VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

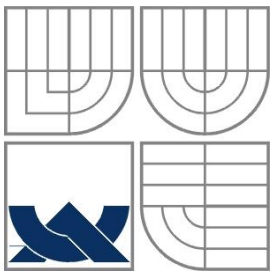
VIZUALIZACE PROGRAMOVÉHO TOKU SPUSTITELNÝCH SOUBORŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

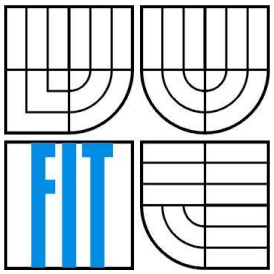
AUTOR PRÁCE
AUTHOR

Bc. JAKUB RUSNÁK

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VIZUALIZACE PROGRAMOVÉHO TOKU SPUSTITELNÝCH SOUBORŮ

VISUALIZATION OF PROGRAM FLOW OF EXECUTABLE FILES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAKUB RUSNÁK

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Dr. Ing. PAVEL ZEMČÍK

BRNO 2011

Zadání diplomové práce/12728/2010/xrusna01

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2010/2011

Zadání diplomové práce

Řešitel: **Rusnák Jakub, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Vizualizace programového toku spustitelných souborů**
Visualization of Program Flow of Executable Files
Kategorie: Počítačová grafika

Pokyny:

1. Nastudujte strukturu spustitelných souborů pro prostředí Microsoft Windows a metody a prostředky 3D vizualizace se zaměřením na IT data.
2. Navrhněte několik variant vizualizace toku programu ve 3D a vyhodnoťe možnosti jejich realizace. Zaměřte se na srozumitelnost, úplnost, snadnost použití a porozumění i namožnosti realizace.
3. Implementujte vybranou metodu vizualizace odle bodu 2 zadání.
4. Demonstrujte funkčnost implementace na vhodném příkladě a otestujte ve spolupráci se vzorkem uživatelů.
5. Diskutujte dosažené výsledky a případné možnosti pokračování práce.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zemčík Pavel, doc. Dr. Ing.**, UPGM FIT VUT
Datum zadání: 20. září 2010
Datum odevzdání: 25. května 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2
L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce se zabývá návrhem a implementací vizualizace programového toku spustitelných souborů na platformě Windows. Na začátku se zabývá strukturou formátu PE EXE a způsobem uložení instrukcí v něm. Následně se práce věnuje současným metodám analýzy a detekce malware, speciálně analýze programového toku. Popisují se též existující metody vizualizace malware a nástroje na implementaci 3D vizualizace dat. Práce se zaměřuje na návrh a implementaci 3D vizualizace programového toku pomocí zobrazení skoků v programu. Výsledkem je nástroj, pomocí kterého je možné na základě vizualizace identifikovat různé druhy malware.

Abstract

This master's thesis describes the visualization of program flow of executable files on Microsoft Windows platform. In theoretical part it describes the PE EXE file format and instruction format. In following chapters there are described current methods of malware analysis, especially the analysis of program flow. Then there are introduced current malware visualization methods and tools for 3D data visualization. The main objective is design and implementation of 3D visualization of jumps in executable files. The result is the visualization tool, which helps to identify different samples of malware from the normal code.

Klíčová slova

PE EXE, instrukce skoku, malware, programový tok, 3D vizualizace

Keywords

PE EXE, jump instruction, malware, program flow, 3D visualization

Citace

Rusnák Jakub: Vizualizace programového toku spustitelných souborů, diplomová práce, Brno, FIT VUT v Brně, 2011

Vizualizace programového toku spustitelných souborů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Doc. Dr. Ing. Pavla Zemčíka. Další informace mi poskytl Ing. Zdeněk Breintenbacher. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Rusnák
23. 5. 2011

Poděkování

Děkuji vedoucímu diplomové práce panu Doc. Dr. Ing. Pavlovi Zemčíkovi za odbornou pomoc a konstruktivní kritiku při zpracovaní této práce. Též děkuji mému konzultantovi z firmy AVG Technologies, panu Ing. Zdeňku Breitenbacherovi za odborné rady při řešení praktické časti.

Obsah

1 Úvod

Pri používaní počítačov sa užívateľ v dnešnej dobe môže okrem užitočných programov, ktoré mu uľahčujú rôzne úlohy, stretnúť aj so softvérom, ktorý sa snaží obísť zabezpečenie počítača a škodiť rôznym spôsobom (kradnutie osobných údajov, mazanie súborov, a pod.). Tento druh softvéru sa nazýva malware (škodlivý kód). Vyvíjajú sa rôzne druhy antivírusového softvéru, ktorý má za úlohu škodlivé programy vyhľadať a odstrániť.

Dnešný moderný malware používa rôzne techniky, ktorými sťažuje ich spoľahlivú analýzu a následnú detekciu. Pri skúmaní týchto techník sa dajú nájsť nové netradičné postupy, ktorými by bolo možné malware odhaliť.

Jeden spôsob, ktorý malware používa na sťaženie ich analýzy je čo najväčšie maskovanie jeho programového kódu. Čím viac sa však o to snaží, tým viac sa jeho kód odlišuje od bežného kódu v aplikáciách. Napríklad používa veľa skokov a rôzne nelogické inštrukcie ako výplň. Normálny program, ktorý je kompilovaný bežným prekladačom a je vysoko optimalizovaný sa tým pádom bude od takéhoto malware líšiť. Fakt, že programový tok malware sa odlišuje od toku bežného programu, sa dá využiť pri detekcii malware. Stačí sa zamerať napríklad čisto na štatistickú analýzu výskytov skokov v kóde programu.

Táto diplomová práca sa zaoberá návrhom a implementáciou aplikácie na 3D vizualizáciu programového toku spustiteľných súborov na platforme Windows. Práca je riešená v spolupráci s firmou AVG Technologies, kde ako jednu z metód na odhalenie malware využívajú analýzu programového toku. Cieľom je navrhnúť rozumnú grafickú reprezentáciu skokov v programoch a implementovať nástroj, ktorý ich zobrazí.

Štruktúra práce pozostáva z kapitol popisujúcich prerekvizity a súčasný stav. Následne je popísaný návrh vizualizácie. Potom sa práca venuje implementácii nástroja na zobrazenie programového toku a jeho testovaniu.

Práca sa na začiatku zaoberá prerekvizitami a popisom súčasného stavu. Táto časť je rozdelená do troch samostatných kapitol, keďže sa v nich popisuje odlišná problematika. Najprv sa práca venuje popisu formátu spustiteľných súborov, ktoré sa používajú v operačnom systéme Miscrosoft Windows a spôsobom uloženia inštrukcií programového kódu. Tieto poznatky sú ďalej dôležité pre štatistickú analýzu skokov v programe. V nasledujúcej kapitole sa práca zaoberá popisom súčasných metód detekcie a analýzy malware, z ktorých sa podrobnejšie venuje analýze programového toku. Najväčšia pozornosť sa venuje štatistickej analýze programového toku pomocou skúmania štruktúry skokov vyskytujúcich sa v programovom kóde, pretože táto metóda tvorí základ pre návrh vizualizácie. Ďalšia kapitola popisuje súčasný stav v oblasti vizualizácie malware a vhodné prostriedky, ktoré by bolo možné na 3D vizualizáciu použiť. Z nich sa potom pri návrhu vyberá.

Nasledujúca kapitola sa venuje krátkemu zhodnoteniu súčasných nástrojov na vizualizáciu malware a popisuje požiadavky na návrh vizualizácie programového toku. Následne sa venuje návrhu dvoch druhov grafov, ktoré zobrazujú skoky v spustiteľných súboroch.

Ďalšia kapitola sa venuje podrobne implementácii vizualizácie. Najprv rozoberá voľbu použitých prostriedkov. Potom sa popisuje princíp fungovania viacerých prvkov použitých vo výslednej vizualizácii a implementáciou samotného nástroja vrátane užívateľského rozhrania.

Záverečná kapitola popisuje na základe viacerých vzoriek spustiteľných súborov použiteľnosť vytvoreného nástroja pri identifikácii rôznych druhov malware. Výsledný nástroj bol testovaný antivírusovými výskumníkmi v spoločnosti AVG. Na konci sú zhrnuté ich skúsenosti pri jeho používaní.

2 Spustiteľné súbory v OS Windows

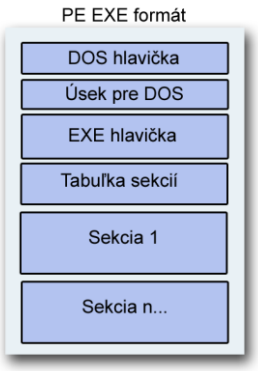
V súčasnej dobe sa na operačnom systéme Microsoft Windows využíva na spustiteľné súbory formát ***Portable Executable*** (ďalej len **PE EXE**). Aby sme sa mohli zaoberať analýzou binárneho kódu malware na tejto platforme, je potrebné si predstaviť tento formát. Táto kapitola sa zaoberá popisom častí PE EXE, ktoré sú potrebné na načítanie súborov tohto typu pre následnú analýzu programového toku. Kvôli nej je potrebné rozobrať si aj spôsob kódovania inštrukcií na platforme 80x86, ktorý je popísaný v závere tejto kapitoly.

Formát PE EXE (podrobnejšie viď [30] a [19]) sa používa sa pre spustiteľné súbory (prípona .exe) a knižnice (prípona .dll) na 32 bitových platformách Windows. 64-bitové platformy používajú formát PE32+, ktorý sa od 32-bitového líši len pridaním niektorých polí alebo ich rozšírením na 64 bitov [20]. Formát bol odvodený pôvodne z unixového formátu COFF. Názov „portable“ znamená, že formát súboru je prenositeľný na všetky platformy Windows, aj keby používali iné CPU ako Intel. Samozrejme môžu byť rozdiely v zakódovaní inštrukcií a podobne, ale štruktúra ostáva rovnaká a tým pádom napríklad systémový PE zavádzač a nástroje na programovanie nemusia byť kompletne prepísané pre každú novú CPU platformu [22].

Najprv je potrebné vysvetliť si adresovanie v EXE súboroch. O PE EXE súboroch platí, že dáta na disku sú veľmi podobné tomu ako vyzerá modul načítaný do pamäte pri spustení. Uľahčuje to prácu systémovému zavádzaču. V súboroch sa často používa relatívna virtuálna adresa (ďalej len RVA), ktorá predstavuje posun (*offset*) od adresy, na ktorú sa modul načíta do pamäte. Používa sa preto, lebo PE zavádzač často nahrá binárny súbor do pamäte na iné miesto, ako je preferovaná adresa, pretože tú často obsadzuje už iný proces. V jednotlivých štruktúrach PE EXE sa často nachádza RAW adresa, čo je obyčajný *offset* od začiatku binárneho súboru na disku.

2.1 Hlavička

PE EXE súbor sa skladá z hlavičky a zo sekcií, ktoré reprezentujú kód alebo určitý druh dát (viď obrázok 2.1). Delenie nie je kvôli logickému usporiadaniu (aj keď býva často pravidlom, že napríklad v sekcii s názvom „.code“ je binárny kód a pod.) ale kvôli vlastnostiam. Každá sekcia ma sadu atribútov, ktoré ju charakterizujú (či sa jedna o kód alebo dáta, či sú dáta čitateľné alebo aj prepisovateľné a pod.). Každá sekcia ma názov, ktorý ju charakterizuje. Tento názov je však iba popisný, pre operačný systém nemá žiaden praktický význam. Názvy sekcií môžu obsahovať teda ľubovoľný reťazec, ktorý závisí od druhu prekladača, prípadne sa dajú aj nastaviť ručne pomocou direktív kompilátoru.



Obrázok 2.1: základné časti PE EXE súboru

PE hlavička nezačína úplne na začiatku súboru. Tam sa nachádza DOS hlavička a takzvaný *DOS stub*, čo je program pre DOS. Je tu kvôli spätnej kompatibilite, v prípade pokusu o spustenie pod operačným systémom DOS vypíše chybovú hlášku. V DOS hlavičke je uložený *offset* na PE hlavičku, ktorý využíva PE zavádzač vo Windows.

PE hlavička obsahuje záhlavie súboru (*file header*, štruktúra *IMAGE_FILE_HEADER*) a nepovinné záhlavie (*optional header*, štruktúra *IMAGE_OPTIONAL_HEADER*). Najdôležitejšie údaje v záhlaví sú:

- *Machine* – označuje procesor, pre ktorý je súbor určený. Pre malware je to dôležitý údaj, ktorý charakterizuje inštrukčnú sadu.
- *NumberOfSections* – obsahuje počet sekcií v PE EXE súbore. Vírusy tento údaj menia napríklad vtedy, ak infekciou modifikujú počet sekcií [30].
- *Charakteristics* – nachádzajú sa tu príznaky súboru. Napríklad je tu uložené, či sa jedná o spustiteľný súbor alebo o dynamický knižnicu.

Štruktúra *IMAGE_OPTIONAL_HEADER* obsahuje ďalšie dôležite informácie o EXE súbore. Spomenieme niekoľko najdôležitejších položiek:

- *Magic* – toto číslo identifikuje, či sa jedná od 32 alebo 64 bitový formát EXE súboru.
- *AddressOfEntryPoint* – obsahuje RVA adresu odkazujúcu na vstupný bod programu (prvú inštrukciu, *entry point*). Táto položka je pre analýzu malware veľmi dôležitá. Dá sa pomocou nej rozpoznať napríklad infekcia počítačovým vírusom, ktorý túto hodnotu môže upraviť tak, aby smerovala na jeho telo [30].
- *ImageBase* – je to preferovaná adresa, na ktorú sa načíta modul do pamäte pri spustení.
- *SectionAlignment* – obsahuje hodnotu, na ktorú sa zarovnávajú sekcie pri načítaní modulu do pamäte. Niektoré vírusy túto hodnotu využívajú pre výpočet umiestnenia svojho tela [30].
- *FileAlignment* – obsahuje hodnotu, ktorá sa využíva na zarovnanie sekcií v PE EXE súbore na disku.
- *SizeOfImage* – je to veľkosť celého modulu nahraného do pamäte, vrátanie hlavičiek.
- *SizeOfHeaders* – určuje veľkosť všetkých hlavičiek a tým pádom a začiatok prvej sekcie.

Ďalšia dôležitá položka voliteľnej hlavičky je pole štruktúr *IMAGE_DATA_DIRECTORY*. Prvky poľa obsahujú dve položky. Prvá je RVA adresa na danú štruktúru, druhá je jej veľkosť v bajtoch. Každý prvok poľa odkazuje na nejakú dôležitú štruktúru v EXE súbore, ako napríklad tabuľka importovaných alebo exportovaných funkcií, ladiace informácie, zdroje (*resources*), tabuľka relokácií a podobne. Ich umiestnenie čo sa týka sekcií je rôzne. Veľkosť poľa nie je pevne daná, dá sa zistiť pomocou položky *NumberOfRvaAndSizes* vo voliteľnej hlavičke PE EXE súboru.

2.2 Tabuľka sekcií

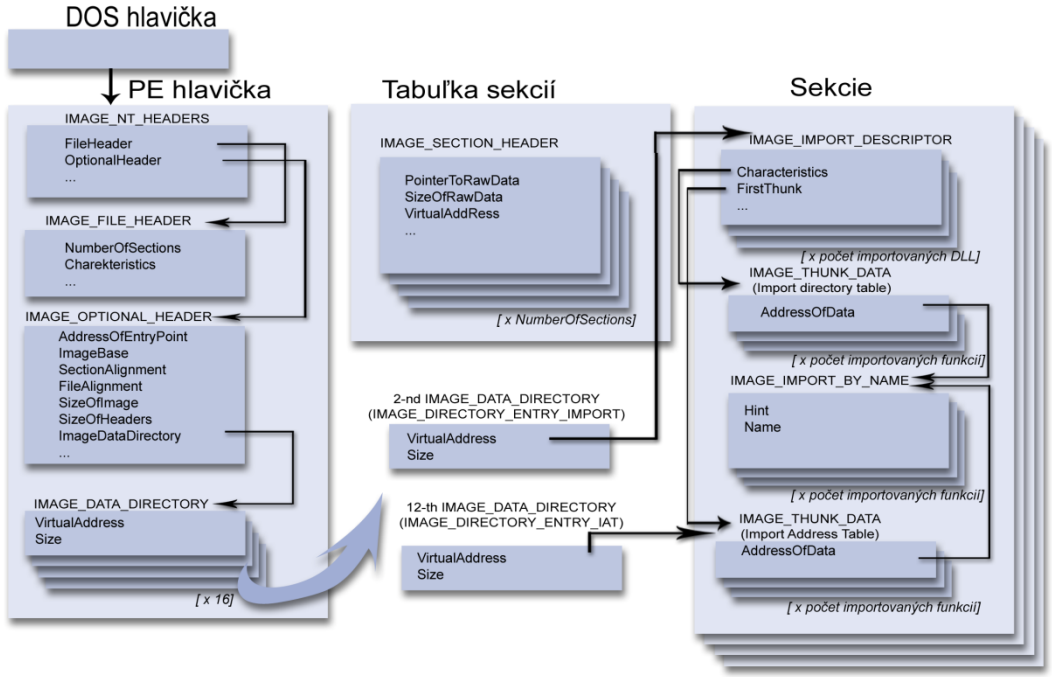
Medzi PE hlavičkou a dátami prvej sekcie leží tabuľka sekcií, čo je vlastne pole štruktúr *IMAGE_SECTION_HEADER*. Veľkosť poľa je daná počtom sekcií, ktorý je určený už spomínanou položkou *NumberOfSections* v hlavičke. Tabuľka sekcií je pre analýzu malware dôležitá, pretože napríklad vírusy ju využívajú pri infekcii buď na zmenu existujúcej alebo vytvorenie novej sekcie. Najvýznamnejšie položky sú:

- *Name* – názov sekcie.
- *SizeOfRawData* – veľkosť sekcie v súbore na disku.
- *PointerToRawData* – RAW adresa začiatku sekcie.
- *VirtualSize* – veľkosť sekcie keď je súbor nahraný do pamäte.
- *VirtualAddress* – RVA adresa začiatku sekcie. Pomocou tohto údaju a RAW adresy v *pointerToRawData* môžeme teda lokalizovať ľubovoľnú inú štruktúru odkazujúcu sa pomocou RVA adresy do niektorej sekcie (dá sa využiť napríklad pri lokalizácii štruktúr odkazovaných z *IMAGE_DATA_DIRECTORY*, alebo pri analýze programového toku, keď je potrebné na základe RVA adresy vypočítať polohu v EXE súbore – RAW adresu).
- *Characteristics* – obsahuje bitové príznaky charakterizujúce sekciu. Napríklad či obsahuje binárny kód, či je zapisovateľná alebo len na čítanie, aké je zarovnanie dát v sekcii a pod.

2.3 Tabuľka importov

Keď chce EXE súbor využívať funkcie z iných DLL knižníc, je potrebné ich importovať. Na to slúži tabuľka importov, ktorá obsahuje názvy funkcií a knižníc. Pri spúšťaní programu PE zavádzač hľadá tieto knižnice a zisťuje konkrétne adresy importovaných funkcií. Preto je z pohľadu analýzy binárneho kódu je tabuľka importov dôležitá štruktúra. Vírusy túto adresy funkcií z tejto tabuľky často používajú na volanie funkcií rôznych API [30].

Informácie o importovaných funkciách začínajú tabuľkou *Import directory table*. Je to v podstate pole štruktúr typu *IMAGE_IMPORT_DESCRIPTOR*. Odkazuje naň druhá položka poľa štruktúr *IMAGE_DATA_DIRECTORY* s názvom *IMAGE_DIRECTORY_ENTRY_IMPORT*. Každý prvok poľa obsahuje informácie potrebné pre importovanie z konkrétnej DLL knižnice. Je to napríklad názov knižnice. Posledný prvok indikujúci koniec poľa je prázdny (vyplnený nulami).



Obrázok 2.2: štruktúra PE EXE súboru (zjednodušene)

Prvky tabuľky *Import directory table* obsahujú odkaz na dve rôzne polia štruktúr typu *IMAGE_THUNK_DATA*. Tieto obsahujú v EXE súbore na disku buď ordinálnu hodnotu, ktorá číselne označuje umiestnenie importovanej funkcie alebo RVA adresu na reťazec s názvom funkcie. Ten je uložený v poli štruktúr *IMAGE_IMPORT_BY_NAME* v položke *Name*.

Prvé pole typu *IMAGE_THUNK_DATA* je takzvaná *Import Address Table* (skrátene IAT). Pri spustení EXE súboru nahradí PE zavádzač v tejto tabuľke odkazy na názvy funkcií za skutočné adresy z príslušnej DLL knižnice. Na tabuľku odkazuje aj dvanásta položka poľa štruktúr *IMAGE_DATA_DIRECTORY* s názvom *IMAGE_DIRECTORY_ENTRY_IAT*.

V niektorých prípadoch je potrebné zachovať pôvodný obsah tabuľky *Import directory table* [21]. Preto sa v EXE súbore nachádza ešte druhé duplicitné pole typu *IMAGE_THUNK_DATA*, ktoré PE zavádzač pri nahrávaní EXE súboru do pamäte nemení.

Pre lepší prehľad v jednotlivých štruktúrach je priložený obrázku 2.2. Sú v ňom znázornené len niektoré popisované prvky (kvôli prehľadnosti).

2.4 Ostatné štruktúry

V dynamických knižniciach sa používa tabuľka exportov. Slúži na exportovanie funkcií v DLL knižniciach. Z nich môžu potom ostatné programy tieto funkcie importovať a následne volať (popísané v predošlej kapitole). Podobne ako u tabuľky importov, môže byť funkcia exportovaná podľa mena alebo podľa umiestnenia. Hlavná štruktúra tabuľky exportov je *IMAGE_EXPORT_DIRECTORY*. Jej najdôležitejšie položky sú tri polia: tabuľka adries funkcií, tabuľka mien funkcií a tabuľka ordinálnych hodnôt funkcií [30]. Na túto tabuľku exportov odkazuje štruktúra *IMAGE_DIRECTORY_ENTRY_EXPORT*, ktorá je prvá položka v poli štruktúr *IMAGE_DATA_DIRECTORY*.

Keď sa program nahráva do pamäte, zavádzač sa pokúsi umiestniť ho na adresu *ImageBase* uloženú v hlavičke *IMAGE_FILE_HEADER*. Ak je daná adresa obsadená, zavádzač nahrá program na inú adresu. V tom prípade je však potrebné upraviť všetky absolútne adresy v binárnom kóde programu. Na to slúži tabuľka relokácií na ktorú odkazuje šiesta položka poľa štruktúr *IMAGE_DATA_DIRECTORY* s názvom *IMAGE_DIRECTORY_ENTRY_BASERELOC*. Tabuľka obsahuje adresy, na ktorých sú hodnoty ktoré treba modifikovať. Zavádzač k nim pripočíta rozdiel medzi pôvodnou preferovanou adresou uloženou v *ImageBase* a aktuálnou adresou na ktorú sa program nahral. Našťastie sa absolútne adresovanie nepoužíva vo veľkom, väčšina inštrukcií skoku a volania funkcií využíva relatívne adresy (tj. *offset* od danej adresy, na ktorej sa inštrukcia nachádza). Absolútne adresovanie sa využíva väčšinou u inštrukcií, ktoré využívajú absolútnu adresu na prístup k nejakým dátam [22].

