytvořen soubor s kterým se výstupní soubor porovná.

Zde můžeme vidět ukázku ze vstupu a také výstupu.

Vstup do programu:

```
var a=1;
if (a==1) console.log('if');
else console.log('else');
if (a!=1) console.log('if');
else console.log('else');

a==1 ? console.log('if') : console.log('else');
a!=1 ? console.log('if') : console.log('else');
```

Výstup z programu:

```
var a = 1;
console.log('if');
console.log('else');
console.log('if');
console.log('else');
```

Poté tento výstup se porovná s připraveným souborem, který má v sobě požadovaný výsledek.

Zde je výběr souborů a jejich popis.

- *Array.dat* - výběr z pole,
- *empty_function.dat* - základní minimizace,
- *hiero_II.dat* - rozsáhlý test znaků +, !, [], (),
- *if.dat* - test na podmínku if,
- *li.dat* - testování vestavěných funkcí,
- *negative.dat* - záporné literály,
- *new_II.dat* - new Array,
- *new_IV.dat* - new Object,
- *pokus_III.dat* - změna identifikátorů,
- *pole.dat* - přidávání do pole,
- *pole_II.dat* - výpis z pole a objektu,
- *promenne.dat* - nastavení proměnných,
- *promenne_II.dat* - binární operátor +,
- *property.dat* - vyhodnocení vlastnosti length,
- *simple_assignment.dat* - nahrazení operátorů +=,
- *switch.dat* - vyhodnocení switch přepínače,
- *variables.dat* - nastavení a změna identifikátorů.

Tyto všechny vybrané testy implementace vyhodnocuje. Některé testy mají ještě další podtesty v příslušně označených souborech.

5.1.2 Testovací program na ukázky malware

Testovací program na ukázky malware nepracuje stejným způsobem jako předchozí testovací program. Jelikož nemohu odhadnout, jak by měl vypadat tento malware. Tento testovací program můžeme vidět v příloze C.

Pracuje na principu: „pusť program a pokud máš výstupový soubor přičti jedna k zadané proměnné a na konci vypiš kolik souborů si vypsal z kolika možných“.

Jelikož byly v této bakalářské práci implementované jen základní části, nemůžeme očekávat, že všechny dodané malwary budou nějak změněny. Jejich změna se může vztahovat jen na deminimizaci.

Jeden malware musel být rozdělen do více souborů (soubory jsou pod označením 27B1 v adresáři *malware/zaklad/*) z důvodu, velkého množství uzlů, které se definovaly.

Zde můžeme vidět ukázku z tohoto malwaru, který mi byl poskytnut na testování.

Vstup do programu:

```
var fromCharCode = new ActiveXObject(
String.fromCharCode(87)+String.fromCharCode(115)
+String.fromCharCode(99)+String.fromCharCode(114)
+String.fromCharCode(105)+String.fromCharCode(112)
+String.fromCharCode(116)+String.fromCharCode(46)
+String.fromCharCode(83)+String.fromCharCode(104)
+String.fromCharCode(101)+String.fromCharCode(108)
+String.fromCharCode(108));
```

Výstup z programu:

```
var fromCharCode = new ActiveXObject('Wscript.Shell');
```

Tento kus malware má za úkol spustit příkazový řádek, jak můžeme vidět. Ale s dalšími parametry například můžeme smazat nějaké důležité soubory nebo změnit práva u složky.

Závěr

V práci jsem se zabýval analýzou, návrhem a implementací programu pro bezpečnost proti programům typu malware napsané v jazyce JavaScript. Vytvořený program pomáhá s deminimizací a deobfuskací programů v našem počítači. Které se poté můžou vyhodnotit pomocí antivirového programu jako škodlivé nebo neškodné pro náš počítač či mobilní zařízení.

Všechny vyjmenované cíle této bakalářské práce se mi podařilo naplnit:

- změna identifikátorů,
- unární operátory,
- binární operátory,
- vyhodnocení polí a objektů,
- změna new array, new object,
- vyhodnocení if podmínek,
- vyhodnocení switch přepínače,
- změna negativních literálů.