Programy často obsahujú rôzne ikony, textové reťazce a podobne. Jedna z možností, ako tieto dáta uložiť sú zdroje (*resources*). Tie sa často nachádzajú v samostatnej sekcii s názvom *rsrc*. Zdroje sú uložené v stromovej hierarchii, podobne ako súborový systém. Každý adresár je štruktúra typu *IMAGE_RESOURCE_DIRECTORY*. Z pohľadu analýzy binárneho kódu nie je táto štruktúra až taká dôležitá, podrobnejší popis viď [21].

Spustiteľné súbory z prostredia .NET sú tiež v PE EXE formáte. Avšak samotný kód a dáta týchto aplikácií sú minimálne. Väčšinou obsahujú len krátky úsek kódu zabezpečujúci získanie metadát potrebných pre .NET prostredie a nahranie IL inštrukcií do pamäte [20]. Informácie potrebné pre .NET prostredie sú uložené v štruktúre *IMAGE_COR20_HEADER*, ktorá je odkazovaná štrnástou položkou z poľa *IMAGE_DATA_DIRECTORY* (je to tzv. *COM Runtime descriptor*).

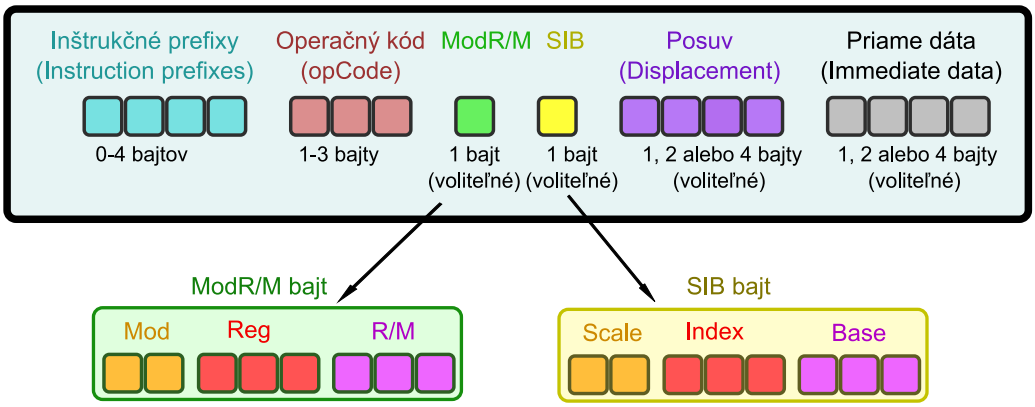
2.5 Kódovanie inštrukcií 80x86

Aby mohol program bežať na procesore, musí byť uložený v binárnej podobe, ktorej procesor rozumie. Ako bolo v predošlých kapitolách spomenuté, pre PE EXE súbory platí, že to, čo je na disku je do veľkej miery zhodné s tým, čo bude v operačnej pamäti, keď sa program spustí. Platí to aj pre binárny kód programu. Aby sme ho teda mohli analyzovať, potrebujeme okrem štruktúry EXE súboru poznať aj kódovanie inštrukcií na procesoroch rodiny 80x86. Táto kapitola je len krátky úvod do tejto problematiky postačujúci na to, aby sme z binárneho kódu mohli dekódovať inštrukcie skokov. Podrobnejší popis viď [13] a [12].

Inštrukcie pozostávajú zo šiestich hlavných častí (viď obrázok 2.3):

- inštrukčné prefixy (*Instruction prefixes*)
- operačný kód (*opcode*)
- ModR/M bajt
- *SIB* bajt
- posuv (displacement)
- priame dáta (*Immediate data*)

Dĺžka inštrukcie je variabilná. Maximálne to môže byť teoreticky 16 bajtov, prakticky procesor nepovolí inštrukcie dlhšie, ako 15 bajtov [12]. Jediná povinná položka je operačný kód, výskyt ostatných častí závisí od inštrukcie, počtu a typu jej parametrov.



Obrázok 2.3: kódovanie inštrukcie

Inštrukčné prefixy sú rozdelené do 4 skupín. V inštrukcii sa nachádza vždy iba jeden prefix z danej skupiny. Ich poradie môže byť ľubovoľné. Ich veľkosť môže byť 0 (žiaden prefix) až 4 bajty. Skupiny prefixov sú tieto:

1. Prefixy ovplyvňujúce cykly u inštrukcií narábajúcich s reťazcami (napríklad MOVS). Ďalej k tejto skupine patrí tzv. *lock prefix*, ktorý zaisťuje prístup k zdieľanej pamäti u systémov s viacerými procesormi.
2. Prefixy, ktoré menia definovaný segmentový register a takzvané *branch hints*, ktoré slúžia ako rada pre procesor, kam sa pravdepodobne bude uberať ďalší tok programu po inštrukcii podmieneného skoku (využitie v prípadoch, kedy je jedna možnosť vysoko pravdepodobná).

3. Prefixy, ktoré špecifikujú veľkosť operandov.
4. Prefixy špecifikujúce veľkosť adresy.

Operačný kód špecifikuje, aká inštrukcia sa bude vykonávať. Môže mať 1 až 3 bajty. Na rozšírenie množstva možných inštrukcií sa využíva takzvaný *opcode expansion prefix* (na 80x86 platforme sa rovná hodnote 0x0F). Je to mechanizmus, kedy napríklad najčastejšie používané inštrukcie majú jeden bajt (kvôli rýchlemu dekódovaniu a úspore pamäti). Ak je nutné zakódovať väčšie množstvo inštrukcií, rozšíri sa operačný kód na dva bajty. Je potrebné ho však v prvom bajte nejako odlíšiť od ostatných jednobajtových. Na do sa používa spomínaný prefix. Druhý bajt potom obsahuje samotný operačný kód.

Operačný kód mnohých inštrukcií obsahuje okrem samotného kódu inštrukcie aj bity špecifikujúce rôzne parametre. Napríklad inštrukcia ADD má v nultom bite uložené, akú veľkosť majú operandy. Prvý bit špecifikuje, ktorý z operandov je zdroj a ktorý destinácia [12]. Tieto prípady tým pádom šetria pamäť tým, že sa nemusia dané informácie uvádzať v inštrukčných prefixoch spomínaných vyššie.

ModR/M bajt špecifikuje použitý adresovací mód a operandy inštrukcie. Pozostáva z troch častí:

- *Mod* (2 bity) – v kombinácii s *R/M* časťou špecifikuje adresovací mód
- *Reg* (3 bity) – špecifikuje použitý register v operandoch inštrukcie
- *R/M* (3 bity) – špecifikje buď použitý register v operandoch inštrukcie alebo v kombinácii s *Mod* časťou adresovací mód

V niektorých prípadoch je okrem ModR/M bajtu na charakterizovanie adresovania potrebný ešte SIB bajt (názov podľa angl. *Scale-Index-Base*). Ten ukladá informácie potrebné pre typ adresovania využívajúci mierku (tzv. *scaled indexing*), ktoré má tvar: *bázová adresa + index * mierka*. Bajt pozostáva z týchto položiek:

- *Scale* (2 bity) – mierka (môže byť 1, 2, 4, alebo 8)
- *Index* (3 bity) – špecifikuje indexový register
- *Base* (3 bity) – špecifikuje bázový register

Posuv (*displacement*) sa využíva v niektorých adresovacích módoch. Jedná sa o hodnotu, ktorá určuje veľkosť posuvu. Napríklad pri inštrukcii *MOV AX, [BP + 12]* bude v tejto položke hodnota 12. Posuv môže zaberať 1, 2 alebo 4 bajty.

Priame dáta (*immediate data*) sú konštanty vyskytujúce sa v inštrukciách. Napríklad pri inštrukcii *MOV AX, 20* bude hodnota 20 uložená v tejto časti. Veľkosť dát môže byť 1, 2 alebo 4 bajty.

Na záver si uvedieme príklad ako vyzerá kódovanie konkrétnej inštrukcie. Bude to inštrukcia blízkeho skoku (*relative near jump*) o daný posuv veľkosti 32 bitov. Jedná sa o jednoduchý prípad, nenachádzajú sa tu žiadne prefixy, ani *ModR/M* bajt. Binárna podoba inštrukcie vyzerá nasledovne (hexadecimálne):

```
E9 FB FE FF FF
```

Operačný kód inštrukcie je 0xE9 (veľkosť 1 bajt), to znamená, že sa jedná o nepodmienený blízky skok (*relative near jump*) s veľkosťou posuvu 4 bajty [13]. Kódovanie posuvu je typu little endian, teda ako prvé sú uložené menej významné bajty. Hodnota posuvu je teda 0xFFFFFEE9. Jedná sa o záporné číslo, dekadicky -261.

3 Analýza a detekcia malware

Škodlivý kód (malware) je druh softvéru, ktorý obsahuje inštrukcie snažiace sa prinútiť počítačový systém aby vykonával činnosť ktorú zamýšľa útočník [28]. Inštrukcie nemusia byť iba vo forme binárneho spustiteľného súboru, ale aj vo forme skriptovacieho jazyka (napríklad makro vírusy). Škodlivá činnosť môže byť rôzneho druhu, napríklad mazanie dôležitých súborov, monitorovanie kláves, umožnenie iným útočníkom prístup a ovládanie počítača a podobne.

Táto kapitola predstavuje základné známe metódy na detekciu škodlivého kódu. Na základe týchto metód rozoberá spôsoby, akými sa malware bráni voči odhaleniu. Keďže sa tieto metódy stávajú stále sofistikovanejšími, musia antivírusoví experti vynachádzať nové postupy ako malware odhaliť. Jedna z nich, analýza programového toku ktorú používajú v spoločnosti AVG, je popísaná na konci tejto kapitoly. Táto metóda slúži ako základ pre vizualizáciu, ktorou sa zaoberá táto práca.

3.1 Druhy malware

Základné charakteristiky podľa ktorých sa delí škodlivý softvér (viď [2]) sú napríklad, či sa malware pri snahe šíriť ďalej sám replikuje (vytvára kópie samého seba). Ďalej to môže byť snaha rozmnožiť sa čo najviac. Jeden zo znakov je parazitické správanie sa malware, ktorý potrebuje na vlastný existenciu iný spustiteľný kód (typický príklad sú počítačové vírusy).

Malware môžeme rozdeliť do viacerých skupín. Delenie nie je úplne presné, často má malware znaky patriace viacerým typom (napríklad vírus môže obsahovať zadné vrátka umožňujúce prístup útočníkovi na počítač). Najznámejšie typy sú (podľa [2] a [25]):

- Vírus – jedná sa o parazitický replikujúci sa malware, ktorý sa ktorý dokáže infikovať ďalšie programy a ich modifikáciou je schopný zaistiť, aby obsahovali potenciálne sa vyvíjajúcu sa kópiu jeho samotného [30]. Vírus môže obsahovať ešte okrem kódu, ktorý zabezpečuje jeho replikáciu aj tzv. *payload*. Je to rutina, ktorá vykonáva ďalšiu škodlivú činnosť, ako napríklad mazanie súborov a pod.
- Červ – malware, ktorý sa tak isto ako vírusy replikuje, nepotrebuje však na to iné programy, dokáže existovať sám. Snaží sa čo najviac rozšíriť pomocou počítačovej siete.
- Trójsky kôň – malware, ktorý svoje škodlivé účinky skrýva za to, že je integrovaný v nejakom navonok užitočnom softvéri. Nevie sa sám replikovať, ako vírusy alebo červy.
- Spyware – program, ktorý zbiera citlivé informácie o užívateľovi a posiela ich ďalej útočníkovi. Nevie sa sám replikovať a nemá parazitický charakter.
- Adware – program zobrazujúci nežiadaný reklamu. Môže odosielať informácie o užívateľovi (podobne ako spyware). Nevie sa sám replikovať a nemá parazitický charakter.
- Zadné vrátka (backdoor) – program umožňujúci prístup neznámemu útočníkovi na počítač cez sieť.

3.2 Metódy detekcie malware

Skenovanie

Najzákladnejšia metóda detekcie malware je skenovanie reťazcov. Je založená na tom, že poznáme určitú sekvenciu bajtov, ktorá sa vyskytuje vo všetkých vzorkách daného typu vírusu. Táto sa potom pri skenovaní na výskyt malware vyhľadáva v binárnom súbore.

Táto metóda má mnoho variant a rozšírení. Napríklad je možné používať zástupné znaky (*wildcards*) ktoré špecifikujú, že na danej pozícii sa môže vyskytovať ľubovoľný znak. Alebo je možné k definícii hľadaného reťazca pridať číselnú hodnotu špecifikujúcu koľko bajtov sa nemusí pri vyhľadávaní na rôznych miestach zhodovať. Pri vyhľadávanie reťazcov sa využíva množstvo algoritmov na jeho zrýchlenie. Napríklad sa vyhľadáva iba v tých častiach binárneho súboru, ktoré obsahujú binárny kód. Vynechávajú sa začiatok a koniec súborov, pretože tie obsahujú napríklad hlavičky, ktorých štruktúra a obsah sa často opakuje. Ďalšia možnosť ako zefektívniť vyhľadávanie je používanie kontrolných súčtov alebo hešovacích funkcií [30]. Týmito metódami sa najmä zmenší veľkosť databáze obsahujúcej detekované vzorky.

Výhoda metód skenovania reťazcov je najmä ich rýchlosť a schopnosť precízne identifikovať známe vírusy. Ich nevýhoda je neschopnosť detekovať neznámy malware a závislosť na aktuálnosti vírusovej databáze.

Detekcia pomocou emulátora

Ďalšia často používaná metóda je detekcia pomocou virtuálneho stroja (emulátora). Ten simuluje beh kódu malware a analyzuje ich aktivitu. Táto metóda je veľmi užitočná napríklad pri detekcii polymorfných vírusov, kedy sa pri emulácii počas jeho dekryptovania používa jeho vlastný kód.

Emulátor pozostáva z piatich základných častí (podľa [2]):

- Emulácia procesora
- Emulácia pamäte
- Emulácia služieb operačného systému. Keďže by bolo náročné spúšťať v emulátore celý operačný systém, emulátor pri volaní konkrétnych funkcií podstrčí emulovanému kódu falošný výsledok.
- Kontrolér - dôležitá súčasť virtuálneho stroja, ktorá má na starosti ukončenie emulácie. Emulácia sa ukončí po určitom počte inštrukcií, alebo na základe analýzy behu programu (napríklad na základe porovnania počtu inštrukcií pristupujúcich do pamäte pri hľadaní dekryptovaného tela polymorfných vírusov).
- Analyzátor behu programu, ktorý si ukladá výsledky emulácie. Napríklad pri detekcii polymorfných vírusov sa ukladá obsah pamäte, aby sa mohlo analyzovať jeho dekryptované telo.

Najväčšia výhoda emulátorov je to, že umožňujú spustenie vírusu v zabezpečenom prostredí a tým pádom sú schopné odhaliť aj nové druhy malware. Nevýhoda virtuálnych strojov je, že sú pomalé v porovnaní s ozajstným procesorom a nie sú teda vhodné na rýchlu detekciu. Tiež implementovať emulátor obsahujúci napríklad kompletnú inštrukčnú sadu daného procesora je náročná úloha.

Statická heuristika

Na rozdiel od detekcie pomocou emulátora (tiež sa nazýva dynamická heuristika, [2]), statická heuristika skúma znaky charakteristické pre rôzne druhy malware nachádzajúce sa ešte v súbore pred jeho spustením. Pri analýze sa napríklad u vírusov zameriava na skúmanie týchto znakov:

- Kontrola vstupného bodu programu (*entry point*). Napríklad ak ukazuje na poslednú sekciu, ktorá väčšinou neobsahuje kód, alebo ak ukazuje do priestoru medzi sekciami, tak je veľká pravdepodobnosť, že súbor bol modifikovaný vírusom.
- Podozrivé názvy sekcií – najpoužívanejšie kompilátory používajú štandardné názvy (ako napríklad *code, text* alebo *data*).
- Podozrivé príznaky sekcií. Napríklad väčšinou je sekcia obsahujúca kód označená ako spustiteľná a iba na čítanie, pretože neobsahuje žiadne dáta. Infikovaná sekcia však môže byť označená ako zapisovateľná, čo je podozrivé.
- Podozrivé záznamy v tabuľke importov. Napríklad importovanie pomocou ordinálnej hodnoty bežné kompilátory väčšinou nepoužívajú [30].

Tieto znaky však samostatne nestačia na spoľahlivú detekciu. Ich spojením sa po ich analýze sa vyhodnocuje, s akou pravdepodobnosťou sa jedná o škodlivý kód.

Výhoda statickej heuristiky je detekcia ja doteraz neznámeho škodlivého kódu. Nevýhoda sú falošné alarmy, pretože aj niektoré legitímne aplikácie môžu mať znaky malware.

Heuristika pomocou štatistických metód

Pri analýze škodlivého kódu sa využívajú aj štatistické metódy, ktoré sú tiež určitým druhom statickej heuristiky (pretože skúmame znaky v spustiteľného súboru v podobe a akej je na disku). Napríklad sa skúma entropia (miera neusporiadanosti dát) v rôznych miestach súboru. Pri tejto analýze malware sa počíta takzvaná mapa entropie. Tá priraďuje každým 16 bajtom spustiteľného súboru konkrétnu hodnotu entropie od 0 do 15. Toto pole hodnôt teda určuje, akú neusporiadanosť má každý 16-bajtový úsek súboru.

Mapa entropie sa využíva na nájdenie miest v súbore, ktoré majú vysokú neusporiadanosť. To sú väčšinou miesta obsahujúce binárny kód alebo miesta, ktoré obsahujú šifrované alebo komprimované dáta (napríklad polymorfné vírusy používajú šifrovacie techniky na skrytie svojho tela). Pri skúmaní vírusov pomocou mapy entropie sa môžeme teda zamerať len na tieto časti súboru, ktoré sú z hľadiska ich analýzy zaujímavé.

Pomocou mapy entropie môžeme tiež vyhľadávať a identifikovať rôzne vzorky polymorfných vírusov. Tie sú síce v binárnej podobe odlišné, ale keď vypočítame ich mapu entropie, tak sa u jednotlivých vzorkov v mnohých miestach zhoduje. Na základe nej je možné potom niektoré typy vírusov identifikovať [24].

3.3 Metódy obrany malware voči odhaleniu

Skenovanie je jedna zo základných metód detekcie malware. Časom vyvinuli autori malware viaceré techniky obrany voči skenovaniu. Prvý spôsob ktorý sa objavil boli zakódované vírusy [30]. Tie obsahujú krátky dekryptor a ich samotné telo je zakódované. Ďalší stupeň vo vývoji boli oligomorfné vírusy, ktoré obsahovali viacero dekryptorov, ktoré sa náhodne striedali. Ďalší vývojový stupeň predstavujú polymorfné vírusy. Tie vedia meniť ich dekódovaciu rutinu do veľmi vysokého počtu inštancií, ktoré môžu mať až milióny rôznych podôb. Aby sa tejto rozmanitosti docielilo, vkladajú sa

do binárneho kódu napríklad náhodne nič nerobiace inštrukcie, ktoré neovplyvňujú výsledok výpočtu alebo dátové oblasti, ktoré sa mixujú s binárnym kódom. Aj tieto druhy vírusov bolo možné však napríklad pomocou emulátoru odhaliť, keďže ich telo je vždy konštantné. Preto vznikli metamorfné vírusy, ktoré nepotrebujú dekryptor, pretože celé ich telo je polymorfné (odlišné kus do kusu). Napríklad vírus *W95/Zperm* používa náhodné vkladanie inštrukcií skoku na zmenu jeho celého tela [30]. Tieto techniky známe z vírusov používajú aj iné druhy malware, napríklad trójske kone (viď napríklad ukážky vzoriek trójskeho koňa *Backdoor.Cycbot* v kapitole 7.1).

Niektorý malware používa techniky na obranu voči emulácii. Napríklad niektoré vírusy používajú rôzne nedokumentované inštrukcie procesoru, ktoré normálne na hardvéri fungujú. Alebo sa snažia využívať pokročilé funkcie procesoru, ako napríklad MMX [30].

Malware využíva mnohé techniky, ktoré sťažujú statickú heuristiku. Jedna z nich je napríklad u vírusov metóda utajenia vstupného miesta kódu (*entry point obscuring*, ďalej len skrátene EPO), kedy vírus nemení údaje o *entry point* v hlavičke, ale modifikuje programový kód na ktorý *entry point* ukazuje. Niektoré vírusy upravia programový kód tak, že prevezmú riadenie v náhodný moment [30]. Toho sa dá docieliť viacerými spôsobmi. Napríklad vírus skenuje inštrukcie v binárnom kóde programu a hľadá inštrukcie skoku alebo volania funkcií (*call* a *jump*). Tie upraví tak, že odkazujú na jeho telo, ktoré prevezme riadenie a po dokončení sa zas vráti do pôvodného programu. Iná možnosť je prepísať záznamy pre niektorú funkciu v tabuľkách importov, kde vloží odkaz na svoje telo.

Pri tvorbe algoritmov na generovanie polymorfných druhov malware alebo kvôli sťaženiu práce antivírových výskumníkov sa používajú rôzne metódy zatemňovania binárneho kódu (angl. *obfuscation*). Proti týmto technikám sa využíva analýza programového toku (viď nasledujúca kapitola).

3.4 Analýza programového toku malware

Jedna z možností ako sa vysporiadať s pokročilými obrannými technikami malware je použitie analýzy programového toku. V tejto kapitole sa spomína hlavne statická heuristika, kde sa skúmajú inštrukcie v binárnom kóde PE EXE súboru.

Jeden variant skúmania programového toku je graf toku riadenia (angl. *Control flow graph*). Tento graf sa vytvára z inštrukcií nachádzajúcich sa v binárnom kóde pomocou disassembleru. Reprezentuje všetky rôzne cesty, ktorými by sa beh programu mohol uberať. Uzly grafu korešpondujú so základnými blokmi programu. Blok je sekvencia inštrukcií, ktorá končí ľubovoľnou inštrukciou ovplyvňujúcou beh programu (skok, volanie funkcie a pod.). Hrany grafu reprezentujú rôzne varianty toku programu medzi blokmi (uzlami grafu) [25].

Graf toku riadenia sa dá využiť na detekciu rôznych polymorfných druhov malware, ktorú prezentujú napríklad v [3]. Pri detekcii sa najprv vytvorí graf toku riadenia daného skenovaného programu. Potom sa hľadajú v tomto grafe rôzne podgrafy toku riadenia typické pre známe druhy malware. Aby sa obišli rôzne techniky zatemnenia kódu používané autormi malware, graf sa redukuje. Napríklad podgraf, v ktorom sa beh programu nevetví, sa zlúči do jedného uzlu.

Výhoda metódy detekcie malware pomocou grafu toku riadenia je dobrá detekcia rôznych polymorfných druhov malware. Nevýhoda je vo výpočtovej náročnosti tejto metódy [2].

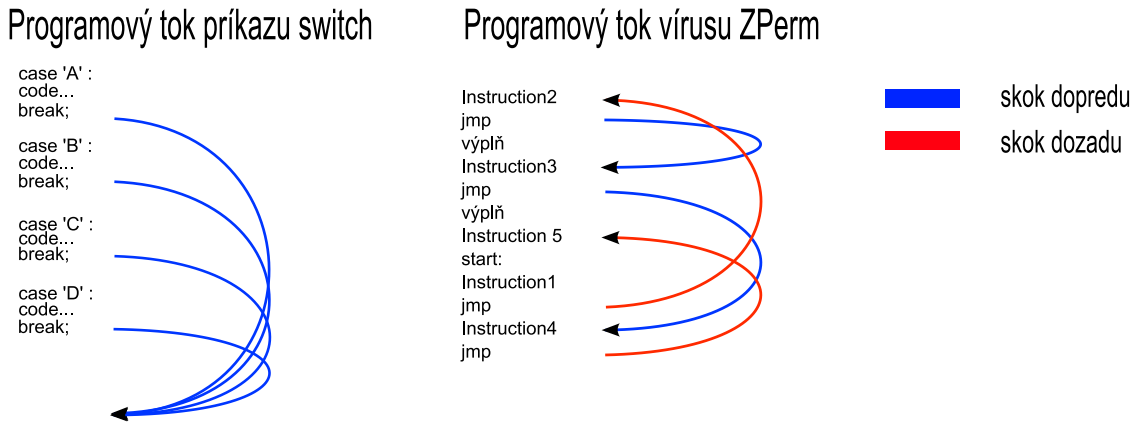
Iná možnosť je neskúmať priamo tok riadenia programu ale zamerať sa na jeho štatistické vlastnosti. Túto metódu používajú v spoločnosti AVG ako pomôcku pri analýze nových druhov malware (viď nasledujúca kapitola).

Štatistická analýza programového toku

Štatistická analýza programového toku skúma inštrukcie meniace programový tok, najmä inštrukcie skokov. Vychádza sa z toho, že binárny kód malware je za použitia rôznych metód zatemňovania programového kódu (*obfuscation*) často plný náhodných inštrukcií skokov v oboch smeroch. Optimalizovaný binárny kód pochádzajúci z bežných kompilátorov naproti tomu obsahuje skoky, ktoré majú oveľa pravidelnejšiu štruktúru. Na základe týchto znakov je možné odlíšiť malware od normálnych aplikácií.

Skok pozostáva z adresy, kde začína (adresa inštrukcie) a adresy kde končí (adresa kam ukazuje operand inštrukcie). Štatistická analýza programového toku sa zameriava sa len na určité druhy skokov. Vzhľadom na vysoký počet rôznych inštrukcií meniacich beh programu sa vyberie vždy len jeden typ skoku, ktorý je pre daný druh malware typický. Často sa skúma najmä inštrukcia nepodmieneného relatívneho blízkeho skoku (*relative near jump*, operačný kód *0xE9* [13]), ktorá má ako operand 32 bitový *offset*. Adresu, kde skok smeruje získame pripočítaním ofsetu a adresy hneď za danou inštrukciou. Offset môže byť aj záporné číslo. Na základe znamienka hodnoty *offsetu* sa delia skoky na skoky smerujúce dopredu (kladný *offset*) a skoky smerujúce dozadu (záporný *offset*). Inštrukcia *relative near jump* sa vyskytuje v binárnom kóde PE EXE súborov často, pretože jej použitie má tú výhodu, že cieľová adresa nie je špecifikovaná absolútnou hodnotou, ale *offsetom*. Preto nie je potrebné používať tabuľku relokácií na vypočítanie správnej cieľovej adresy skoku v programe zavedenom v pamäti.

Pri analýze programového toku sa skúma štruktúra skokov. Ako príklad pravidelnej štruktúry sa dajú uviesť inštrukcie skoku, ktoré vygeneruje kompilátor jazyka C pri príkaze *switch*. Každý blok ukončený kľúčovým slovom *break* predstavujúci jednu podmienku je zakončený inštrukciou skoku, ktorá smeruje na kód nasledujúci za príkazom *switch*. Skoky sú vždy jedného smeru. Ako príklad nepravidelnej štruktúry možno uviesť metamorfný vírus *W95/Zperm*. Ten vytvára svoje mutácie náhodným pridávaním inštrukcií skoku oboch smerov a rôzneho „smetia“ do svojho kódu [30]. Na obrázku 3.1 sú pre názornosť uvedené príklady normálneho programového toku a programového toku malware.



Obrázok 3.1: porovnanie štruktúry programového toku

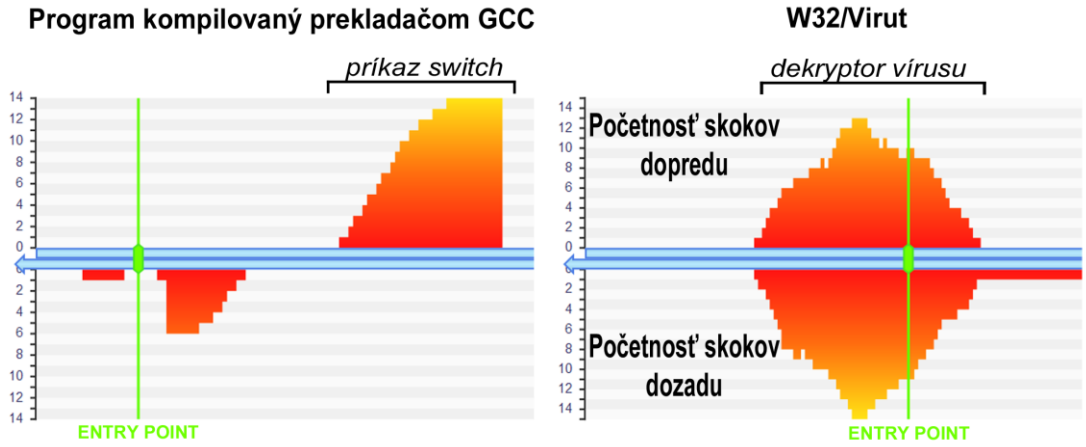
Ďalší znak, ktorý je objektom skúmania je početnosť (hustota) skokov na určitej adrese v binárnom kóde sekcie PE EXE súboru. Hodnota početnosti na určitej adrese určuje počet skokov, ktoré danou adresou prechádzajú (vrátane tých, ktoré na nej začínajú alebo končia). Početnosť skokov sa počíta pre každú adresu v určitom rozsahu (väčšinou adresy prislúchajúce sekcii obsahujúcej binárny kód).

Ako príklad, kde skúmanie početnosti skokov pomáha pri analýze malware si môžeme uviesť polymorfný vírus *W32/Virut*. (iný názov *W32/Virtu*, podrobný popis viď [1]). Pre sťaženie analýzy využíva polymorfné techniky, a to vkladanie nadbytočných inštrukcií a cyklov (aj kvôli spomaleniu prípadnej emulácie). Jeho obfuskovaný kód dekryptoru obsahuje veľa inštrukcií skokov v oboch smeroch. Náhodné rozmiestnenie skokov spôsobuje, že ich početnosť je v tele vírusu v oboch smeroch približne rovnaká, pravidelne stúpa a v strede je najvyššia (akoby bola určená Gaussovou funkciou, viď obrázok 3.2 vpravo). U normálneho programového kódu sa početnosť skokov dopredu aj dozadu väčšinou výrazne odlišuje a jej hodnota sa na viacerých miestach prudko mení (pretože často smeruje väčší počet skokov na jednu konkrétnu adresu, čo spôsobí veľkú zmenu hodnoty ich početnosti, viď obrázok 3.2 vľavo). Ako príklad sa dá opäť uviesť binárny kód generovaný príkazom *switch*, kde početnosť pravidelne stúpa smerom ku koncu bloku. Tam prudko klesá, keďže všetky skoky smerujú na jeho koniec. Keďže sa tu nachádzajú skoky len v jednom smere, mení sa len početnosť skokov jedného druhu.

Na obrázku 3.2 je znázornené porovnanie početnosti skokov normálneho programu a tela dekryptoru polymorfného vírusu *W32/Virut*. Vodorovná os predstavuje RAW adresy v PE EXE súbore, zvislá os početnosť skokov na danej adrese. Graf znázorňuje početnosť skokov (čím vyššia hodnota, tým viac skokov prebieha ponad danú adresu). Vrchný pás zobrazuje skoky smerujúce dopredu, spodný skoky smerujúce dozadu. Zelená značka zobrazuje vstupný bod (*entry point*).

Ukážkový program implementovaný v jazyku C bol kompilovaný prekladačom GCC. Jeho hlavná funkcia bola demonštrovať rozsiahly príkaz *switch*, ktorý obsahuje 15 podmienok. Na obrázku je vidno, že oblasť, kde sa nachádza jemu prislúchajúci kód obsahuje skoky len smerom dopredu. Početnosť postupne narastá až po adresu, na ktorú všetky skoky jednotlivých vetiev smerujú, a tam klesá prudko na nulu.

V prípade vírusu *W32/Virut* sa dá jeho dekryptovacia rutina rozoznať podľa vysokej početnosti skokov v blízkosti *entry point*. To, že sa v kóde nachádza približne rovnaký počet skokov smerujúcich dopredu aj dozadu a fakt, že početnosť stúpa v strednej časti dekryptoru sú podozrivé znaky malware.



Obrázok 3.2: Porovnanie početnosti skokov normálneho programu a polymorfného vírusu

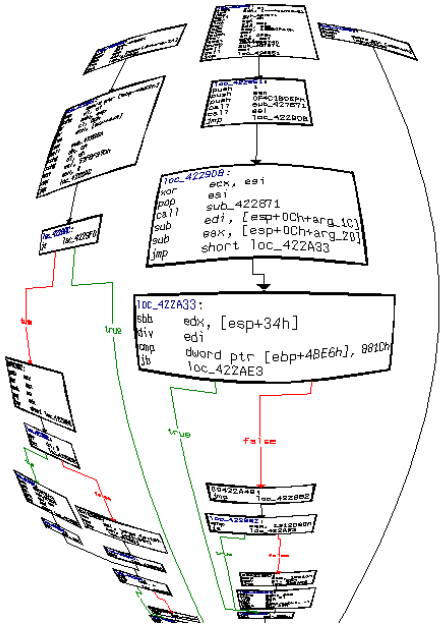
Štatistická statická analýza programového toku je vhodný doplnok, ktorý využívajú antivírusoví analytici pri skúmaní malware. Nie je však vhodná na všetky druhy malware. Ak kód používa inštrukcie meniace beh programu, ktorých operandy (cieľová adresa) sú nepriame a počítajú sa až za behu programu, statická analýza nestačí. V tom prípade sa používa napríklad dynamická analýza pomocou emulátora (viď kapitola 3.2).

4 Vizualizácia dát

Vizualizácia je oblasť, ktorá sa zaoberá vytváraním obrazovej reprezentácie rôznych dát a procesov [10]. Využíva sa v rôznych odvetviach, napríklad v medicíne, fyzike, geografii. Obraz sa dá vytvoriť na základe reálnych objektov, ale aj z abstraktných dát. Aj pri analýze malware sa využívajú vizualizácie, ktoré umožňujú pomocou grafov lepšie pochopiť a odhaliť rôzne druhy škodlivého kódu. Táto kapitola sa zaoberá popisom súčasných metód vizualizácie malware. Následne sú popísané prostriedky na vizualizáciu, ktoré by bolo možné použiť pri implementácii v tejto práci.

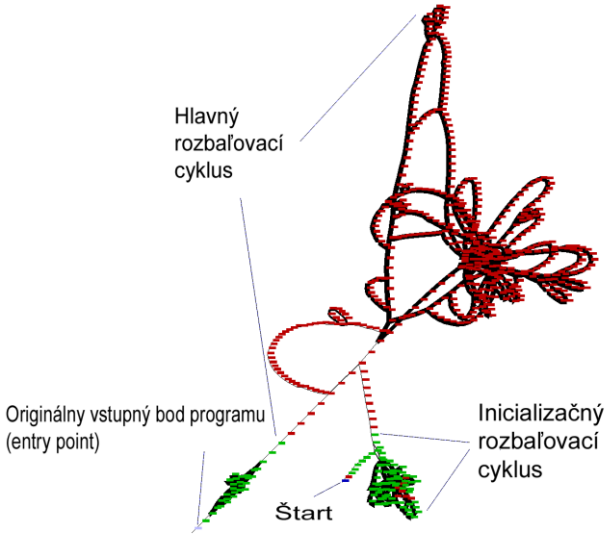
4.1 Existujúce metódy vizualizácie dát malware

Jeden z prostriedkov na analýzu malware je graf toku riadenia (spomínaný v predošlej kapitole). Jeho grafická reprezentácia sa využíva aj v iných odvetviach, napríklad ako pomôcka pri návrhu software [8]. Vizualizáciu grafu toku riadenia obsahuje aj disassembler IDA [7]. Jedná sa o 2D vizualizáciu, jednotlivé bloky sú reprezentované štvorcami, ktoré obsahujú textovú reprezentáciu programových inštrukcií. Aby sa zefektívnila práca s veľkými grafmi u rozsiahlych programov, využíva sa na priblíženie obrazu aj efekt „rybie oko“. Zachová sa takto kontext, kedy pri priblížení niektorého skúmaného úseku vidíme zároveň aj celý graf (viď obrázok 4.1).



Obrázok 4.1: Graf toku riadenia v programe IDA, efekt "rybie oko"

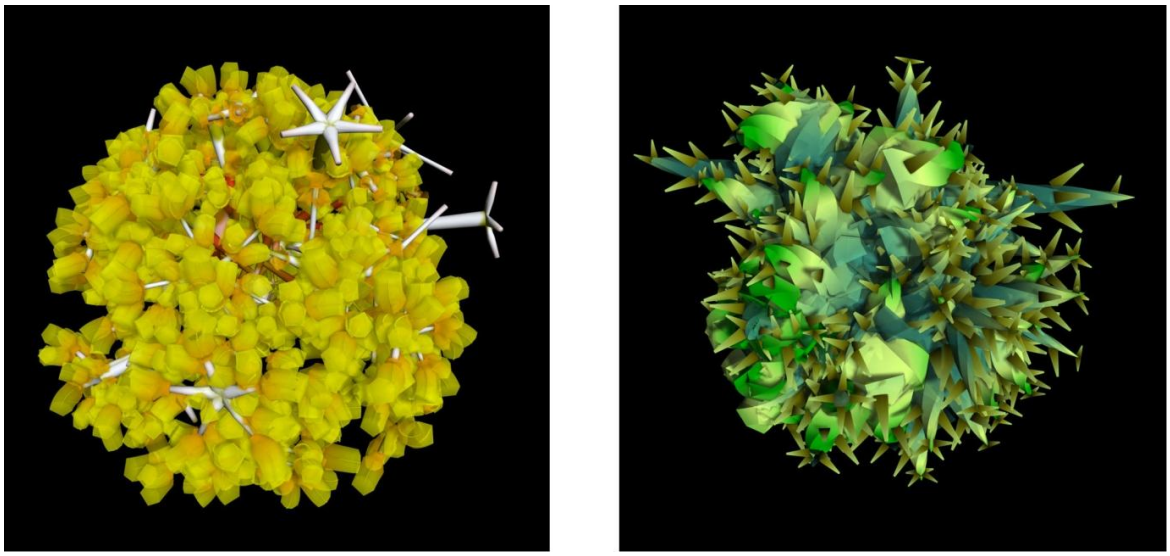
Iný príklad vizualizácie programového toku je prezentovaný v [23]. Využíva dynamickú heuristiku pomocou emulátora na získanie grafu programového toku a výpočet entropie na zvýraznenie komprimovaných častí. Graf obsahuje len tie bloky, ktoré boli počas emulácie vykonané a sú farebne odlíšené. Červenou farbou sú vyznačené bloky, ktoré sa nachádzajú v časti s vysokou entropiou. To často znamená, že táto časť je zašifrovaná alebo komprimovaná v PE EXE súbore na disku. Zelenou farbou sú vyznačené časti, ktoré sa vznikli dynamicky za behu a nenachádzajú sa v EXE súbore. Žltou sú časti, ktoré sa za behu nemenia. Toto rozdelenie napomáha v analýze vírusov. Dajú sa takto identifikovať napríklad rozbaľovacie rutiny polymorfných vírusov (viď obrázok 4.2, [23]).



Obrázok 4.2: Vizualizácia rozbaľovacieho cyklu vírusu *Netbull*[23].

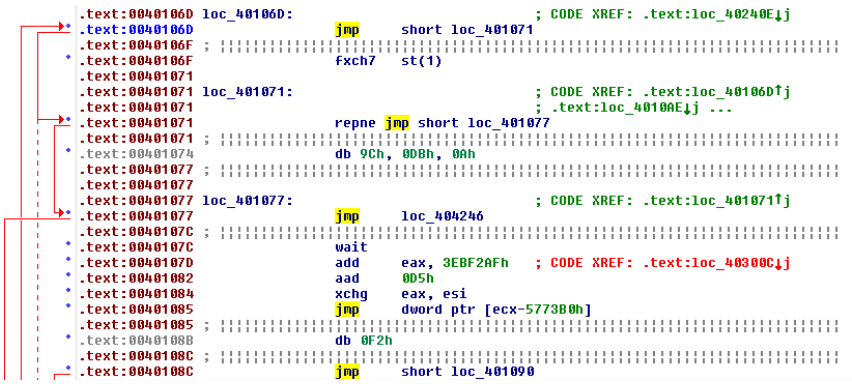
Medzi vizualizácie malware v 3D patrí *Malwarez* od Alexa Dragulescu, ktorá sa zameriava sa na vizualizáciu kódu červov, vírusov, trójskych koňov a spyware (ukážky viď obrázok 4.3). Každý úsek programového kódu, volania funkcie a pamäťové adresy sú analyzované a sledované. Ich hustota a frekvencia sú mapované na vstup algoritmu, ktorý vytvára 3D objekty. Tým pádom štruktúry ktoré sa tam nachádzajú ovplyvňujú konfiguráciu vznikajúcej grafiky [6].

Aj keď táto vizualizácia má skôr charakter umenia [29], dá využiť pri analýze červov. V [29] prezentujú metódu, ktorá výstup z tejto vizualizácie používa ako vstup pre klasifikačný algoritmus. Ten je schopný potom rozoznávať konkrétne druhy červov.



Obrázok 4.3: Vizualizácia „*Malwarez*“. Vpravo *IRCbot*, vľavo *Virutmytob* [6].

Niektoré nástroje na disassembling kódu na platforme Windows umožňujú okrem zobrazenia inštrukcií aj pomocné zobrazenie skokov. Napríklad program IDA zobrazuje programový tok pomocou šípok predstavujúcich skoky, ktoré sú zobrazené naľavo od inštrukcií programu (viď obrázok 4.4, červené šípky v ľavom okraji).



Obrázok 4.4: zobrazenie skokov v programe IDA

4.2 Prostriedky na 3D vizualizáciu dát

Na zobrazovanie 3D grafiky akcelerovanej na hardvéri sa dnes používajú najmä rozhrania (API) *OpenGL* a *Direct3D*. Nad týmito nízkoúrovňovými API sú postavené viaceré nástroje určené na 3D vizualizáciu, väčšinou to však býva *OpenGL*, pretože funguje na viacerých platformách. Niektoré nástroje ako *OpenSceneGraph*[18] a *Visualization Library* [4] sú zamerané obecnejšie a sú vhodné nielen na implementáciu vedeckej alebo dátovej vizualizácie ale aj ľubovoľnej 3D grafiky (napríklad hry). Iné ako *VTK* [26] alebo *OpenDX* [15] sa zameriavajú priamo na vizualizáciu dát. Umožňujú spracovať rôzne dáta (skaláry, vektory) a poskytujú viaceré možnosti ich vizualizácie. Obsahujú tiež skriptovací jazyk, ktorým je možné ich ovládať. Špecializovaný programovací jazyk je napríklad aj nástroj *Processing* [16], ktorý slúži rapídny vývoj grafiky a umožňuje 3D vizualizáciu postavenú nad *OpenGL*.

V tejto kapitole je podrobnejšie popísaný výber *open source* vysokoúrovňových knižníc, ktoré by mohli byť použiteľné pri implementácii v tejto práci. Sú to *Visualization Toolkit* (VTK), *Visualization Library* (VL) a *OpenSceneGraph* (OSG). Sú postavené nad *OpenGL* API, sú naprogramované v jazyku C++ a sú použiteľné na viacerých platformách. Všetky poskytujú nasledujúce možnosti:

- renderovanie plošných aj objemových dát pomocou viacerých algoritmov
- funkcie uľahčujúce programovanie 3D grafiky (operácie s vektormi a maticami, *culling*, *level od detail*, *depth sorting*, texty a fonty, a pod.)
- funkcie uľahčujúce programovanie v C++, napríklad chytré ukazatele (*smart pointer*)
- nástroje na organizáciu scény (graf scény alebo iný spôsob)
- načítanie rôznych dátových formátov
- hotové prvky pre integráciu v grafických užívateľských rozhraniach na viacerých platformách

VTK

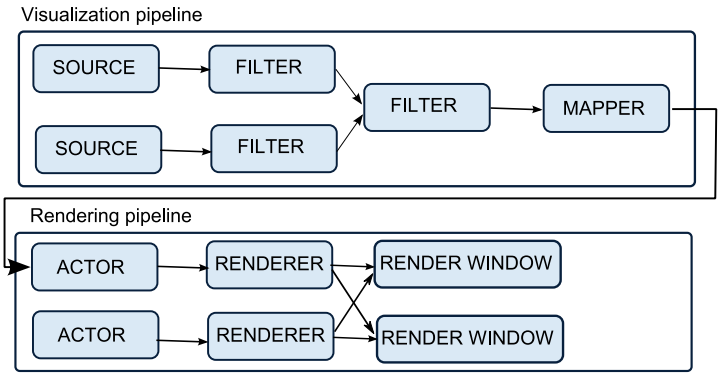
VTK (*Visualization Toolkit*) je sada nástrojov na zameraná na 3D vizualizáciu a spracovanie obrazu. Je síce naprogramovaná v jazyku C++, ale dá sa používať z viacerých programovacích jazykov (*Python*, *Tcl/Tk*, *Java*). Pozostáva z dvoch hlavných subsystémov. Jadro je knižnica pozostávajúca z C++ využívajúca princípy objektovo orientovaného programovania a interpretovaná vrstva umožňuje používanie VTK z iných jazykov.

VTK obsahuje mnoho nástrojov na vedeckú aj dátovú vizualizáciu, napríklad (kompletný zoznam viď [14]):

- algoritmy na vizualizáciu skalárnych a vektorových dát
- vizualizácia informácií (napríklad rôzne algoritmy na rozvrhnutie grafov)
- 3D a 2D modelovacie algoritmy (napríklad decimácia, orezávanie, generovanie normál, extrudovanie, generovanie rôznych objektov ako guľa, valec, atď.)

Základ VTK je takzvaná grafická a vizualizačná linka (*pipeline*), viď obrázok 4.5. Vizualizačná časť transformuje dáta pomocou filtrov. Dátové zdroje (*source*) načítajú dáta zo súboru alebo ich procedurálne generujú (napríklad rôzne geometrické objekty). Ich výstupom sú dátové objekty. Filtre sú prvky s viacerými vstupmi aj výstupmi, ktoré dáta transformujú pomocou rôznych algoritmov. Koniec vizualizačnej časti predstavuje objekt *Mapper*, ktorý transformuje dátové objekty na grafické objekty, ktoré vie zobraziť grafické jadro VTK [26].