V budoucnosti by bylo možné program rozšířit o další vyhledávání škodlivého softwaru. Jelikož každým dnem, může přijít člověk s další možností, jak obejít toto vyhledávání a tím tedy nainstalovat škodlivý software k nám do počítače.

Literatura

[1] Microsoft Corporation: *Stažení Security Essentials [online]*. [cit. 2017-04-26]. Dostupné z: https://support.microsoft.com/cs-cz/help/14210/security-essentials-download

[2] AVAST Software s.r.o.: *Poznejte Avast [online]*. [cit. 2017-04-26]. Dostupné z: https://www.avast.com/cs-cz/about

[3] AVG Technologies CZ: *AVG – naší snahou je vaše bezpečnost online [online]*. [cit. 2017-04-26]. Dostupné z: http://www.avg.com/cz-cs/profile

[4] ESET, spol. s r.o.: *O společnosti ESET [online]*. [cit. 2017-04-26]. Dostupné z: https://www.eset.com/cz/o-nas/

[5] Symantec Corporation: *Securing the Cloud Generation [online]*. [cit. 2017-04-26]. Dostupné z: https://www.symantec.com/

[6] CN Invest a.s.: *McAfee [online]*. [cit. 2017-04-26]. Dostupné z: http://www.zive.cz/mcafee/sc-773/default.aspx

[7] Kaspersky Lab: *Jsme tu, abychom chránili svět. [online]*. [cit. 2017-04-26]. Dostupné z: https://www.kaspersky.cz/about-us

[8] AVAST Software s.r.o.: *Hacker [online]*. [cit. 2017-04-27]. Dostupné z: https://www.avast.com/cs-cz/c-hacker

[9] AVAST Software s.r.o.: *Malware [online]*. [cit. 2017-04-27]. Dostupné z: https://www.avast.com/cs-cz/c-malware

[10] AVAST Software s.r.o.: *Spyware [online]*. [cit. 2017-04-27]. Dostupné z: https://www.avast.com/cs-cz/c-spyware

[11] AVAST Software s.r.o.: *Keylogger [online]*. [cit. 2017-04-27]. Dostupné z: https://www.avast.com/cs-cz/c-keylogger

[12] AVAST Software s.r.o.: *Adware [online]*. [cit. 2017-04-27]. Dostupné z: https://www.avast.com/cs-cz/c-adware

[13] AVAST Software s.r.o.: *Spam [online]*. [cit. 2017-04-27]. Dostupné z: https://www.avast.com/cs-cz/c-spam

[14] AVAST Software s.r.o.: *Phishing [online]*. [cit. 2017-04-27]. Dostupné z: https://www.avast.com/cs-cz/c-phishing

[15] Internet Info, s.r.o.: *Rhybaření střídá pharming [online]*. [cit. 2017-04-27]. Dostupné z: http://www.lupa.cz/clanky/rhybareni-strida-pharming/

[16] AVAST Software s.r.o.: *Počítačový virus [online]*. [cit. 2017-04-30]. Dostupné z: https://www.avast.com/cs-cz/c-computer-virus

[17] AVAST Software s.r.o.: *Trojský kůň [online]*. [cit. 2017-04-30]. Dostupné z: https://www.avast.com/cs-cz/c-trojan

[18] AVAST Software s.r.o.: *Počítačový červ [online]*. [cit. 2017-04-30]. Dostupné z: https://www.avast.com/cs-cz/c-computer-worm

[19] AVAST Software s.r.o.: *Rootkit [online]*. [cit. 2017-04-30]. Dostupné z: https://www.avast.com/cs-cz/c-rootkit

[20] AVAST Software s.r.o.: *Ransomware [online]*. [cit. 2017-04-30]. Dostupné z: https://www.avast.com/cs-cz/c-ransomware