Grafické jadro pozostáva z objektov typu *Actor*, ktorý predstavuje konkrétny objekt v 3D alebo 2D scéne. Napája sa na výstup objektu *Mapper*. Objekty *Renderer* a *RenderWindow* spravujú prepojenie s grafickým užívateľským rozhraním (GUI) operačného systému. *RenderWindow* predstavuje konkrétne okno v GUI, na ktoré kreslí objekt *Renderer*. Na jedno okno sa môže vykresľovať viacero objektov typu *Renderer* a naopak, jeden *Renderer* môže zobrazovať obraz na viacerých oknách. *Renderer* predstavuje aj viditeľnú oblasť scény. Môže obsahovať viac objektov typu *Actor*, ktoré sa v nej zobrazujú. VTK kompletne implementuje viaceré princípy 3D renderovania pomocou *OpenGL* (svetlá, vlastnosti materiálu, textúrovanie, primitíva ako trojuholníky, čiary a pod.) a kompletný renderovací *engine*. Samotné *OpenGL* nemusí užívateľ VTK ovládať.



Obrázok 4.5: Architektúra VTK

Visualization Library

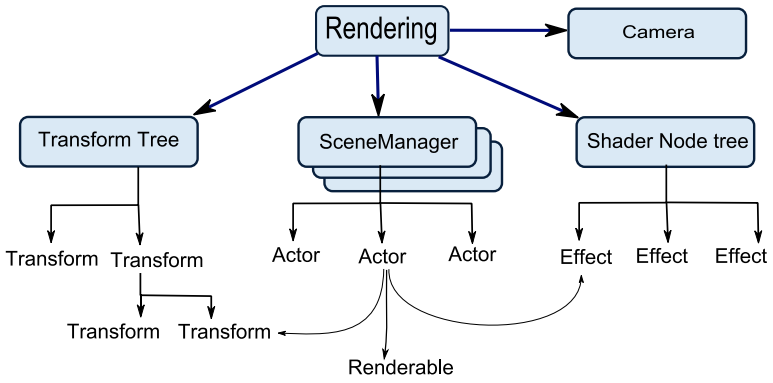
Visualization Library (VL) je knižnica, ktorá implementuje tzv. tenkú vrstvu nad *OpenGL* API. Na rozdiel od VTK neimplementuje kompletný renderovací *engine*, ale snaží sa princípmi byť čo najbližšie k *OpenGL*. Často sú funkcie z *OpenGL* a triedy z VL vo vzťahu 1:1. Jedná skôr o obecný nástroj na programovanie grafiky. Obsahuje okrem algoritmov implementujúcich vizualizačné techniky aj nástroje uľahčujúce napríklad tvorbu hier. Z algoritmov určených na vyzualizáciu dát je to napríklad generovanie geometrických objektov, decimácia (zjednodušenie) polygonálnych modelov, vizualizácia molekúl, Catmull-Rom krivky, Bezierove plochy.

Architektúra *Visualization Library* pozostáva z viacerých objektov (viď obrázok 4.6). Na rozdiel napríklad od OSG nezakladá sa iba na grafe scény ako dátovej štruktúre, ktorou by boli reprezentované naraz viaceré prvky (transformácie, priestorové vzťahy, viditeľnosť a pod.) [4].

Základný prvok je trieda *Actor*, ktorou je reprezentovaný každý objekt nachádzajúci sa v scéne. Scéna nepozostáva z jedného grafu scény, ale z troch samostatných štruktúr:

- Manažér scény (*Scene manager*), ktorý má za úlohu zoskupovať prvky *Actor* podľa ich priestorovej alebo logickej príslušnosti. Implementuje špecifický algoritmus na ich priestorové delenie a správu viditeľnosti. Napríklad generický strom podobne ako graf scény (*Actor tree*), KD strom, alebo portály. Manažérov scény môže byť pri vykresľovaní scény použitých viacero druhov podľa potreby.
- Strom transformácií (*Transform tree*), ktorý predstavuje hierarchiu transformácií v scéne.
- Strom efektov (*ShaderNode tree*) predstavuje hierarchiu použitých efektov (*Effect*).

Actor v sebe spája tri druhy informácie: geometriu, ktorú treba vykresliť (*Renderable*), transformáciu, a použitý efekt. Použitie efektu a transformácie je voliteľné. Objekt, ktorý v sebe združuje všetky potrebné informácie na vykreslenie scény (kamera, manažér scény, strom transformácií, strom efektov) sa nazýva *Rendering*.



Obrázok 4.6: Architektúra *Visualization Library*

OpenSceneGraph

OpenSceneGraph (OSG) je open source knižnica zameriavajúca sa na tvorbu grafických aplikácií pomocou grafu scény. Slúži ako nástroj pre vývoj širokého spektra grafických aplikácií, napríklad vedecké simulácie a vizualizácie, hry, virtuálna realita. Medzi hlavné vlastnosti OSG patrí:

- Graf scény – hlavná časť OSG. Zabezpečuje viaceré funkcie vrátane renderovania, animácie a pod. Zapuzdruje väčšinu funkcionality *OpenGL*. Automaticky implementuje rôzne techniky ako napríklad *level of detail, occlusion culling, OpenGL state sorting* (triedenie renderovaných primitív tak, aby sa stav v *OpenGL* menil čo najmenej).
- Rozšírenia grafu scény (*NodeKits*) – rozšírujú graf scény o funkcia ako napríklad časticové systémy, grafické efekty (napríklad tieňovanie), text, renderovanie terénu.

Graf scény je hierarchická stromová štruktúra pomocou ktorej sa organizujú priestorové dáta kvôli efektívnemu renderovaniu. Graf scény začína koreňovým uzlom z ktorého vyrastajú ďalšie uzly. Uzly môžu mať jeden a viac dcérskych uzlov. Listy stromu obsahujú konkrétnu geometriu prvkov v scéne. Uzly okrem zoskupovania jednotlivých častí grafu majú rôzne funkcie. Napríklad transformačné uzly aplikujú transformáciu na všetky svoje poduzly. Uzol typu LOD (*level of detail*) pri zobrazovaní prepína medzi svojimi podgrafmi ktoré obsahujú geometriu objektu v rôznych úrovniach detailu na základe toho, ako ďaleko sa daný objekt nachádza od kamery [18].

5 Návrh vizualizácie

Súčasné vizualizácie malware sa zameriavajú najmä na 2D grafy toku riadenia (viď kapitola 4.1), ktoré sú vhodným doplnkom pri analýze a detekcii škodlivého kódu. Okrem tejto metódy sa však používa aj štatistická analýza programového toku, ktorá sa zaoberá štruktúrou a početnosťou skokov meniacich beh programu (popísaná v kapitole 3.4). Bolo by užitočné vytvoriť vizuálnu reprezentáciu programového toku zobrazujúcu tieto skoky, ktorá by uľahčila prácu antivírusových výskumníkov používajúcich túto metódu. Cieľom je túto reprezentáciu navrhnúť a implementovať. Vzniknutý nástroj bude slúžiť ako pomôcka pri analýze malware v spoločnosti AVG.

Táto kapitola sa zaoberá zhodnoteniu existujúcich metód vizualizácie malware a návrhu vizualizácie programového toku pomocou zobrazenia skokov v spustiteľných súboroch. Prezentované sú dva návrhy 3D vizualizácie. Na konci sa riešia aj požiadavky na výslednú aplikáciu a jej grafické užívateľské rozhranie.

Existujúce disassemblery umožňujú zobraziť skoky ovplyvňujúce programový tok okrem textovej podoby aj formou vizualizácie. Ako príklad bol uvedený v kapitole 4.1 program IDA, ktorý zobrazuje skoky pomocou šípok vedľa inštrukcií programu. Táto reprezentácia sa hodí na podrobnú analýzu programového kódu, avšak nie je veľmi vhodná na získanie celkového pohľadu a na mediálnu prezentáciu. Prínos vytvoreného nástroja by mal byť v tom, že na identifikáciu podozrivého kódu nebude potrebné podrobné skúmanie, ale malo by to byť možné na prvý pohľad. Vizualizácia by mala byť estetická, vhodná nielen na praktické použitie ale aj na mediálnu prezentáciu pre laickú verejnosť.

Vzniknutá vizualizácia nemá nahradiť všetky možnosti zobrazení pomocou grafu toku riadenia (viď kapitola 4.1), ale vhodne ich dopĺňať napríklad tým, že na rozdiel od predošlého poskytne dobrý celkový pohľad na napadnutý súbor. Vhodným výsledkom by bolo, keby z grafu bolo na prvý pohľad viditeľné podozrivé miesto kde sa nachádza kód malware podľa jeho špecifických znakov (napríklad veľké množstvo náhodne rozmiestnených skokov). Ideálne by bolo, keby na základe tejto vizualizácie bolo možné rozoznávať napríklad niektoré konkrétne rodiny malware.

Vizualizáciu programového toku som sa rozhodol navrhnúť v trojrozmernom priestore, pretože umožňuje zobraziť v grafe viacero veličín a poskytuje tiež viac možností na návrh estetickej grafiky. To, že sa má jednať o 3D graf, bola aj jedna z konkrétnych požiadaviek zadávateľa. Na základe konzultácie vo firme AVG sú vo vizualizácii zobrazené výhradne nepodmienené skoky typu *relative near jump*, ktoré sú pre analýzu kódu malware najviac zaujímavé (viď kapitola 3.4).

Rozhodol som sa vytvoriť dva grafy zobrazujúce skoky v EXE súbore: takzvaný *graf skokov* a *graf početnosti*. Každý z nich je zameraný na iný aspekt štatistickej analýzy programového toku. Každý graf je navrhnutý tak, aby mal nasledujúce vlastnosti:

- Graf korešponduje s binárnym usporiadaním PE EXE súboru na disku. Je delený na jednotlivé sekcie.
- Skoky sú v grafoch reprezentované ako krivky v 3D priestore spájajúce začiatočnú a koncovú adresu skoku. Aby sa zvýraznil priestor, je pri ich kreslení použitý efekt, ktorý zobrazuje skoky ako žiariace 3D čiary pozostávajúce z dvoch farieb. Tento efekt umožňuje v priestore napríklad odlíšiť prekrývajúce sa skoky.
- Skoky sú farebne odlíšené podľa smeru (dopredu a dozadu).
- V grafe je možné zobraziť početnosť skokov na danej RAW adrese v sekcii PE EXE.

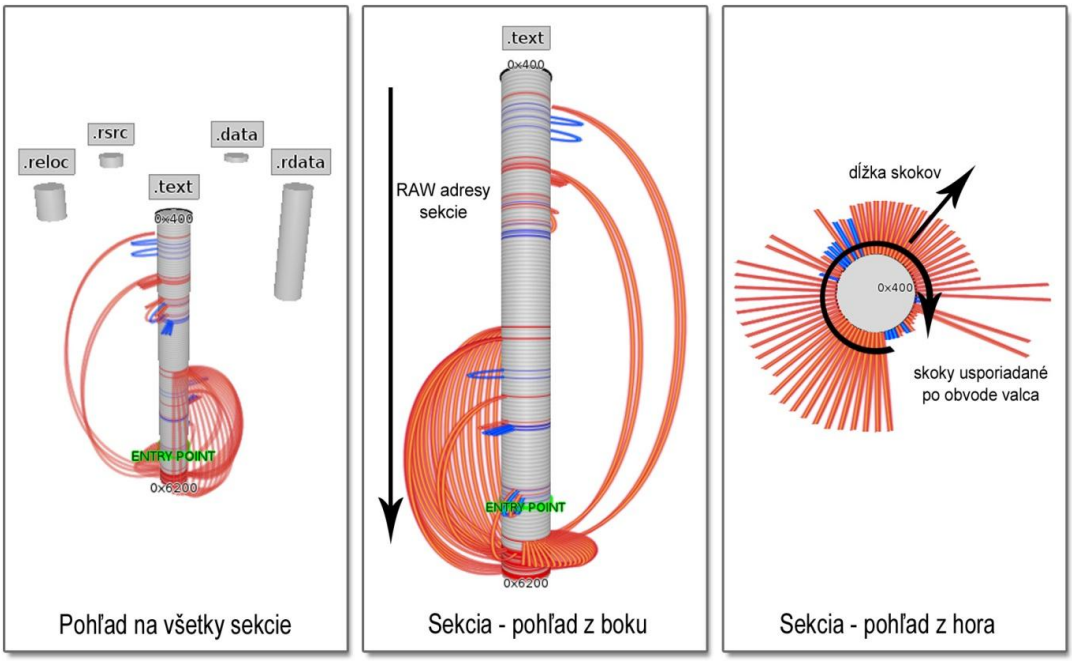
- Každý graf má dva farebné štýly, jeden hodiaci sa na prezentáciu na obrazovke, druhý na prezentáciu na papieri (biele pozadie šetriace náplň tlačiarne).
- Miesta (adresy), kde skoky začínajú alebo končia sú odlíšené od tých, kde skoky len prebiehajú. Toto umožní lepšie rozoznať oblasti, kde sa nachádzajú nejaké inštrukcie od oblastí, kde sa nachádzajú iné dáta.

5.1 Graf skokov

Graf skokov je zameraný na to, aby bola dobre rozoznateľná ich štruktúra (vzájomné usporiadanie). Návrh tohto grafu je inšpirovaný siločiarami magnetu. Jednotlivé sekcie PE EXE súboru sú znázornené ako valec na ktorom sú po jeho obvode vyznačené skoky. Os pozdĺž valca predstavuje adresy danej sekcie, dĺžka valca je určená veľkosťou sekcie. Pohľad zboku znázorňuje skoky ako oblúky spájajúce začiatočnú a koncovú adresu skoku. Výška oblúku je určené dĺžkou skoku. Poloha skoku je vzhľadom na os valca (dĺžku valca) určená jeho začiatočnou a koncovou adresou. Skoky sú usporiadané po obvode valca podľa ich poradia, ktoré je určené nižšou adresou skoku (tj. u skoku dopredu to je začiatočná adresa, u skoku dozadu koncová). Skoky sú farebne odlíšené podľa smeru (modrou sú skoky smerom dopredu, červenou dozadu). V grafe budú obsiahnuté textové informácie podávajúce užívateľovi ďalšie informácie, ako napríklad názov sekcie, jej začiatočná a konečná adresa a poloha vstupného bodu programu (*entry point*).

Aby boli adresy, kde začínajú alebo končia skoky odlíšené od tých, kde len prebiehajú, sú na valci pomocou textúry farebne vyznačené kruhy určujúce miesta, kde sa nachádzajú inštrukcie skokov alebo ich cieľová adresa. Ich farba korešponduje s farbou prislúchajúceho skoku. Ak sa danej adrese ústi viacero skokov rôznych typov, farba sa mixuje na základe ich pomeru.

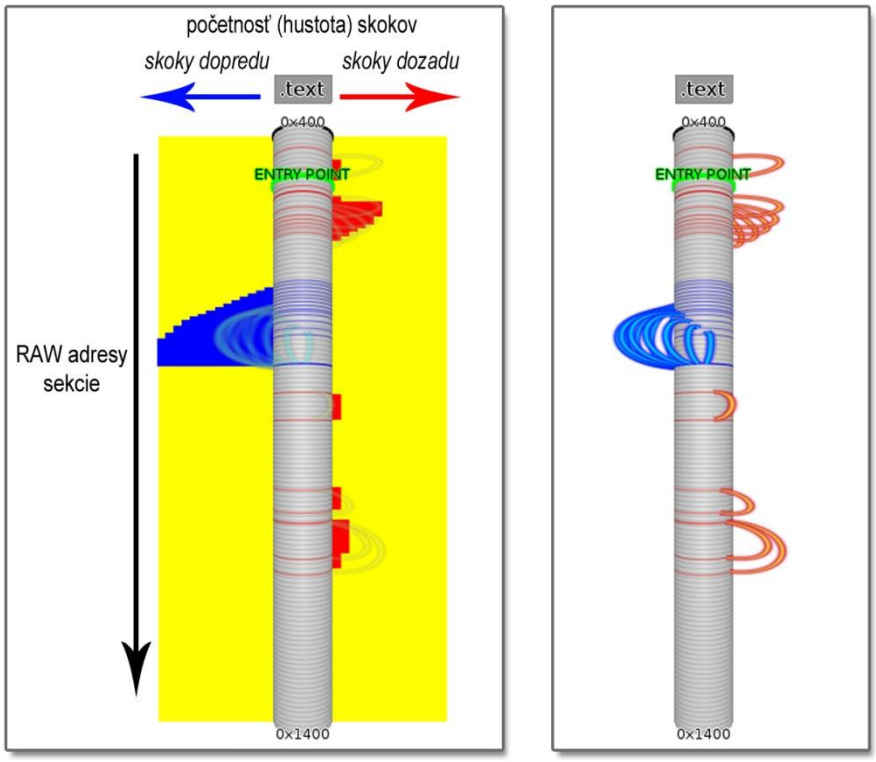
Pohľad zboku by mal umožňovať náhľad na priebeh jednotlivých skokov rozoznať adresy, kde sa vyskytujú. Pri pohľade zhora by malo byť možné vidieť všetky skoky tak, že sa neprekrývajú, rozoznať ich dĺžku a usporiadanie. Zvolené usporiadanie pomáha pri znázornení štruktúry skokov, pretože takto sú dobre viditeľné striedajúce sa zoskupenia skokov v oboch smeroch. Ukážka zobrazenia sekcie obsahujúcej binárny kód programu je na obrázku 5.1 v strede a vpravo.



Obrázok 5.1: Návrh grafu skokov

Graf umožňuje dva režimy zobrazenia. Prvý poskytuje náhľad na všetky sekcie, ktoré sú usporiadané do kruhu (viď obrázok 5.1 vľavo). Umožňuje to získať predstavu o štruktúre zobrazeného EXE súboru, identifikovať sekcie, kde sa nachádza binárny kód a kde sa nachádza vstupný bod programu. V tomto režime sa zobrazujú aj skoky medzi sekciami, ak sa v programe nejaké vyskytujú. Druhý režim zobrazenia zobrazuje len jednu zvolenú sekciu. V tomto režime sa dá so sekciou ľubovoľne manipulovať (otáčať, zoomovať a podobne, viď obrázok 5.1 v strede a vpravo).

V grafe je možné voliteľne zobraziť početnosť skokov (viď obrázok 5.2). Tá je zobrazená na 2D ploche pomocou histogramu samostatne pre skoky smerujúce dopredu a dozadu. Plocha sa bude automaticky natáčať smerom ku kamere (princíp *billboardu*), aby bolo početnosť stále vidno pri manipulácii s grafom v 3D priestore. V historgame je na osi X je zobrazená početnosť skokov prechádzajúcich danou adresou. Intervaly na osi Y (rovnobežne s valcom) prislúchajú jednotlivým adresám v PE EXE súbore. Oba histogramy početnosti sú vykreslené v rovnakej farbe ako im prislúchajúci typ skokov (modrá a červená) po stranách valca znázorňujúceho sekciu. V tomto režime zobrazenia budú skoky vyznačené menej výrazne sivou farbou, aby vyniklo zobrazenie ich početnosti.



Obrázok 5.2: Zobrazenie početnosti skokov (vľavo zapnuté, vpravo vypnuté)

Manipulácia s grafom

Základná manipulácia s grafom v 3D priestore bude realizovaná pomocou nástroja *trackball*. Ten umožňuje rotovať objekt vo všetkých smeroch, presúvať, priblížiť a oddialiť. Manipuláciu s grafom uľahčujú nasledujúce funkcie:

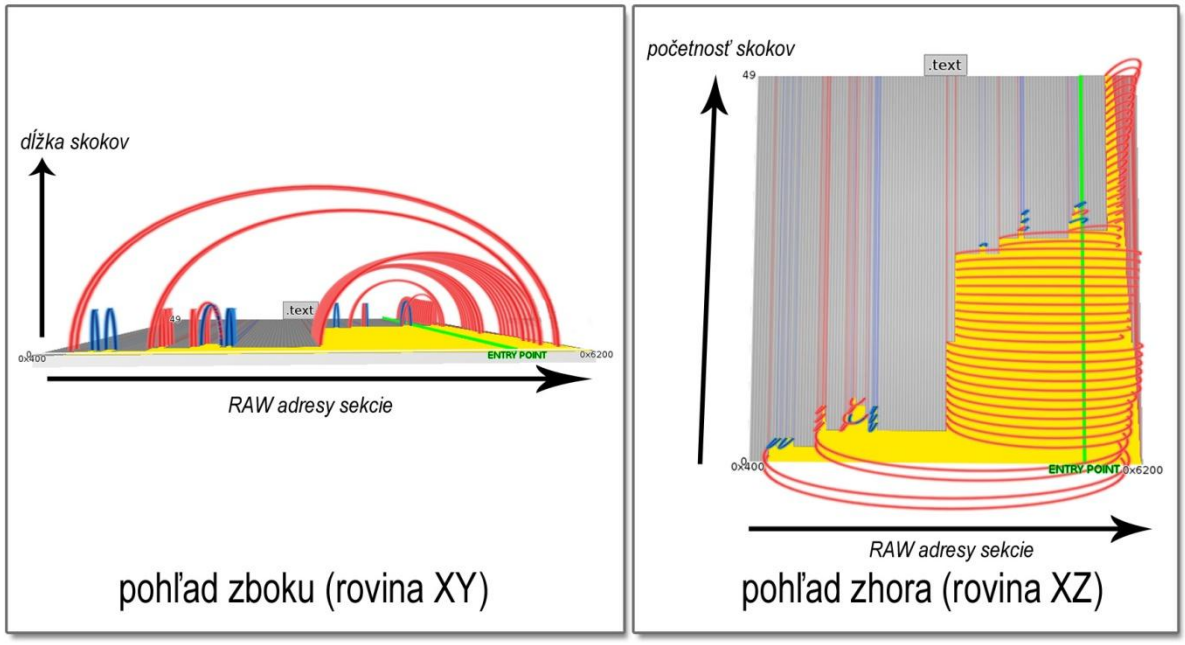
- V režime zobrazenia jednej sekcie je možné sekciu posúvať iba pozdĺž jej osi. Je to kvôli lepšej manipulácii, aby sa užívateľ pri skúmaní potenciálne dlhej sekcie nestratil.
- Graf obsahuje rotačný režim, kedy je možné sekciu točiť iba okolo jej osi.

- Keďže sú skoky rozmiestnené po obvode valca, nie sú vždy všetky dobre viditeľné pri pohľade zboku. Druhý rotačný preto režim automaticky natočí graf tak, aby bol skok nachádzajúci sa pod kurzorom orientovaný vždy ku kamere.
- Pri manipulácii s grafom sa zobrazuje na valci predstavujúcom sekciu značka zobrazujúca aktuálnu pozíciu ako RAW adresu v EXE súbore.

Často je potrebné pri skúmaní skokov v programe priblížiť jeden skúmaný výsek, a pritom si zachovať kontext, čo znamená vidieť aj zároveň celý súbor. Program IDA používa napríklad pri zobrazovaní grafu toku riadenia efekt *rybie oko* (viď kapitola 4.1). Týmto som sa inšpiroval a do grafu som pridal možnosť priblížiť jeden zaujímavý úsek, pričom je vidno zároveň aj celú sekciu. Okraje budú deformované spôsobom akým sa počíta súdkovité skreslenie (*barrel distortion*).

5.2 Graf početnosti

Predošlý graf síce umožňuje zobraziť početnosť (hustotu) skokov, ale jedná sa o voliteľné zobrazenie. V základnom režime zobrazenia nie je početnosť skokov z ich usporiadania na prvý pohľad dobre rozoznateľná. Preto som vytvoril ešte jeden variant zobrazenia programového toku, ktorý je zameraný na zobrazenie početnosti skokov na danej adrese v EXE súbore. Slúži ako doplnok ku grafu skokov. Základná idea návrhu tohto grafu je, že čiary predstavujúce skoky by mohli byť v jednej rovine usporiadané tak, aby sa dala rozpoznať ich početnosť priamo podľa ich usporiadania, bez potreby zobraziť navyše histogram, ako to bolo v predošlom grafe.



Obrázok 5.3: Návrh grafu početnosti skokov sekcie EXE súboru

Skoky sú tu tak isto ako v predošlej variante zobrazené formou oblúkov. Sekcie sú znázornené ako kváder. Pri pohľade zboku (rovina XY, viď obrázok 5.3 vľavo) predstavuje os X adresy v EXE súbore, os Y predstavuje dĺžku skokov (výška oblúku znázorňujúceho skoky je určená ich dĺžkou). Pri pohľade zhora predstavuje os Z početnosť skokov na danej adrese. V rovine XZ sú skoky usporiadané tak, aby vynikla ich početnosť (poloha skoku na osi Z je určená hodnotou početnosti na jeho začiatočnej adrese). Pre väčšiu názornosť je v tejto rovine tiež zobrazený histogram znázorňujúci

početnosť skokov (žltá farba, spoločne pre oba typy skokov, viď obrázok 5.3 vpravo). Podobne ako v predošlom grafe sú na kvádri farebnými pruhmi vyznačené adresy, na ktorých začína alebo končí nejaký skok. Tento graf bude obsahovať iba jeden režim zobrazenia, v ktorom sú sekcie usporiadané vedľa seba. Manipuluje sa vždy len s jednou aktuálnou sekciou, ktorá je v strede okna a bude možné medzi nimi prepínať.

5.3 Návrh aplikácie

Nástroj na vizualizáciu skokov bude slúžiť antivírusovým výskumníkom v spoločnosti AVG ako pomôcka pri analýze malware. Program má byť riešený ako jednoduchý nástroj, ktorý je možné spustiť na platforme Windows bez inštalácie, aby neboli potrebné administrátorské práva na jeho prvé spustenie. Aby šlo použiť výstupy programu ako mediálnu prezentáciu, mal by program okrem samotného zobrazovania grafov umožňovať aj export do obrázka.

Grafické užívateľské rozhranie programu (ďalej len GUI) bude riešené ako *Multiple document interface* (MDI) pomocou záložiek, aby umožnilo otvoriť viacero súborov naraz. Takto sa dá napríklad rýchlo porovnať vírusom napadnutý súbor s originálom. GUI programu bude obsahovať štandardné prvky ako menu a panel nástrojov, v ktorom budú začlenené najpoužívanejšie funkcie. Keďže vizualizácia umožňuje viacero druhov manipulácie s grafom ktoré budú kontrolované výhradne pomocou klávesových skratiek, malo by byť možné zobraziť rýchlu pomoc s informáciami o ovládaní programu priamo v grafe.

6 Implementácia

Táto kapitola sa zaoberá implementáciou vizualizácie. Najskôr popisuje použité prostriedky a následne konkrétne prvky a efekty, ktoré boli pri implementácii použité. Kapitola sa venuje aj princípom implementácie interakcie s užívateľom. Na konci sa venuje popisu načítania EXE súboru a inštrukcií skokov a spôsobu, akým je riešené grafické užívateľské rozhranie a konfigurácia programu.

Pre implementáciu nástroja na vizualizáciu programového toku bol zvolený jazyk C++, lebo na rozdiel od nízkoúrovňových jazykov ako napríklad C má viac možností (objektovo orientované programovanie, preťažovanie operátorov, štandardná knižnica STL) a zároveň umožňuje jednoduchý prístup k nízkoúrovňovým API ako *OpenGL*. Pre tento jazyk tiež existuje široký výber knižníc a nástrojov použiteľných na tvorbu 3D vizualizácie (viď kapitola 4.2). Program je kompilovaný pomocou vývojového nástroja *Microsoft Visual Studio 2008*.

Na vykresľovanie 3D grafiky je použité *OpenGL* API. Program má byť síce funkčný hlavne na platforme Windows, ale použitím tohto API ostáva stále možnosť portovať program aj na iné platformy. Z dostupných nástrojov na 3D vizualizáciu som zvolil knižnicu *Visualization Library* (ďalej len VL) postavenú nad *OpenGL* API. Táto knižnica mi vyhovovala svojou architektúrou, pretože je úzko prepojená s *OpenGL* API a zároveň poskytuje dostatok možností. V úvodnej fáze bola implementovaná testovacia vizualizácia skokov pomocou *VTK toolkitu*, ktorý obsahuje v porovnaní s VL viac algoritmov užitočných na vizualizáciu (krivky, predpripravené 3D *Widgety* a pod.), ale neumožňuje takú kontrolu nad kreslením v *OpenGL* ako VL. Napríklad VL vie použiť *vertex buffer object* (VBO) ako obecné pole, ktoré je možné neskôr aj meniť. VL tiež podporuje lepšie moderné funkcie *OpenGL* ako *shadery* a VBO (tie používa implicitne ak sú v danej implementácii *OpenGL* dostupné). Pri návrhu bola tiež zvažovaná knižnica OSG, ale nakoniec bolo od toho upustené, pretože pri implementácii vizualizácie sa viac hodil spôsob organizácie scény použitý v knižnici VL (viď kapitola 6.2).

Grafické užívateľské rozhranie je implementované pomocou knižnice QT. Tá je zvolená nie len kvôli jej prenositeľnosti, ale hlavne vďaka širokým možnostiam a jednoduchému použitiu aj na platforme Windows (existuje zásuvný modul pre *Visual Studio* uľahčujúci kompiláciu a návrh GUI).

Program je kompilovaný ako jediný binárny súbor, ktorý nepotrebuje iné neštandardné DLL knižnice ani inštaláciu (všetky knižnice vrátane QT sú k programu prilinkované staticky). Rôzne dátové súbory potrebné pre beh programu (napríklad textúry) sú zakompilované do spustiteľného súboru pomocou systému *Resources* v knižnici QT.

6.1 Volumetrické čiary

Skoky sú vo vizualizácii znázornené ako krivky v 3D priestore spájajúce začiatočnú adresu s koncovou adresou pomocou efektu pripomínajúceho žiariaci lúč. Efekt je implementovaný pomocou takzvaných falošných volumetrických čiar (tzn. že vyzerajú ako keby boli trojrozmerné).

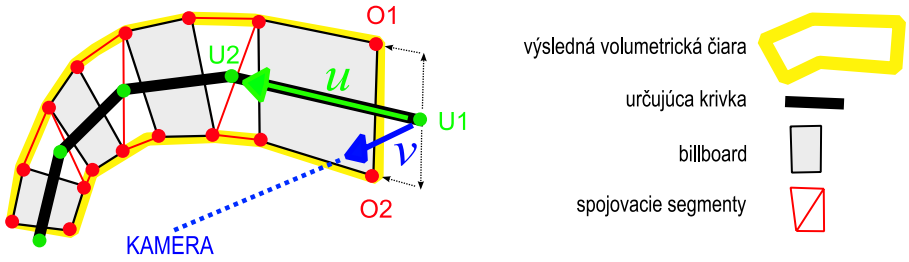
Prvý variant implementácie kreslenia počítal s kreslením čiar pomocou *OpenGL* primitív *GL_LINES*. Aby sa zlepšilo priestorové vnímanie čiar, počítalo sa s pridaním halo efektu, podobne ako prezentujú v [27]. Čiary v *OpenGL* však neumožňujú textúrovanie a okrem toho sú zobrazované vždy bez perspektívy (vzdialené čiary sú rovnako hrubé ako blízke). Ak majú byť čiary zobrazené pomocou antialiasingu, je potrebné v *OpenGL* použiť blending. Pri jeho použití sú ale viditeľné spoje medzi jednotlivými segmentmi (čiarami) z ktorých pozostáva krivka.

Kreslenie čiar s priestorovým efektom je nakoniec implementované pomocou obdĺžnikových navzájom pospájaných segmentov. Každý segment sa natáča okolo osi určenej stredom čiary stále ku kamere (princíp *billboardu*). Priestorový efekt je dosiahnutý aplikáciou textúry, ktorá vytvára ilúziu žiariaceho lúča. Princíp je inšpirovaný kreslením volumetrických čiar prezentovaným v [17] a [11], ale je kúsok pozmenený, aby sa viac hodil na kreslenie spojitých kriviek namiesto samostatných čiar.

Volumetrická čiara je určená v 3D priestore krivkou pozostávajúcou z úsečiek (viď obrázok 6.1, hrubá čierna čiara). Každej úsečke prislúcha jeden obdĺžnikovitý segment volumetrickej čiary (*billboard*) určený dvoma koncovými bodmi úsečky. Aby bol tento *billboard* natočený stále smerom ku kamere okolo osi určenej úsečkou, vypočítajú sa jeho vrcholy posunutím o vektor, ktorý sa získa vektorovým súčinom osi úsečky a vektoru určujúceho smer ku kamere (viď rovnica 6.1 a Obrázok 6.1). Ďalej je potrebné *billboard* skrátiť, aby sa neprekrýval so susednými v prípade, že je krivka veľmi ostro zatočená. Vzniknuté medzery je možné vyplniť ďalším spojovacím segmentom pozostávajúcim z trojuholníkov (viď obrázok 6.1). Týmto sa krivka ešte viac zjemní, pretože pozostáva z viacero častí. Skrátením *billboardu* o štvrtinu pôvodnej dĺžky úsečky na oboch stranách sa dosiahne toho, že všetky segmenty budú mať rovnakú dĺžku a čiara bude mať pravidelnú štruktúru. Výpočet vrcholov jednej strany *billboardu* sa dá vyjadriť pomocou vzťahu:

$$O = U_1 \pm r(\vec{u} \times \vec{v}) - \vec{u}\left(\frac{|U_1 - U_2|}{4}\right) \tag{6.1}$$

O je vrchol obdĺžnika, U_1 a U_2 sú vrcholy úsečky, r je polomer volumetrickej čiary a $\vec{u}$ a $\vec{v}$ sú normalizované vektory, prvý určuje je smer úsečky a druhý určuje smer ku kamere. Pre výpočet protiľahlej strany platí rovnaký vzťah, len treba vymeniť vrcholy úsečky.



Obrázok 6.1: Princíp zobrazovania volumetrických čiar

Na urýchlenia kreslenia veľkého počtu skokov sa používa priemerovanie. Keď sú skoky rovnakého typu blízko seba a ich začiatočné aj koncové adresy sú v určitom intervale, sú reprezentované jednou spoločnou krivkou. Jej hrúbka je priamo úmerná počtu skokov, ktoré znázorňuje.

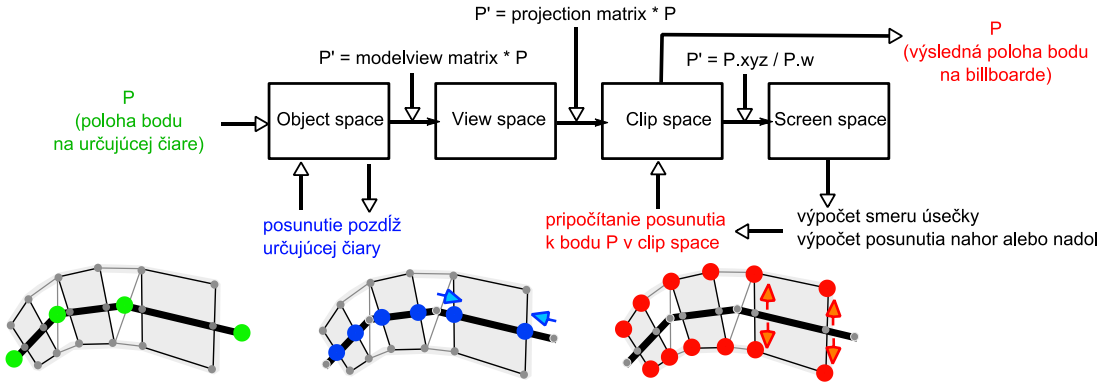
Kreslenie čiar je implementované pomocou *vertex* a *fragment shaderov* v jazyku GLSL a potrebné dáta sú uložené vo VBO (*vertex buffer object*). Pozície určujúcej čiary sú uložené pre každý bod 4 krát, pretože na jeden jej bod pripadajú dva vrcholy dvoch susedných *billboardov* (okrem krajov, kde sú to len 2 body prislúchajúce jednému *billboardu*). *Billboardy* aj spojovacie segmenty sú kreslené pomocou trojuholníkových primitív (*GL_TRIANGLES*) a indexového poľa. Všetky krivky (skoky) v scéne sú uložené v jednom spoločnom VBO. Týmto spôsobom sa môžu vykresliť len pomocou jedného volania funkcie *glDrawElements*, a tak sa zabráni réžii vzniknutej viacerými volaniami *OpenGL* API.

Pre každý vrchol *billboardu* sú vo VBO uložené tieto údaje:

- pozícia bodu určujúcej čiary (4D vektor)

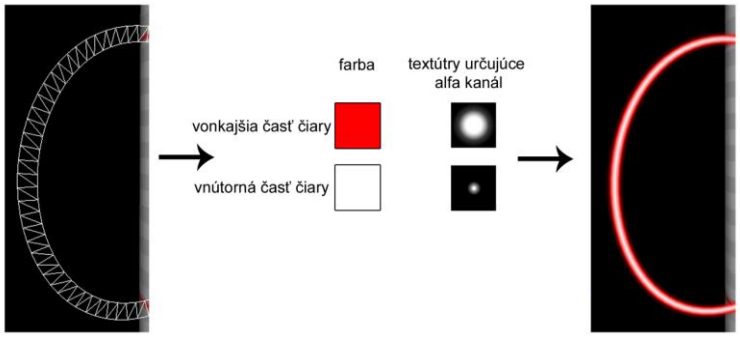
- pozícia protiľahlého bodu určujúcej čiary (4D vektor)
- textúrovacie súradnice (2D vektor)
- ďalšie atribúty, ktoré sú uložené v jednom 4D vektore: typ skoku, príznak, či sa jedná o vrchný alebo spodný bod, váha čiary (určuje koľko skokov reprezentuje daná čiara), a jeden rezervovaný pre budúce použitie

Vo *vertex shaderi* sa počíta poloha *billboardu* na základe dvoch pozícii určujúcej čiary. Najprv sa upraví poloha bodu, aby sa susedné *billboardy* neprekrývali tak, že sa *billboard* skráti posunutím bodu o štvrtinu dĺžky *billboardu* pozdĺž čiary. Na výpočet vrcholu *billboardu* sa nepoužíva presne vektor, ktorý určuje smer ku kamere, ale spôsob ktorý prezentujú v [17]. Násobením súradnice bodu modelovo-pohľadovou maticou (*modelview matrix*) a projekčnou maticou (*projection matrix*) sa získa orezávacia súradnica bodu (*clip space*). Tá sa vydelí štvrtým prvkom súradnice *w* (*perspective divide*). Výsledok je súradnica na monitore (*screen space*). V tomto priestore sa vypočíta vektor kolmý na smer určujúcej čiary, ktorého veľkosť je rovná polovici hrúbky výslednej volumetrickej čiary. Tento vektor sa nakoniec pripočíta k orezávaciej súradnici (*clip space*). Smer posunutia (nahor alebo nadol) určuje jeden z atribútov uložených vo VBO. Celý postup ja naznačený na obrázku 6.2. Nakoniec sa na základe atribútu špecifikujúceho typ skoku predá do *fragment shaderu* presná hodnota farieb z ktorých bude pozostávať výsledný efekt.



Obrázok 6.2: Výpočet polohy vo *vertex shaderi*

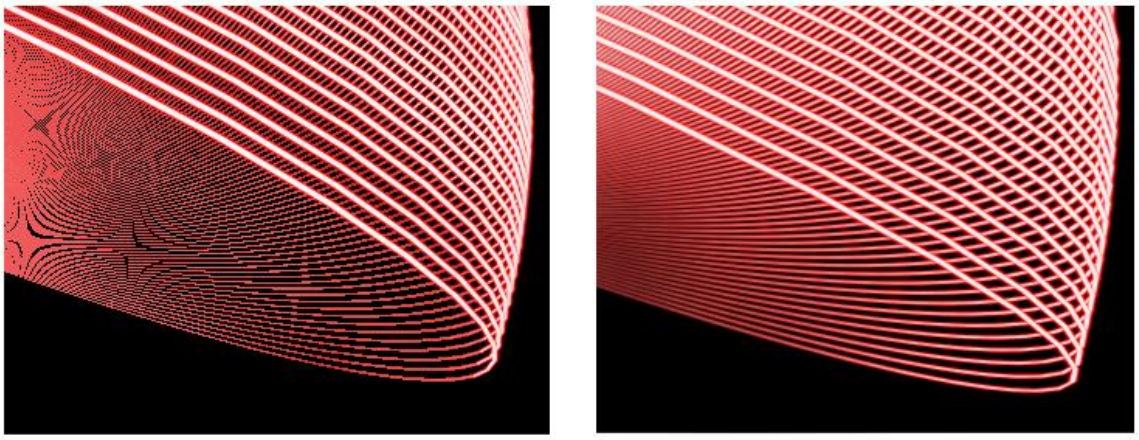
Vo *fragment shaderi* sa pomocou textúrovacích súradníc aplikujú dve textúry, ktoré dohromady vytvárajú efekt žiary pozostávajúci z dvoch farieb. Prvá textúra špecifikuje alfa kanál pre vonkajšiu časť čiary, druhá pre vnútornú. Postup je znázornený na obrázku 6.3.



Obrázok 6.3: Textúrovanie vo *fragment shaderi*

Keďže sa pri vytváraní efektu žiary používa v *OpenGL* alfa kanál (*blending*), je potrebné vykresľovať jednotlivé trojuholníky z ktorých pozostávajú čiary podľa ich poradia od kamery (tzv. *depth sorting*). Tento algoritmus je implementovaný pomocou knižnice VL. Funguje takom princípe, že usporiada pole indexov tak, aby kreslenie bodov prebiehalo v správnom poradí.

Pri zobrazovaní veľkého množstva čiar sa vyskytol jeden problém. Keď je ich hrúbka na obrazovke veľmi malá, nastáva *aliasing* (viď obrázok 6.4 vľavo). Preto sú čiary od určitej vzdialenosti od kamery zobrazované takou konštantnou hrúbkou, aby k tomu javu nedošlo. Aby sa zachoval priestorový dojem vzdialených čiar, je ich alfa kanál vo *vertex shaderi* upravený tak, že sa zmenšuje priamo úmerne vzdialenosti od kamery (viď obrázok 6.4 vpravo).



Obrázok 6.4: Problém s *aliasingom* u tenkých čiar

6.2 Organizácia scény

Organizácia scény korešponduje so spôsobom, aký je použitý v knižnici VL (viď kapitola 4.2). To znamená, že scéna nie je reprezentovaná jedinou štruktúrou (ako napríklad graf scény v OSG). Scénu reprezentujú dve stromové štruktúry: strom objektov (*Actor tree*) a strom transformácií (*transform tree).*

Ako manažér scény je použitý takzvaný strom objektov v scéne (*Actor tree*), ktorého uzly obsahujú objekty triedy *Actor*. Je to podobná štruktúra ako graf scény, ale reprezentuje len jej priestorové usporiadanie (nie sú v nej uzly predstavujúce transformáciu alebo efekty, tie sú v inej hierarchii). Pomocou nej je vo VL implementované orezávanie pomocou pohľadového telesa (*view frustum culling*), aby sa urýchlilo kreslenie scény. Pre každý objekt v scéne je vypočítaný obalový štvorec, ktorého strany sú rovnobežné s hlavnými osami (*axis aligned bounding box*). Ten sa využíva pri výpočte viditeľných objektov v scéne.

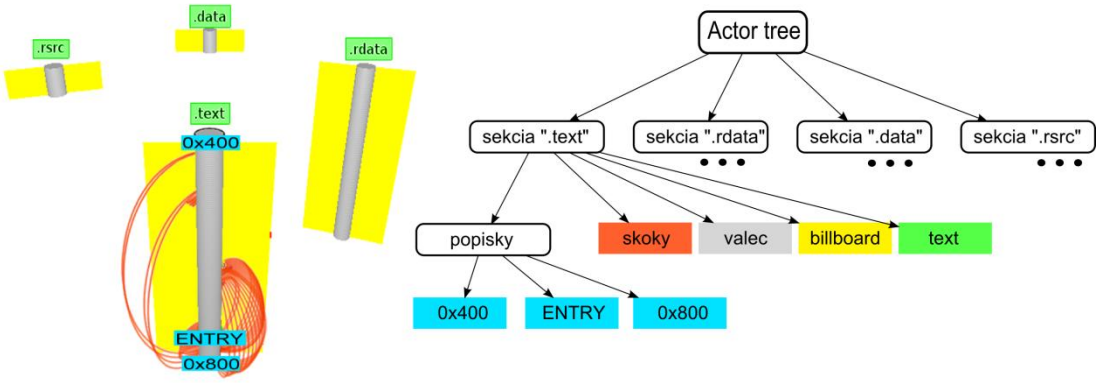
Transformácie aplikované na objekty v scéne sú uložene v transformačnom strome (*transform tree*). Každý jeho uzol obsahuje transformačnú maticu. Uzly sú hierarchicky usporiadané tak, že vzájomným vynásobením im prislúchajúcich transformačných matíc sa získa výsledná poloha objektu v scéne.

Oba vizualizácie (grafy skokov a graf početnosti) sú organizované podobne. Každá sekcia EXE súboru predstavuje v strome objektov jeden samostatný podstrom. V strome transformácií sú hierarchicky uložené transformácie ktoré zabezpečujú posunutie sekcií na ich pozíciu a manipuláciu s nimi. Nasledujúci popis je platí hlavne pre graf skokov, ale mnohé prvky sú prevzaté a použité aj v grafe početnosti.

V grafe skokov je každá sekcia reprezentovaná týmito objektmi (viď obrázok 6.5):

- valec, ktorý reprezentuje sekciu
- skoky, ktorých geometria je uložená ako jeden objekt v spoločnom VBO.
- obdĺžnik natočený stále ku kamere okolo zvislej osi (*billboard*)
- text zobrazujúci názov sekcie
- popisky, ktoré sú ako samostatné objekty usporiadané v jednom podstrome

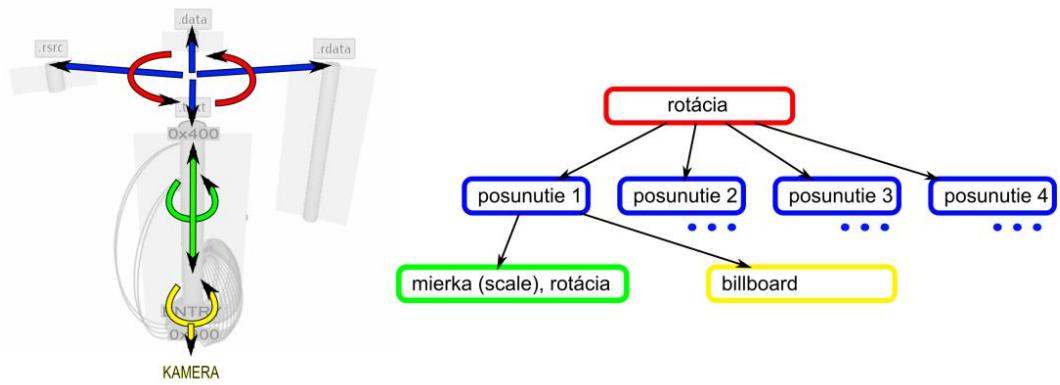
Obdĺžnik (*billboard*) sa využíva na zobrazovanie dodatočných informácií. Napríklad voliteľné zobrazenie početnosti je implementované pomocou textúry, ktorá je na ňom nanesená (textúra sa najprv generuje na CPU pomocou nástroja na kreslenie 2D vektorovej grafiky v knižnici QT). Obdĺžnik nie je vždy viditeľný, ale jeho poloha sa stále využíva pri výpočte prieniku objektov pod kurzorom pri manipulácii s grafom (viď kapitola 6.3).