[21] AVAST Software s.r.o.: *Neautorizované změny prohlížečů [online]*. [cit. 2017-04-30]. Dostupné z: https://www.avast.com/cs-cz/c-browser-hijacker

[22] AVAST Software s.r.o.: *Sociální inženýrství [online]*. [cit. 2017-04-30]. Dostupné z: https://www.avast.com/cs-cz/c-social-engineering

[23] AVAST Software s.r.o.: *Internetový podvod [online]*. [cit. 2017-04-30]. Dostupné z: https://www.avast.com/cs-cz/c-scam

[24] PEŠA, Radim: *Makroviry v dokumentech MS Office [online]*. [cit. 2017-04-29]. Dostupné z: http://webserver.ics.muni.cz/bulletin/articles/247.html

[25] *Počítačový vir [online]*. [cit. 2017-04-30]. Dostupné z: http://cechlovsky.webzdarma.cz/prace/pocitace/viry/viry.htm

[26] Internet Info, s.r.o.: *Počítačové viry: 20 let tam a zpátky [online]*. [cit. 2017-04-26]. Dostupné z: http://www.lupa.cz/clanky/pocitacove-viry-20-let-tam-a-zpatky/

[27] ČERMÁK, Miroslav: *Obfuskace a základní obfuskační techniky [online]*. [cit. 2017-04-23]. Dostupné z: http://www.cleverandsmart.cz/obfuskace-a-zakladni-obfuskacni-techniky/

[28] ESET, spol. s r.o.: *Za každou druhou kybernetickou hrozbou v listopadu stál malware Danger [online]*. [cit. 2017-04-25]. Dostupné z: https://www.eset.com/cz/o-nas/pro-novinare/tiskove-zpravy/za-kazdou-druhou-kybernetickou-hrozbou-v-listopadu-stal-malware-danger/

[29] ESET, spol. s r.o.: *Škodlivý kód Danger byl i v závěru roku 2016 největší virovou hrozbou v Česku [online]*. [cit. 2017-04-25]. Dostupné z: https://www.eset.com/cz/o-nas/pro-novinare/tiskove-zpravy/skodlivy-kod-danger-byl-i-v-zaveru-roku-2016-nejvetsi-virovou-hrozbou-v-cesku/

[30] ESET, spol. s r.o.: *Hrozba malware Danger v Česku klesá, v lednu posiloval downloader Agent [online]*. [cit. 2017-04-25]. Dostupné z: https://www.eset.com/cz/o-nas/pro-novinare/tiskove-zpravy/hrozba-malware-danger-v-cesku-klesa-v-lednu-posiloval-downloader-agent/

[31] ESET, spol. s r.o.: *Malware Danger opět posiluje, v únoru zdvojnásobil svůj podíl na internetových hrozbách v Česku [online]*. [cit. 2017-04-25]. Dostupné z: https://www.eset.com/cz/o-nas/pro-novinare/tiskove-zpravy/malware-danger-opet-posiluje-v-unoru-zdvojnasobil-svuj-podil-na-internetovych-hrozbach-v-ce/

[32] ESET, spol. s r.o.: *Za každou čtvrtou internetovou hrozbou v březnu stál malware Danger, posiloval trojan Java/QRat [online]*. [cit. 2017-04-25]. Dostupné z: https://www.eset.com/cz/o-nas/pro-novinare/tiskove-zpravy/za-kazdou-ctvrtou-internetovou-hrozbou-v-breznu-stal-malware-danger-posiloval-trojan-javaqrat/

[33] ESET, spol. s r.o.: *JS/TrojanDownloader.Nemucod [online]*. [cit. 2017-04-25]. Dostupné z: http://www.virusradar.com/en/JS_TrojanDownloader.Nemucod/detail

[34] ESET, spol. s r.o.: *JS/Chromex.Submelius [online]*. [cit. 2017-04-26]. Dostupné z: http://www.virovyradar.cz/en/JS_Chromex.Submelius/detail

[35] *Esprima [online]*. [cit. 2017-04-24]. Dostupné z: https://esprima.readthedocs.io/en/latest/