Obrázok 6.5: Strom objektov (graf scény), vľavo schematické zobrazenie odpovedajúcej scény

Transformačné stromy sú v grafe skokov použité dva. Jeden predstavuje organizáciu objektov v režime zobrazenia všetkých sekcií EXE súboru, druhý v režime zobrazenia jednej sekcie. Každý objekt v scéne má priradený uzol z oboch transformačných stromov. Pri zmene režimu stačí špecifikovať, ktorý transformačný strom sa používa. Tu sa prejavila výhoda oddelenej hierarchie objektov a transformácií, ktorá je použitá v knižnici VL. Na obrázku 6.6 je ukážka transformačného stromu použitého v režime zobrazenia všetkých sekcií.

V transformačnom strome sú viaceré druhy uzlov. Uzol typu *Transform* predstavuje základný typ, ktorý obsahuje transformačnú maticu. Uzol typu *Billboard* prepočítava svoju transformačnú maticu stále tak, aby bol jemu prislúchajúci objekt natočený ku kamere.



Obrázok 6.6: Hierarchia transformácií v scéne

Graf početnosti je organizovaný podobne, ale obsahuje len jeden transformačný strom, keďže má len jeden režim, ktorý zobrazuje všetky sekcie naraz. Jedna sekcie EXE súboru je reprezentovaná dvoma objektmi: skoky a kváder (predstavuje sekciu, je na ňom pomocou textúry zobrazený histogram početnosti).

6.3 Interakcia s užívateľom

Na základnú manipuláciu so scénou sa využíva nástroj *Trackabll* implementovaný v knižnici VL. Funguje tak, že na základe pohybu kurzora, prepočítava pohľadovú transformačnú maticu prislúchajúcu kamere (objekt *Camera*). Aby bol nástroj *Trackball* použiteľný aj v iných režimoch, boli ešte pridané ďalšie funkcie, ako obmedzenie rotácie a posunu na určenú os.

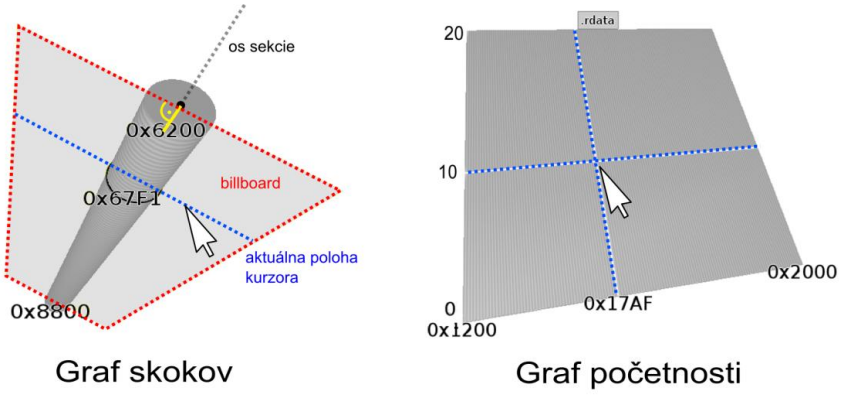
Na prácu s grafmi boli implementované takzvané 3D manipulátory (*3D Widget).* Sú to objekty v scéne, ktoré umožňujú zobraziť dodatočné informácie a s grafom nejakým spôsobom manipulovať. Každý pozostáva z dvoch druhov objektov:

- viditeľné objekty (sú zobrazované v scéne)
- pomocné objekty (určujú polohu)

Toto rozdelenie je z dôvodu urýchlenia výpočtu priesečníka s objektmi nachádzajúcimi sa pod kurzorom myši. Viditeľné objekty (napríklad valce znázorňujúce sekcie v grafe skokov) obsahujú kvôli kvalitnému zobrazeniu podrobnejšiu geometriu. Pre niektoré je v strome objektov (*Actor tree*) uložený pomocný (zástupný) objekt so zjednodušenou geometriou, ktorý sa používa na výpočet priesečníka. Výpočet priesečníka je ďalej urýchlený pomocou obalových telies, ktoré má vypočítané každý objekt v scéne (*Actor*).

Mnohé manipulátory v scéne zdieľajú pomocné objekty. Napríklad *billboard* a valec predstavujúci sekciu sa využívajú na výpočet aktuálnej adresy vo všetkých manipulátoroch v grafe skokov. Preto sú pomocné objekty spravované samostatnou triedou, ktorá vypočíta prienik ich telies s kurzorom a táto hodnota je predaná všetkým manipulátorom, ktoré túto hodnotu potrebujú.

Prvý z manipulátorov je nástroj zobrazujúci informácie o sekcii EXE súboru v grafe skokov. Zobrazuje začiatočnú a koncovú adresu, *entry point* a akruálnu adresu nachádzajúcu sa pod kurzorom (viď obrázok 6.7 vľavo). Medzi viditeľné objekty ktoré spravuje patrí aj valec reprezentujúci sekciu. Každá informácia je zobrazená pomocou textu a kruhovej značky na sekcii (voliteľné). V strome objektov má tento manipulátor vlastný podstrom (viď obrázok 6.5, uzol *popisky*), ktorého listy predstavujú objekty zobrazujúce text a značky. Text je renderovaný pomocou knižnice VL.



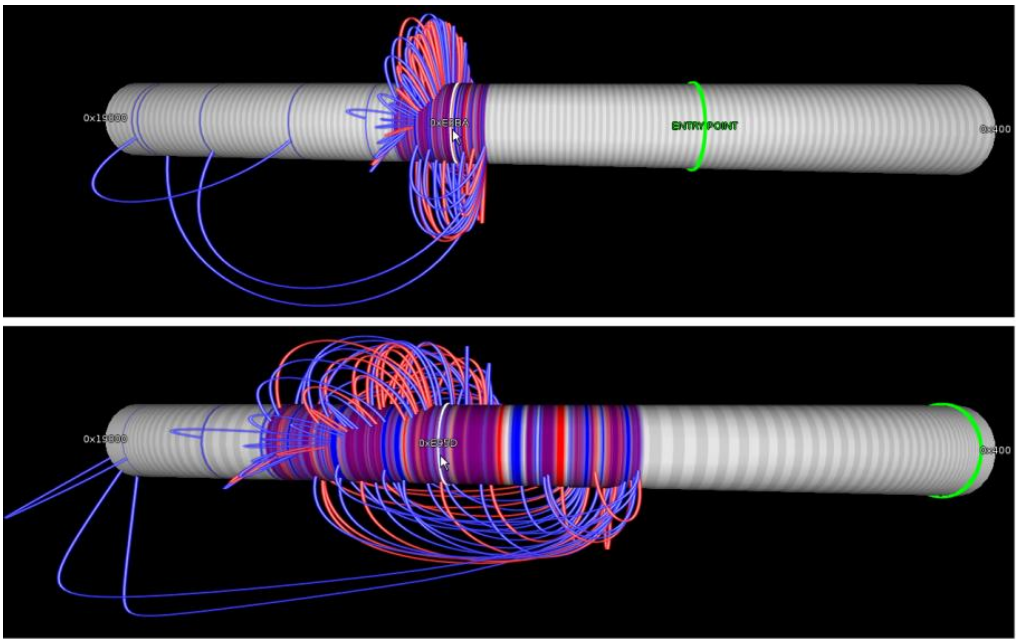
Obrázok 6.7: Schéma manipulátorov zobrazujúcich textové informácie

Podobný manipulátor zobrazujúci informácie je implementovaný aj v grafe početnosti (viď obrázok 6.7 vpravo). Medzi viditeľné objekty ktoré spravuje, patrí kváder reprezentujúci sekciu, popisky a značky, ktoré sú na ňom znázornené. Zobrazuje vždy aktuálnu adresu a hodnotu početnosti, ktorá sa nachádza na polohe pod kurzorom myši. Podobne ako v grafe skokov, je aj tu kváder zastúpený pri určovaní polohy pomocným objektom (ten, ktorý je viditeľný, pozostáva z viacero bodov kvôli lepšiemu osvetleniu).

Pri manipulácii so scénou sa môže stať, že sa textové položky navzájom prekrývajú (napríklad keď je *entry point* hneď na začiatku, prekrýva sa so začiatočnou adresou). Kvôli tomu je implementovaný algoritmus riadiaci viditeľnosť textových objektov tak, že z navzájom prekrývajúcich sa položiek vidno vždy len jednu. Výpočet je realizovaný na základe detekcie prieniku obalových telies.

V grafe skokov je text je zobrazovaný po obvode valca tak, aby bol orientovaný stále k pozorovateľovi. Na určenie polohy textu sa využíva transformácia počítajúca *billboard* uložená v transformačnom strome. Obdĺžnik predstavujúci *billboard* sa využíva ako pomocný objekt na výpočet priesečníka určujúceho pozíciu aktuálnej adresy pod kurzorom. Takto nemusí užívateľ pri manipulácii so sekciou na zistenie aktuálnej adresy prejsť kurzorom presne nad úzky valec, ale stačí, že je v oblasti nad *billboardom* (viď obrázok 6.7 vľavo).

Ďalší manipulátor umožňuje v grafe skokov natočiť sekciu v tvare valca tak, aby bol skok začínajúci na adrese pod kurzorom orientovaný presne k pozorovateľovi. Je to užitočné v prípadoch, keď je potrebné rýchlo na aktuálnej pozícii zobraziť skoky, ktoré môžu byť inak zakryté telesom valca, keďže sú rozmiestnené po jeho obvode. Na určenie aktuálnej polohy sa využíva tiež *billboard* ako pomocný objekt. Na základe adresy sa binárnym vyhľadávaním nájde skok, ktorého adresa je najbližšie. Potom sa vypočíta transformačná matica kontrolujúca natočenie sekcie tak, aby bol daný skok smerovaný ku kamere.



Obrázok 6.8: Priblíženie pomocou efektu "rybie oko"

Posledný manipulátor umožňuje deformovať zobrazenie sekcie pomocou efektu rybie oko (*fish eye*, viď obrázok 6.8). Výpočet deformácie vychádza z rovnice 6.2 pre súdkovité skreslenie (*barrel distortion*), kde x je originálny bod obrazu a x' je súradnica v skreslenom obraze. Parametre k_i určujú mieru skreslenia, r je vzdialenosť originálneho bodu od stredu obrazu [5].

$$x' = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{6.2}$$

Vzťah 6.2 sa používa v spracovaní obrazu, kde sa počíta pre obe dimenzie (*x* aj *y*). V tomto prípade sa skreslenie počíta len pre jednu dimenziu (obraz sa deformuje len pozdĺž osi sekcie EXE súboru). Efekt je implementovaný pomocou *vertex shaderu*. Výpočet prebieha v objektovom priestore (*object space*) ešte pred transformáciou *modelview* maticou. Vstup je posunutá pozícia bodu. Súradnica stredu obrazu použitá vo výpočte je určená aktuálnou pozíciou kurzora vzhľadom na os sekcie. Parametre určujúce pozíciu stredu a mieru skreslenia sú predávané do *vertex shaderu* pomocou premenných typu *uniform*.

6.4 Načítanie EXE súboru a inštrukcií

Načítanie EXE súboru má na starosti samostatný modul, ktorý využíva štruktúry z hlavičiek *Windows.h*. Najprv sa načíta celý súbor do pamäte. Následne sa zistí poloha hlavičky súboru a voliteľnej hlavičky (*IMAGE_OPTIONAL_HEADER*). Na základe údajov z týchto štruktúr sa vypočíta poloha tabuľky sekcií a zisťujú sa informácie o jednotlivých sekciách. Pri načítavaní sú zisťované nasledujúce údaje, ktoré sa neskôr využívajú pri načítaní inštrukcií skokov a zobrazovaní vizualizácie:

- Hodnota *Image base*, ktorá sa využíva pri adresovaní v inštrukciách.
- Hodnota *Entry point* (vstupný bod), ktorá je zobrazená vo vizualizácii.
- Informácie o sekciách EXE súboru (RAW adresa *PointerToRawData*, RVA adresa *VirtualAddress*, veľkosť sekcie *SizeOfRawData*). RAW adresa je potrebná na samotnú lokalizáciu sekcie v binárnom súbore, RVA adresa sekcie sa využíva pri výpočte adries ostatných štruktúr v sekcii, ktoré sú špecifikované RVA adresami.
- Informácie a štruktúrach *IMAGE_DATA_DIRECTORY*.

Pri načítavaní z EXE súboru sa číta z rôznych adries v ňom špecifikovaných. Pri súboroch infikovaných počítačovými vírusmi však môže dôjsť k tomu, že sa nejaká informácia poškodí a hodnota je neplatná. Napríklad RAW adresa sekcie (ofset od začiatku súboru) môže odkazovať napríklad úplne mimo EXE súbor. Aby sa program v týchto prípadoch správal korektne, je potrebné vždy kontrolovať RAW adresy, či nie sú väčšie ako veľkosť EXE súboru. Aby sa táto činnosť automatizovala, je implementovaná trieda, ktorá sa používa na prístup do pamäte k načítanému súboru podobným spôsobom, ako sa používajú v C++ napríklad chytré ukazatele (*smart pointer)*. Pri vytváraní tohto ukazateľa sa špecifikuje veľkosť súboru v pamäti a pri prístupe do pamäte pomocou RAW adresy sú vykonávané automatické kontroly. Keď je adresa pri načítaní nejakej položky neplatná tak sa nepoužije a položka sa nezobrazí.

Po získaní všetkých potrebných informácií o EXE súbore sa načítava binárny kód programu. Najprv sa vyhľadáva v sekcii obsahujúcej *entry point*, ale hľadá sa aj v iných sekciách, pretože pri infikovaných súboroch sa môže nachádzať aj inde. Na analýzu inštrukcií programu sa používa knižnica vytvorená v spoločnosti AVG. Tá obsahuje funkciu, ktorej vstup je adresa v pamäti a výstup nájdená inštrukcia a jej veľkosť, prípadne chybový kód. K tomu bolo potrebné implementovať logiku, ktorá má na starosti nájdenie začiatku binárneho kódu (miesto, kde sa nachádzajú inštrukcie).

Pri načítavaní inštrukcií sa najprv začína na začiatku sekcie. Keď je inštrukcia platná, získa sa jej veľkosť a po jej spracovaní sa pokračuje na adrese za danou inštrukciou. Ak je inštrukcia neplatná, tak sa na danom mieste nenachádza binárny kód alebo bol zadaný nesprávny posun (*offset*). V tom

prípade sa ďalšia inštrukcia hľadá na adrese o bajt ďalej až kým algoritmus nenarazí na platnú inštrukciu.

Keď sa podarí načítať inštrukciu, zisťuje sa, či sa jedná o inštrukciu skoku, ktorá je zobrazená vo vizualizácii (*relative near jump*). Ak áno, vykoná sa ešte kontrola, či *offset* v inštrukcii smeruje na adresu v rámci súboru. Ďalej sa kontroluje, či sa na cieľovej adrese kam smeruje skok, nachádza ľubovoľná platná inštrukcia. Ak nie sú oba tieto podmienky splnené, pravdepodobne sa inštrukcia načítala nesprávne (napríklad sa načítali iné dáta, ktoré sú zhodné s operačným kódom inštrukcie). V tom prípade sa pokračuje v hľadaní spôsobom, ako keď sa nájde neplatná inštrukcia.

Pri zvolenom spôsobe načítania sa môže stať, že sa omylom načítajú inštrukcie skokov, ktoré sú v skutočnosti náhodné dáta (ak súhlasí operačný kód a nasledujúci *offset* má rozumnú hodnotu, viď vyššie). Pri testovaní bolo však zistené, že sa týchto prípadov vyskytuje veľmi málo a len vo veľkých súboroch, kde sa vo veľkom počte skokov zopár chybných stratí (pomer býva napríklad 1 chyba na 100 skokov).

6.5 GUI a konfigurácia programu

Prvky grafického užívateľského rozhrania (ako menu, panel nástrojov a podobne) sú implementované pomocou štandardných prostriedkov, ktoré na to poskytuje knižnica QT. Scéna s 3D grafom je zobrazená pomocou špeciálneho prvku (*QGLWidget*), ktorý je na to v QT určený. *OpenGl* kontext je najprv vytvorený pomocou knižnice QT, ale neskôr spravovaný pomocou knižnice VL, ktorá má na starosti samotné vykresľovanie scény.

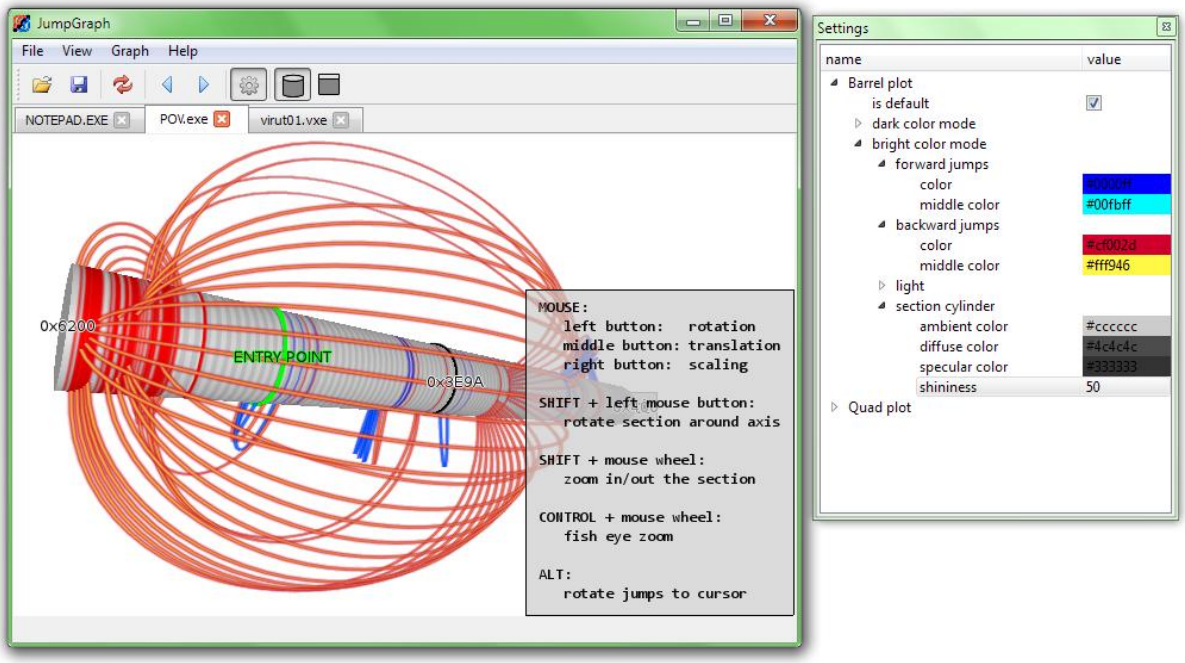
Pri práci s viacerými súbormi naraz sa vytvára viacero okien vykresľujúcich scénu pomocou *OpenGL*. Tie zobrazujú jednotlivé grafy, v ktorých sú použité podobné textúry a objekty programov v jazyku GLSL (*vertex* a *fragment shader*). Aby sa zabránilo plytvaniu pamäťových prostriedkov, využíva sa v programe zdieľanie *OpenGl* kontextu (takzvaný *OpenGL context sharing*). Pri vytváraní prvého grafu sa vytvorí najprv nový *OpenGL* kontext a inicializujú sa všetky spoločné prostriedky (textúry a podobne). Pri vytváraní ďalšieho grafu sa vytvára *OpenGl* kontext, ktorý je zdieľaný s predošlým, pričom sa znovu použijú pred tým vytvorené prostriedky, ktoré má prvý a druhú graf spoločné.

Jedna z požiadaviek na aplikáciu bola, aby pozostávala pokiaľ možno len z jedného spustiteľného súboru kvôli jednoduchej distribúcii. Program ale využíva pri svojom behu mnoho dátových súborov (textúry, ikony, texty obsahujúce GLSL programy a podobne). Aby nebolo potrebné tieto zdroje načítavať za súboru, sú zakompilované do spustiteľného súboru pomocou systému *Resources* v knižnici QT. Aby sa uľahčilo ladenie programu pri vývoji aplikácie, je využívaný na načítavanie týchto zdrojov virtuálny súborový systém, ktorý je k dispozícii v knižnici VL. Ten umožňuje prístup rovnakým spôsobom k súborom na disku aj v pamäti. Vo verzii na ladenie sú do neho namapované reálne adresáre na disku. Vo finálnej verzii na distribúciu sú do virtuálneho súborového systému namapované zdroje zakompilované v spustiteľnom súbore (ktoré sa v čase behu programu nachádzajú v pamäti).

Vo výslednej vizualizácii sa používajú rôzne farby. Na základe užívateľskej spätnej väzby sa zistilo, že by bolo vhodné mať v programe možnosť takéto vlastnosti meniť. Preto bola do GUI pridaná možnosť nastaviť vybrané parametre vizualizácie (väčšinou sú to farby a číselné hodnoty). Keďže je týchto parametrov veľa, bola zvolená na ich organizáciu hierarchická stromová štruktúra (takzvaný strom nastavení). Uzly stromu pozostávajú z kľúča (reťazec predstavujúci názov nastavenia) a hodnoty, ktorá je implementovaná ako variantný záznam. Na ukladanie nastavení programu bol zvolený súbor v jazyku XML. Na implementáciu týchto prostriedkov som využil knižnicu *Boost* (kontajnery *Boost.PropertyTree* a *Boost.Variant*). V grafickom užívateľskom rozhraní je strom s nastaveniami zobrazený pomocou prvku *QTreeView*. Ten umožňuje vykresliť stromovú

štruktúru nastavení a poskytuje grafické prvky na interaktívnu zmenu ich hodnôt, ktoré môžu byť rôznych typov (napríklad pre farbu sa zobrazí zafarbený štvorec, pre číslo sa zobrazí editovateľný prvok a podobne).

Na obrázku 6.9 je ukážka grafického užívateľského rozhrania. Naľavo sa nachádza hlavné okno programu, napravo je ukážka editora nastavení. V grafe je demonštrované voliteľné zobrazenie rýchlej pomoci.



Obrázok 6.9: Grafické užívateľské rozhranie

7 Dosiahnuté výsledky

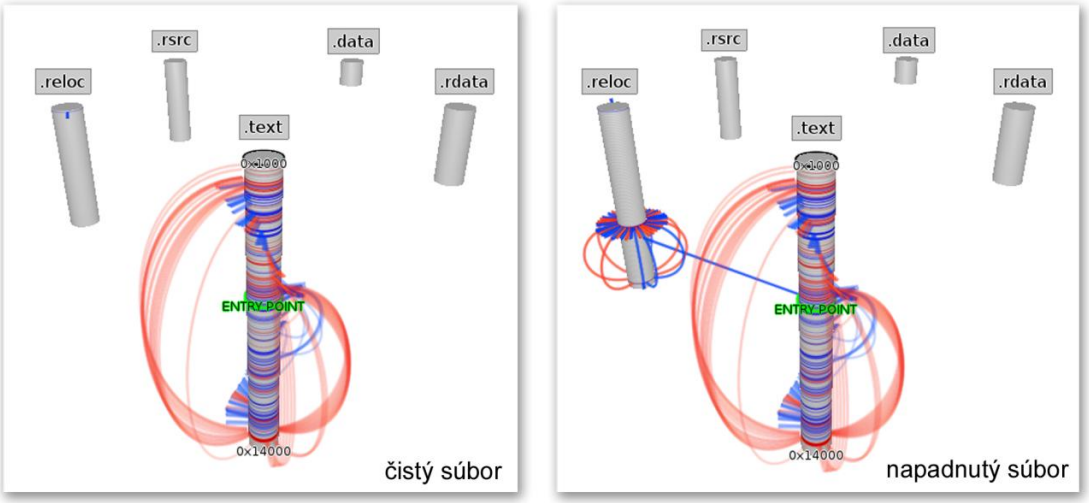
Vizualizácia programového toku bola navrhnutá tak, aby bolo možné na základe výskytu skokov v programe lokalizovať podozrivé miesta, kde by sa mohol nachádzať kód malware. Vytvorené grafy umožňujú získať najprv rýchly prehľad o štruktúre zobrazeného PE EXE súboru. V jednotlivých sekciách je možné potom lokalizovať programový kód a podľa jeho štruktúry určiť, či sa jedná o kód produkovaný štandardnými kompilátormi, alebo niečo podozrivé.

V tejto kapitole je najprv pomocou vhodných príkladov demonštrovaná užitočnosť vytvorenej vizualizácie pri analýze malware. Kvôli kontinuite sa ukážky zameriavajú na graf skokov (zobrazenie pomocou druhého grafu je možné nahliadnuť v prílohe C). Na záver je popísané užívateľské testovanie, ktoré momentálne prebieha v spoločnosti AVG.

7.1 Ukážky vybraných vzoriek malware

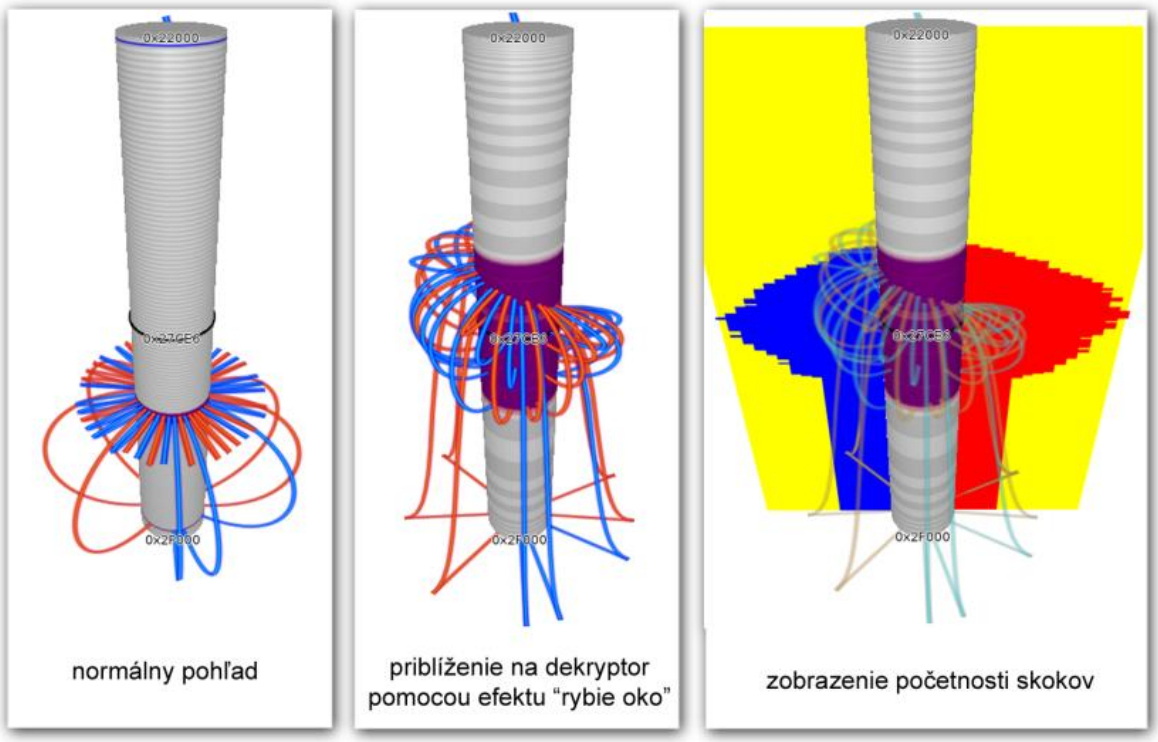
Funkčnosť vizualizácie sa dá demonštrovať napríklad na prípade polymorfného vírusu *W32/Virut* (bol spomínaný už v kapitole 3.4, podrobný popis viď [1]). Napáda spustiteľné programy na 32 bitových platformách Windows s príponou *EXE* alebo *SCR*. Pri infekcii najčastejšie pripojí svoje zakódované telo vrátane dekryptoru na koniec súboru do poslednej sekcie. Vírus buď zmení vstupný bod programu (*entry point*) tak, aby smeroval na dekryptovací kód v tejto sekcii (ukážka v kapitole 3.4), alebo využíva EPO techniky. Napríklad vyhľadáva od vstupného bodu programu najbližšie volanie inštrukcie *CALL* smerujúce na *kernel32.dll*, ktoré zmení na skok na vlastné telo. Po ukončení svojej činnosti predá riadenie pôvodnému programu na mieste, kde ho prevzal.

Na obrázku 7.1 je naľavo zobrazený pôvodný súbor (jedná sa o *ExportController.exe*, súčasť balíka *QuickTime*) a napravo súbor napadnutý vírusom *W32/Virut* pomocou grafu skokov. Je zvolený režim zobrazenia všetkých sekcií EXE súboru, kde vidno aj skoky medzi sekciami. Vstupný bod programu je vyznačený zelenou farbou pomocou popisu (*entry point*). Sekcia *text* obsahuje programový kód. Vírus v tomto prípade použil EPO techniku a namiesto priamej zmeny hodnoty vstupného bodu programu vložil kúsok zaň inštrukciu skoku na svoj dekryptor. Ten sa nachádza na konci sekcie *reloc*, kde sa normálne vyskytuje tabuľka relokácií. Vo vizualizácii je inštrukcia skoku z napadnutého programu na dekryptor vírusu znázornená ako čiara spájajúca sekciu *text* v blízkosti vstupného bodu a sekciu *reloc*.



Obrázok 7.1: Súbor napadnutý vírusom *W32/Virut*

Pri pohľade na štruktúru skokov sa dá rozoznať dekryptor vírusu (krátky prstenec nahustených skokov náhodne v oboch smeroch). Keďže dekryptor vírusu je v tomto prípade krátka rutina, bolo by vhodné mať možnosť preskúmať podrobnejšie len túto časť súboru. Na to je vhodný druhý režim zobrazenia jednej sekcie v grafe skokov a nástroj na priblíženie pomocou efektu rybie oko. Na obrázku 7.2 je znázornená sekcia *reloc*, v ktorej sa nachádza dekryptor. Napravo je celkový pohľad na sekciu a v strede je ukážka priblíženia pomocou efektu rybie oko tak, aby bolo vidno štruktúru v tele dekryptora. Po priblížení sa dá na základe rôznorodej dĺžky skokov rozoznať nepravidelná štruktúra programového toku v dekryptore. Ich farebné odlíšenie zvýrazňuje to, že sa dva typy skokov náhodne striedajú. Oproti tomu štruktúra skokov pôvodného programu v sekcii *text* pozostáva zo zhlukov, ktoré obsahujú skoky rovnakého typu, čo je typické pre normálny kód (viď obrázku 7.1). Na obrázku 7.2 napravo je znázornené voliteľné zobrazenie početnosti skokov. Kvôli väčšej zrozumiteľnosti sú skoky v tomto režime zobrazenia znázornené menej výrazne, aby sa farba histogramu znázorňujúceho početnosť nekryla s farbou skokov.

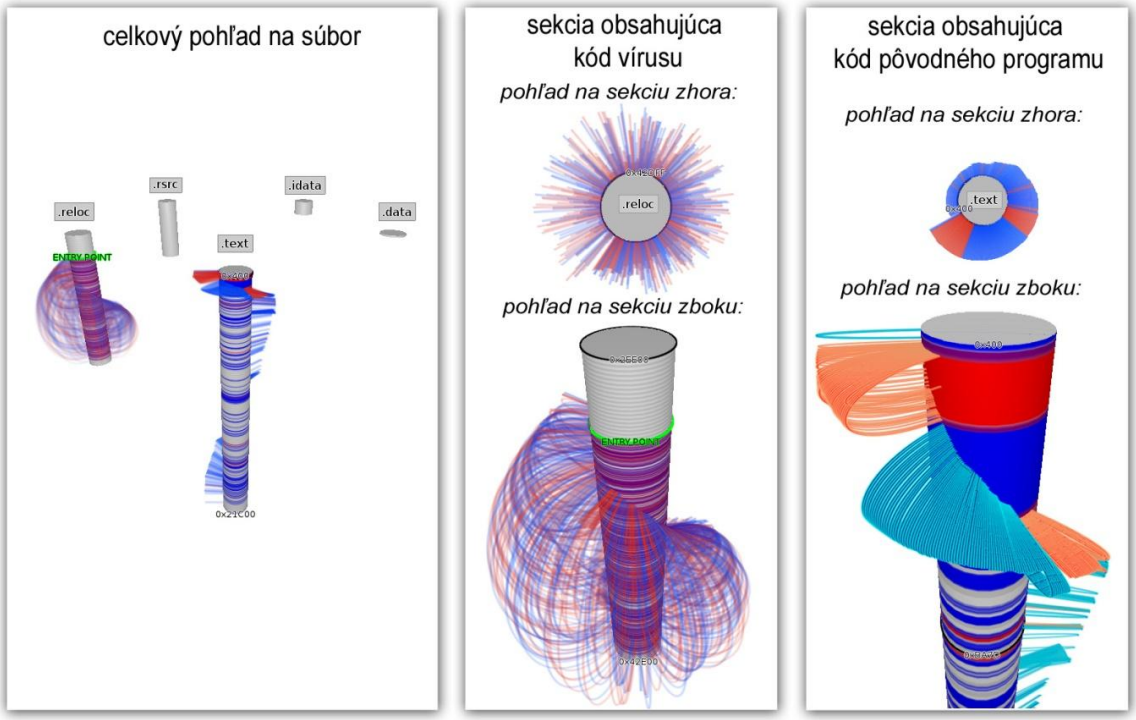


Obrázok 7.2: Súbor napadnutý vírusom *W32/Virut*, sekcia *.reloc* obsahujúca jeho dekryptor

Ďalší vhodný príklad je metamorfný vírus *W32/Bistro*, ktorý v princípe vychádza z vírusu *W95/ZPerm* (spomínaný v kapitole 3.4). Podobne ako on využíva metódy zatemňovania kódu (*obfuscation*). Svoj programový kód delí na malé úseky, ktoré sú náhodne permutované (poprehadzované). Tie pospája inštrukciou skoku. Okrem toho využíva metódu, ktorá je určená na znemožnenie detekcie pomocou emulátora tak, že pred dekryptor vkladá nič nerobiace inštrukcie a dlhé cykly (podrobný popis vírusov *ZPerm* a *Bistro* viď [9] a [30]).

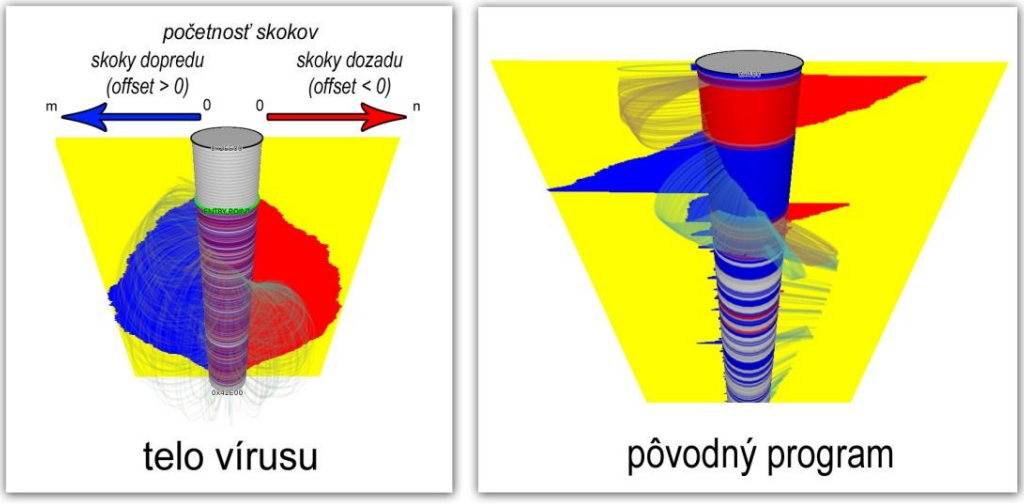
Na príklade súboru napadnutého vírusom *W32/Bistro* sa dá demonštrovať rozdiel medzi štruktúrou normálneho programového kódu a obfuskovaného kódu vírusu (viď obrázok 7.3). Zobrazený súbor je *explorer.exe* (súčasť operačného systému Windows 98). Sekcia *text* obsahuje pôvodný program. Pri pohľadu na túto sekciu zhora aj zboku sa dajú rozoznať zhluky skokov rovnakého smeru s pravidelnou štruktúrou. Oproti tomu kód vírusu, ktorý sa nachádza v sekcii *reloc*,

obsahuje veľké množstvo náhodne usporiadaných skokov v oboch smeroch s rôznou dĺžkou. Farebné odlíšenie skokov vo vizualizácii napomáha rozpoznať takýto obfuskovaný kód. Tým, že sa skoky náhodne striedajú, mixujú sa ich farby (červená a modrá) do fialovej. Pri pohľade zhora na sekciu *reloc* sa dá ich náhodná štruktúra rozpoznať aj na základe ich rôznorodej dĺžky.



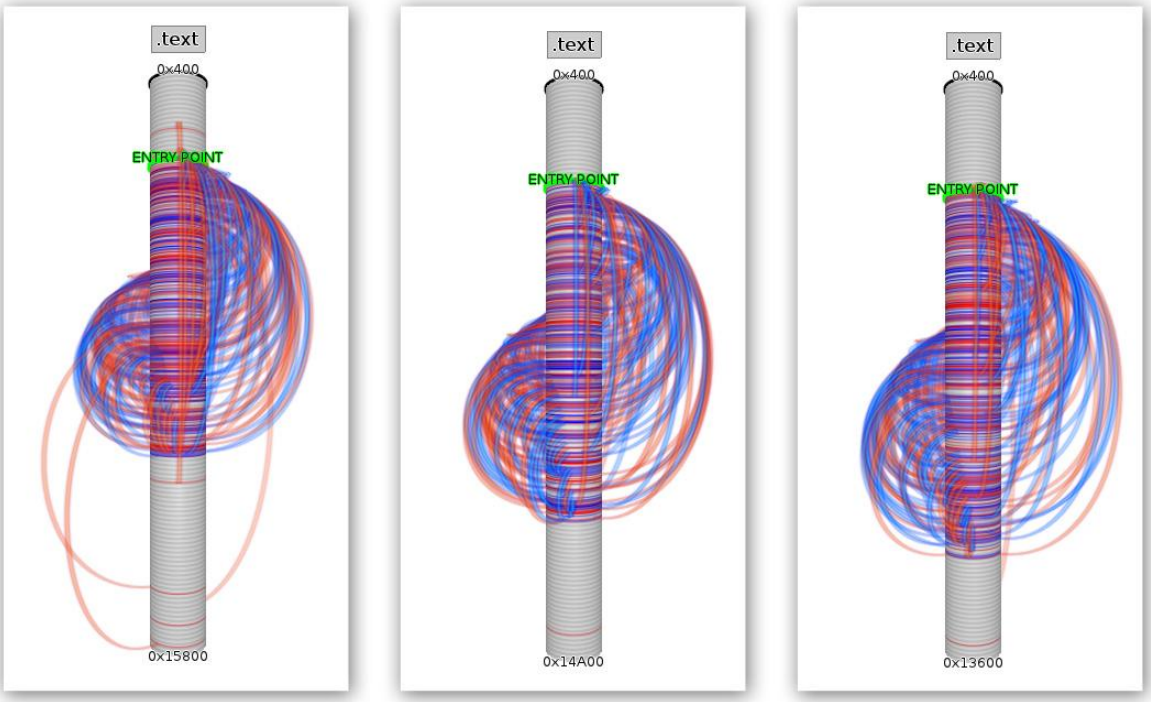
Obrázok 7.3: Súbor napadnutý vírusom *W32/Bistro*

Na obrázku obrázok 7.4 je zobrazená početnosť skokov sekcie obsahujúcej telo vírusu a sekcie obsahujúcej pôvodný program. V tele vírusu ktorá stúpa pre oba typy skokov ich početnosť plynule k jeho stredu. Fakt, že sa vírus snaží náhodne generovať skoky dopredu aj dozadu spôsobuje, že je ich početnosť v oboch smeroch rovnaká. U skokov v normálnom programovom kóde, ktoré majú pravidelnejšiu štruktúru, sa však ich početnosť častejšie prudko mení a býva pre skoky dopredu a dozadu odlišná na danej adrese (viď obrázok 7.4 vpravo).



Obrázok 7.4: Súbor napadnutý vírusom *W32/Bistro* – zobrazenie početnosti skokov

Dosiaľ spomínané príklady patria do jednej rodiny malware (vírusy). Podobné techniky obfuskácie kódu sú aplikované aj u niektorých trójskych koňov. Pomocou vizualizácie skokov sa dá dobre identifikovať napríklad *Backdoor.Cycbot*. Tento trójsky kôň sa šíri tak, že je generovaný na strane serveru, a k užívateľovi sa dostane vždy binárne unikátny variant napríklad pomocou podvodnej webovej stránky. Aby toho dosiahol, využíva podobne ako metamorfný vírus *W32/Bistro* zmenu programového toku pomocou inštrukcie skokov. Na rozdiel od predošlého vírusu však nenapáda existujúce súbory a jeho programový kód sa nachádza hneď v prvej sekcii. Na obrázku 7.5 sú znázornené tri binárne odlišné vzorky, ktoré sa však pri zobrazení pomocou vizualizácie skokov navzájom podobajú.



Obrázok 7.5: Vzorky trójskeho koňa *Backdoor.Cycbot*, zobrazená vždy len prvá sekcia

7.2 Testovanie

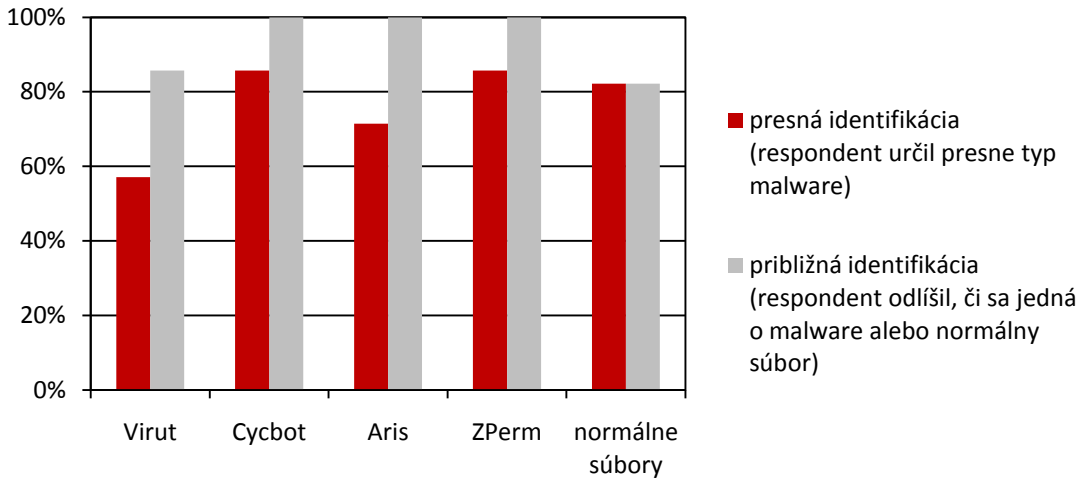
Nástroj na vizualizáciu programového toku je testovaný v spolupráci s antivírusovými výskumníkmi v spoločnosti AVG. Pre účely testovania bol vytvorený dotazník, pomocou ktorého sa snažím zistiť, či je vytvorený nástroj nápomocný pri rozoznávaní rôznych druhov malware od normálneho programového kódu.

Dotazník je rozdelený na dve časti. Prvá časť pozostáva z praktického testovania. Respondent má za úlohu rozoznávať rôzne spustiteľné súbory na základe vizualizácie. Najprv si môže prezrieť ukážkové súbory. Medzi nimi sú zaradené pomenované ukážky štyroch druhov malware, každý po troch vzorkách. Sú to vírusy *W95/ZPerm*, *W32/Virut*, *W32/Aris* a trójsky kôň *Backdoor.Cycbot*. Ďalej sú medzi ukážkovými súbormi zaradené spustiteľné súbory normálnych aplikácií (napríklad *Skype*, *regedit.exe* a podobne). Následne má respondent identifikovať 8 rôznych súborov. Sú medzi nimi spomínané vzorky malware (každý jeden krát) a 4 normálne súbory. U každého neznámeho súboru sú nasledujúce možnosti odpovede:

- nedá sa určiť o aký súbor sa jedná
- normálny súbor

- podozrivý súbor
- konkrétny druh malware (tu si respondent môže vybrať zo spomínaných 4 druhov)

Výsledky tejto časti sú zobrazené na obrázku 7.6 pomocou grafu znázorňujúceho percentá respondentov, ktorý správne určili daný druh EXE súboru. Ako presná identifikácia je označený prípad, keď respondent označil neinfikovaný súbor správne ako normálny, alebo ak sa jednalo o vzorku malware, tak určil presne konkrétny typ. Ako približná identifikácia sa myslí prípad, keď respondent označil vzorku malware aspoň ako nejaký druh malware alebo ako podozrivý súbor. V grafe vidno, že na základe vizualizácie je možné celkom presne odlíšiť malware od normálnych súborov. Pomocou grafov sa tiež darí identifikovať rozdielne vzorky malware.



Obrázok 7.6: Úspešnosť identifikácie malware pomocou vizualizácie skokov v programe

Druhá časť dotazníka rieši názor respondenta na oba druhy vizualizácií, ktoré môže nástroj zobraziť (graf skokov a graf početnosti). Obsahuje otázky, v ktorých môže respondent zhodnotiť zrozumiteľnosť oboch grafov a prínos implementovaných funkcií, ktoré uľahčujú manipuláciu s nimi. Počet otázok bol volený čo najnižší, aby sa zachovala stručnosť dotazníka (vzhľadom na to, že prvá časť vyžaduje veľa času). Presné znenie dotazníka a jeho podrobné vyhodnotenie sú v prílohe A.