[36] *Module Syntax [online]*. [cit. 2017-04-23]. Dostupné z: http://definitelytyped.org/docs/esprima--esprima/modules/esprima.syntax.html

[37] *Usage [online].* [cit. 2017-04-24]. Dostupné z: `https://github.com/estools/estraverse/wiki/Usage`

[38] WOLF, Markus: *ECMAScript code generator [online].* [cit. 2017-04-24]. Dostupné z: `https://github.com/estools/escodegen`

[39] FICARRA, Michael: *confirm that a SpiderMonkey format AST represents an ECMAScript program [online].* [cit. 2017-04-24]. Dostupné z: `https://github.com/estools/esvalid`

[40] SUZUKI, Yusuke: *Index [online].* [cit. 2017-04-24]. Dostupné z: `http://estools.github.io/escope/`

[41] *Appendix A. Syntax Tree Format [online].* [cit. 2017-04-20]. Dostupné z: `https://esprima.readthedocs.io/en/latest/syntax-tree-format.html`

[42] KUMMEL, Roman: *Cross-Site Scripting v praxi [online].* [cit. 2017-04-26]. Dostupné z: `https://www.soom.cz/data/Cross-Site_Scripting_v_praxi__Roman_Kummel.pdf`

Příloha A

Seznam použitých zkratek

AST Abstract syntax tree

JS JavaScript

PDF Portable Document Format

VBA Visual Basic for Applications

Win32 Windows API

Příloha **B**

Testovací program na základní části

```
#!/bin/bash

Baka="Bakalarka_vXX.js"
Test=( ... )
for((i=0; i<${#Test[*]}; i++ ))
do
    echo "Test $i ${Test[$i]}"
    node "$Baka" malware/Zaklad/"${Test[$i]}".dat

    if [[ -e malware/Zaklad/${Test[$i]}.dat.vystup
        && -e malware/Zaklad/${Test[$i]}_dobre.dat ]]; then
        DATA=$( diff malware/Zaklad/${Test[$i]}.dat.vystup
        malware/Zaklad/${Test[$i]}_dobre.dat )
    else
        echo "Test $i ${Test[$i]} Se nezdařil !!!!!
            nemáme nějaký soubor"
        exit 1
    fi

    if [[ -z "$DATA" ]]; then
        echo "Test $i ${Test[$i]} Proběhl dobře"
    else
        echo "Test $i ${Test[$i]} Se nezdařil !!!!!"
        exit 1
    fi
    rm "malware/Zaklad/${Test[$i]}.dat.vystup"
done
```

Příloha C

Testovací program na ukázky malware

```
#!/bin/bash

rm malware/Ukazky/*.vystup

Baka="Bakalarka_vXX.js"

Test=( ... )
x=0
for((i=0; i<${#Test[*]}; i++ ))
do
    echo "Test $i ${Test[$i]}"
    node "$Baka" malware/Ukazky/"${Test[$i]}".dat

    if [[ -e malware/Ukazky/${Test[$i]}.dat.vystup ]]; then
        x=$( echo "$x + 1" | bc -l )
    fi
done

echo "max ${#Test[*]} / uděláno $x"
```

Příloha D

Obsah přiloženého média

- `Bakalarka.js` ..... Bakalářská práce
- `Tester.sh` ..... Testovací program pro základní soubory
- `Tester_malware.sh` ..... Testovací program pro ukázky soubory
- `src`
  - `builtIn.js` ..... Vestavěné funkce
  - `operatory.js` ..... Unární a Binární operátory
- `kniha`
  - `BP_Václav_Hrabě_2017.pdf` ..... Text práce ve formátu PDF
  - `BP_Václav_Hrabě_2017.tex` .. Zdrojová forma práce ve formátu LaTeX
  - `Desky_Václav_Hrabě_2017.pdf` ..... Desky ve formátu PDF
- `malware`
  - `Ukazky` ..... Ukázky malware
  - `Zaklad` ..... Základní soubory