Dotazník vyplnilo zatiaľ len 7 respondentov, keďže program je v čase odovzdania tejto práce ešte stále v štádiu testovania v spoločnosti AVG. Z dostupných výsledkov vyšlo najavo, že graf skokov je aj vďaka svojej prepracovanosti a väčšej zrozumiteľnosti kladnejšie hodnotený ako graf početnosti. Kladne bola hodnotená najmä funkcia priblíženia pomocou efektu rybie oko, pretože mnohé druhy malware obsahujú veľmi krátku pre analýzu zaujímavú časť programového kódu v porovnaní s celou sekciou (viď napríklad ukážka dekryptoru vírusu *W32/Virut* v predošlej kapitole). Z ďalších získaných komentárov a poznámok k programu vysvitlo, že užívateľom chýbala možnosť zobrazenia programového toku v spustiteľných súboroch, ktoré sú skomprimované nástrojom ako napríklad UPX. Táto funkcia by sa dala v rámci nasledujúceho rozšírenia implementovať. Veľmi kladne bola hodnotená estetická stránka vizualizácie, ktorá umožňuje použiť grafy ako vhodný doplnok na konferenciách a prezentáciách aj pre laickú verejnosť.

8 Záver

V rámci diplomovej práce boli naštudované metódy analýzy programového toku a prostriedky na vizualizáciu dát. Cieľom práce bolo navrhnúť a implementovať vizualizáciu, ktorá by túto analýzu antivírusovým výskumníkom uľahčila. Cieľ bol splnený. Výsledkom práce je nástroj, ktorý umožní pomocou 3D grafu zobraziť štruktúru a početnosť skokov v programovom kóde tak, že je v ňom možné identifikovať podozrivé znaky malware.

Na začiatku boli naštudované potrebné prerekvizity spojené s analýzou malware (formát spustiteľných súborov a inštrukcií) a rôzne metódy jeho detekcie, špeciálne analýza programového toku. Potom boli predstavené súčasné metódy vizualizácie dát malware a prostriedky, ktoré by sa hodili na ich 3D vizualizáciu. Následne boli demonštrované dva návrhy 3D vizualizácie, ktoré zobrazujú skoky v programe. Ako vhodné riešenie sa ukázalo využiť pri implementácii pokročilé techniky akcelerácie 3D grafiky na hardvéri. Skoky sú reprezentované pomocou efektu falošných volumetrických čiar, ktorých geometria sa počíta na GPU pomocou *vertex shaderu*. Na organizáciu scény sú využité stromové štruktúry podobné grafu scény, ktoré umožňujú reprezentovať oddelene hierarchiu objektov a hierarchiu transformácií. Na zvýšenie užívateľského komfortu boli pridané funkcie uľahčujúce manipuláciu s grafom, ako napríklad priblíženie pomocou efektu „rybie oko“. V rámci rozšírenia bol implementovaný okrem základného typu grafu zameriavajúceho sa na zobrazenie štruktúry skokov graf zobrazujúci ich početnosť. Vytvorený program je testovaný antivírusovými výskumníkmi v spoločnosti AVG.

Výsledkom práce je nástroj, pomocou ktorého je možné rozlišovať podozrivé znaky malware na základe štruktúry skokov v programovom kóde. Doterajším testovaním v spoločnosti AVG sa zistilo, že nástroj môže dobre poslúžiť nielen pri odlíšení malware od normálnych programov, ale aj na presné identifikovanie, o aký druh malware sa jedná. Vytvorená vizualizácia môže okrem toho nájsť využitie aj pri mediálnej prezentácii tohto typu analýzy pre laickú verejnosť.

V rámci nasledujúceho vývoja by bolo vhodné neostávať len pri statickej analýze binárneho EXE súboru. Zaujímavou možnosťou rozšírenia by bolo napríklad napojenie existujúcej vizualizácie na emulátor behu programu a animovať dynamicky jeho priebeh pomocou skokov. Súčasná vizualizácia by sa mohla rozšíriť aj o zobrazenie iných typov inštrukcií skoku. Na to bude potrebné implementovať pokročilejšiu logiku v použitom disassembleri, ktorá vie lepšie odlíšiť miesta, kde sa nachádza binárny kód od náhodných dátových oblastí. Pri súčasnej implementácii vizualizácie sa myslelo na to, aby mohol program bežať aj na staršom hardvéri. Preto sa pri implementácii nepoužili najnovšie prostriedky akcelerácie grafiky ako napríklad *geometry shader*. V budúcnosti by bolo vhodné využiť tieto možnosti napríklad pri implementácii zobrazovania čiar, kde by nebolo potrebné ukladať v pamäti ich pozície opakovane kvôli výpočtu vo *vertex shaderi*.

Literatúra

[1] Álvarez, V. M.: Beyond Virtu(e) and evil. *Virus Bulletin*, s. 6-8, Máj 2007.

[2] Aycock, J.: *Computer Viruses and Malware*. New York, USA: Springer Science+Business Media, LLC, 2006.

[3] Bonfante, G.; Kaczmarek, M.; Marion, Y.: Control Flow to Detect Malware. In *Inter-Regional Workshop on Rigorous System Development and Analysis*, 2007 [Online]. `http://hal.inria.fr/docs/00/17/62/41/PDF/IRWRSDA.pdf` [cit. 2011-01-09]

[4] Bosi, M.: Visualization Library documentation [Online]. `http://visualizationlibrary.com/documentation/index.html`, 2010 [cit. 2011-01-19]

[5] Bradski, G.; Kaehler, A.: *Learning OpenCV*. Sebastopol, USA: O'Reilly, 2008.

[6] Dragulescu, A.: Malwarez [Online]. `http://www.sq.ro/malwarez.php`, 2008 [cit. 2011-01-03]

[7] Eagle, Ch.: *The IDA Pro Book*. San Francisco, USA: No Start Press, 2008.

[8] Evstiougov-Babaev, A. A.: Call Graph and Control Flow Graph Visualization for Developers of Embedded Applications. In *Revised Lectures on Software Visualization, International Seminar*, London, 2002, s. 337-346.

[9] Fernandes, R. T.; Finones, R. G.: Solving The Metamorphic Puzzle. *Virus Bulletin*, s. 14-19, Marec 2006.

[10] Hansen, Ch. D.; Johnson, Ch. R.: *The Visualization Handbook*. Burlington, USA: ElsevierAcademic Press, 2005.

[11] Hillaire, S.: Volumetric lines [Online]. `http://sebastien.hillaire.free.fr/demos/simplevolumeline/vline.htm`, 2011 [cit. 2011-03-29]

[12] Hyde, R.: *The art of assembly*. San Francisco, USA: No Starch Press, 2003.

[13] Intel Corporation: *Intel® 64 and IA-32 Architectures Software Developer's Manual*. 2010.

[14] Kitware, Inc.: VTK - The Visualization Toolkit [Online]. `www.vtk.org`, 2009 [cit. 2011-01-03]

[15] Kolektív autorov: OpenDX Home Page [Online]. `http://www.opendx.org`, 2011 [cit. 2011-04-02]

[16] Kolektív autorov: Processing [Online]. `http://processing.org`, 2011 [cit. 2011-04-02]

[17] Lorach, T.: Fake Volumetric lines. Technická správa, NVIDIA Corporation, 2005.

[18] Martz, P.: *OpenSceneGraph Quick Start Guide*. Louisville, USA: Skew Matrix Software LLC, 2007.

[19] Microsoft Corporation: *Microsoft Portable Executable and Common Object File Format Specification, Revision 8.2*. 2010 [Online]. http://www.microsoft.com/whdc/system/platform/firmware/pecoff.mspx [cit. 2010-12-15]

[20] Pietrek, M.: An In-Depth Look into the Win32 Portable Executable File Format [Online]. http://msdn.microsoft.com/sk-sk/magazine/cc301805(en-us).aspx, 2002 [cit. 2010-12-05]

[21] Pietrek, M.: An In-Depth Look into the Win32 Portable Executable File Format, Part 2 [Online]. http://msdn.microsoft.com/sk-sk/magazine/cc301808(en-us).aspx, 2002 [cit. 2010-12-05]

[22] Pietrek, M.: Peering Inside the PE: A Tour of the Win32 Portable Executable File Format [Online]. http://msdn.microsoft.com/sk-sk/magazine/ms809762(en-us).aspx, 1994 [cit. 2010-12-05]

[23] Quist, Q. A.; Liebrock, L. M.: Visualizing Compiled Executables for Malware Analysis. In *Vizsec 2009*, Atlantic City, USA, 2009.

[24] Rusnák, J.: Grafická prezentace výsledků statistické analýzy PE EXE. Brno, 2008, bakalárska práca, FIT VUT v Brne.

[26] Schroeder, W. J.: *The VTK User's Guide*. Kitware Inc., 2001.

[27] Schussman, G.; Ma, K.; Schissel, D.; Evans, T.: Visualizing DIII-D Tokamak Magnetic Field Lines. In *Visualization Conference, IEEE*, 2000, s. 56.

[25] Sharp, R.: An Introduction to Malware [Online]. http://www2.imm.dtu.dk/courses/02233/malware.pdf, 2009 [cit. 2010-12-08]

[28] Skoudis, E.; Zeltser, L.: *Malware: Fighting Malicious Code*. Prentice Hall PTR, 2003.

[29] Swanson, I.: Malware, Viruses and Log Visualisation. In *Proceedings of The 6th Australian Digital Forensics Conference*, Edith Cowan University, Perth, Western Australia, 2008, s. 181-190.

[30] Szor, P.: *Počítačové viry, analýza útoku a obrana*. Brno: Zoner Press, 2006.

Zoznam príloh

Príloha A

Užívateľské hodnotenie programu

Program je hodnotený antivírusovými výskumníkmi v spoločnosti AVG. Hodnotenie prebieha pomocou dotazníka, kde sa najprv zisťuje úspešnosť, s ktorou je možné rozoznávať druhy malware. Druhá časť dotazníka obsahuje otázky týkajúce sa ovládania a zrozumiteľnosti vizualizácie.

Znenie dotazníka

Program zobrazuje skoky v EXE súboroch (inštrukcia JMP, operačný kód 0xE9). Pre nápovedu pozri *"help\help.html"*. Cieľom dotazníka je zistiť, či je pomocou grafov možné rozoznávať druhy malware.

Rozoznávanie malware:

Najprv si môžeš pozrieť, ako vyzerajú anotované súbory. Je to vždy viac súborov naraz, načítanie bude chvíľu trvať. Výber normálnych súborov sa zobrazí pomocou skriptu *"normal.bat"*. Výber malware sa zobrazí pomocou skriptov *"malware-xxxx.bat"* - sú tu 4 druhy, každé po troch vzorkoch:

- ZPerm: *"malware-zperm.bat"*
- Virut: *"malware-virut.bat"*
- Cycbot: *"malware-cycbot.bat"*
- Aris: *"malware-aris.bat"*

Potom si pomocou skriptu *"guess.bat"* otvor očíslované vzorky EXE súborov a pokus sa určiť, o aký druh sa jedná (8 súborov, u každého vyber len jednu možnosť):

	Vzorky EXE súborov:							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
nedá sa určiť...								
normálny EXE súbor								
Zperm								
Virut								
Cycbot								
Aris								
podozrivý EXE súbor								

Hodnotenie programu

Ak si danú funkciu nepoužil alebo nechceš hodnotiť, môžeš nechať nevyplnené (alebo označ príslušnú položku).

- 0bodov - nehodnotím/nepoužil som
- 1bod - najhoršie
- 3body – najlepšie

		Počet bodov			
		0	1	2	3
Graf skokov (Barrel plot)	zrozumiteľnosť				
	funkcia zobrazenie početnosti				
	funkcia *fish eye zoom*				
	funkcia *"rotate jumps to cursor"*				
Graf početnosti (Quad plot)	zrozumiteľnosť				
	funkcia *fish eye zoom*				

Poznámky a pripomienky

Tu môžeš napísať poznámky a pripomienky k programu.

Vyhodnotenie dotazníka

Program bol hodnotený antivírusovými výskumníkmi v spoločnosti AVG. Na dotazník odpovedalo do 23.05.2011 7 respondentov, podrobné výsledky sú v nasledujúcich tabuľkách.

Výsledky identifikácie malware

V dotazníku boli 4 normálne súbory a 4 druhy malware, každý z predstavených jeden krát. Podrobné výsledky, ako respondenti určili osem ukážkových súborov, sú v nasledujúcej tabuľke. Zelenou je označená správna odpoveď. Grafické vyhodnotenie viď v kapitole 7.2 diplomovej práce.

	Súbor 1 (d3dx.dll)	Súbor 2 (Virut)	Súbor 3 (osgcppd.exe)	Súbor 4 (Cycbot)	Súbor 5 (pdfsettings.dll)	Súbor 6 (Aris)	Súbor 7 (Zperm)	Súbor 8 (cjpeg.exe)
1	normálny EXE súbor	Virut	podozrivý EXE súbor	Cycbot	normálny EXE súbor	Aris	ZPerm	normálny EXE súbor
2	normálny EXE súbor	nedá sa určiť...	Aris	ZPerm	normálny EXE súbor	Aris	Cycbot	normálny EXE súbor
3	normálny EXE súbor	Virut	normálny EXE súbor	Cycbot	normálny EXE súbor	Aris	ZPerm	normálny EXE súbor
4	normálny EXE súbor	Virut	normálny EXE súbor	Cycbot	normálny EXE súbor	Aris	ZPerm	normálny EXE súbor
5	nedá sa určiť...	podozrivý EXE súbor	nedá sa určiť...	Cycbot	Virut	Aris	ZPerm	normálny EXE súbor
6	normálny EXE súbor	Virut	normálny EXE súbor	Cycbot	normálny EXE súbor	podozrivý EXE súbor	ZPerm	normálny EXE súbor
7	normálny EXE súbor	podozrivý EXE súbor	normálny EXE súbor	Cycbot	normálny EXE súbor	podozrivý EXE súbor	ZPerm	normálny EXE súbor

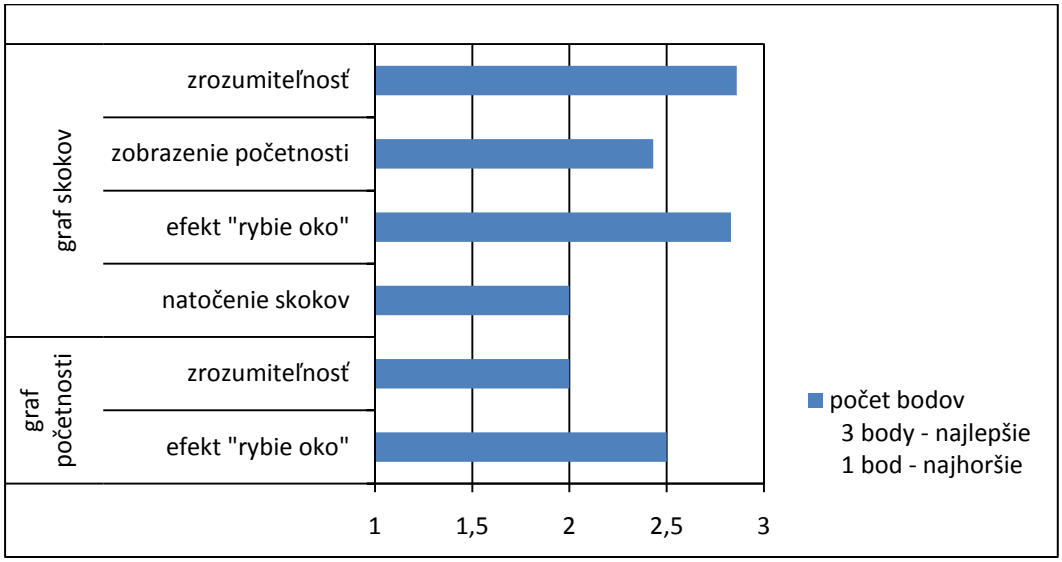
Tabuľka A.1: Podrobné vyhodnotenie úspešnosti identifikácie malware

Výsledky hodnotenia programu

Hodnotenie jednotlivých funkcií a grafov je znázornené v grafe na obrázku A.1. Keď respondent označil položku „*nehodnotím/nepoužil som*“, odpoveď sa do hodnotenia v grafe neráta. Podrobné výsledky sú v tabuľke A.2.

	Graf skokov (Barrel plot)				Graf početnosti (Quad plot)	
	zrozumiteľnosť	**funkcia zobrazenie početnosti**	**funkcia *fish eye zoom***	**funkcia *"rotate jumps to cursor"***	**zrozumiteľnosť**	**funkcia *fish eye zoom***
1	3 b	3 b	3 b	3 b	3 b	3 b
2	3 b	3 b	3 b	2 b	2 b	2 b
3	3 b	2 b	3 b	1 b	-	-
4	3 b	3 b	3 b	2 b	1 b	3 b
5	3 b	2 b	-	-	2 b	2 b
6	3 b	2 b	2 b	2 b	2 b	3 b
7	2 b	2 b	3 b	2 b	2 b	2 b

Tabuľka A.2: Podrobné výsledky hodnotenia funkcií v programe



Obrázok A.1: Užívateľské hodnotenie programu

Príloha B

Manuál na použitie programu

JumpGraph

Program graficky zobrazuje skoky v PE EXE súboroch.

- Za skok sa považuje inštrukcia *relative near jump* (operačný kód 0xE9), ktorá má ako operand 32 bajtový *offset*.
- Skoky sú odlíšené podľa smeru:
  - skoky dopredu (kladný offset) - modrá farba
  - skoky dozadu (záporný offset) - červená farba
- Program vie zobraziť aj početnosť skokov (jumps rate), je to množstvo skokov, ktoré prechádzajú ponad danú adresu. K dispozícii sú 2 druhy grafov:
  - Graf skokov (barrel plot)
  - Graf početnosti (quad plot)

Základné ovládanie programu

Program vie načítať súbory vo formáte PE EXE (.exe, .dll a pod.). Otvoriť súbor je možné presunutím myšou (*drag and drop*). Dá sa otvoriť viacero súborov naraz, každý je zobrazený na samostatnej záložke (*tab*). Priamo v grafe je možné zobraziť rýchlu nápovedu pomocou **F1.**

Základné ovládanie:

- **Myš**
  - Ľavé tlačidlo myši - rotácia
  - Stredné tlačidlo myši - posunutie
  - Pravé tlačidlo myši - zoom
- **R** - resetovať pohľad
- **šípka vľavo / vpravo** - predošlá / ďalšia sekcia EXE súboru
- **ENTER** - prepnúť medzi pohľadom na všetky sekcie / len jednu sekciu
- **M** - prepnúť farebnú schému (tmavá / svetlá)
- **J** - zobraziť početnosť skokov (jumps rate)
- **F1** - rýchla nápoveda

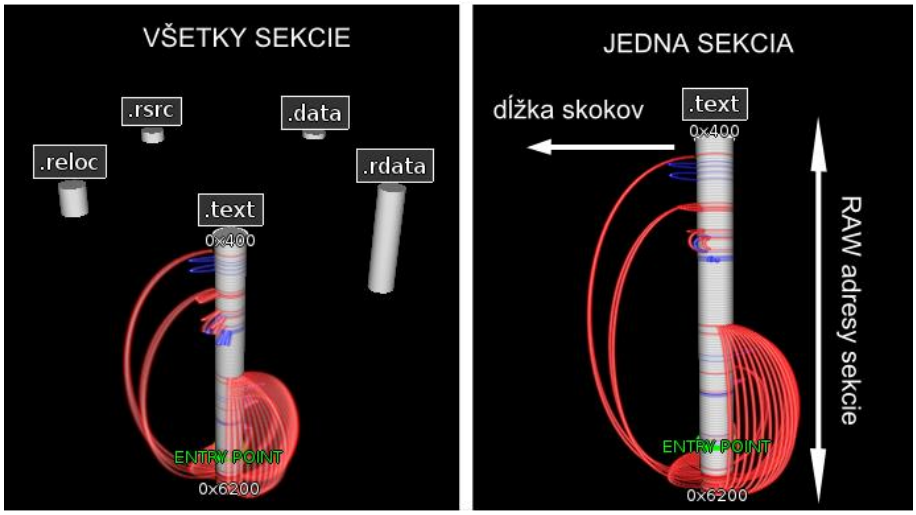
Ďalšie funkcie:

- **CTRL + O** - otvoriť súbor

- **CTRL + S** - uložiť obrázok
- **S** - nastavenia
- **1** - zobraz graf skokov (barrel plot)
- **2** - zobraz graf početnosti (quad plot)

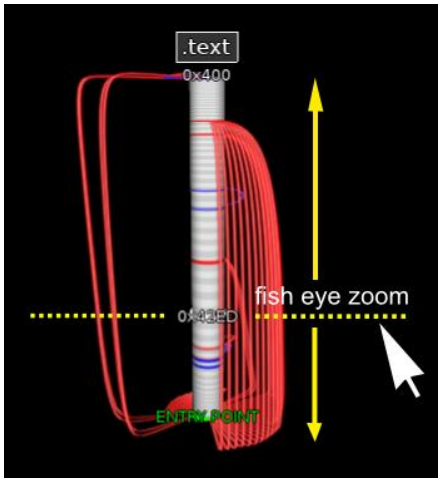
Graf skokov (barrel plot)

Graf skokov zobrazuje sekcie EXE súboru ako valec, ktorého dĺžka odpovedá veľkosti sekcie. Skoky sú reprezentované modrými a červenými čiarami. Skoky sú po obvode valca usporiadané podľa ich menšej adresy. Sú možné dva režimy zobrazenia: zobrazenie všetkých sekcií, alebo len jednej sekcie (prepínanie medzi nimi pomocou **ENTER**). Pomocou popisiek sa zobrazuje začiatok a koniec sekcie a aktuálna pozícia (RAW adresy).



Ovládanie (použiteľné v oboch režimoch):

- **CTRL + kolečko myši** - fish eye zoom
  - Posunúť kurzor nad pozíciu, kde chceme mať najväčšie priblíženie
  - Kolečko myši - priblížiť/oddialiť



- **ALT** - natočí skoky k pozorovateľovi
  - Posunúť kurzor nad pozíciu, na ktorej majú byť skoky natočené k nám
- **ENTER** - prepnúť medzi pohľadom na všetky sekcie / len jednu sekciu

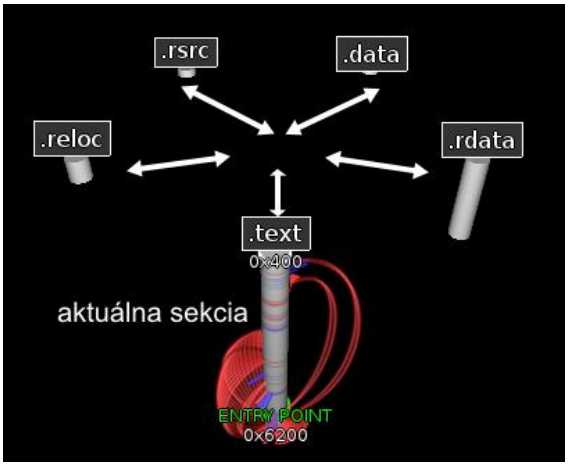
- **M** - prepnúť farebnú schému (tmavá / svetlá)
- **J** - zobraziť/schovať početnosť skokov (jumps rate). Tá je zobrazená samostatne pre skoky dopredu (modrá) a dozadu (červená).

Režim zobrazenia všetkých sekcií

Zobrazuje všetky sekcie EXE súboru usporiadané do kruhu. Aktuálna sekcia (zobrazená najbližšie) má zobrazené ďalšie podrobnosti a je možné s ňou manipulovať (fish eye zoom, natáčanie).

Ovládanie:

- **Klik pravým tlačidlom myši** - zvoliť aktuálnu sekciu
- **SHIFT + posunutie pravým tlačidlom myši** - posunutie sekcií od seba/k sebe

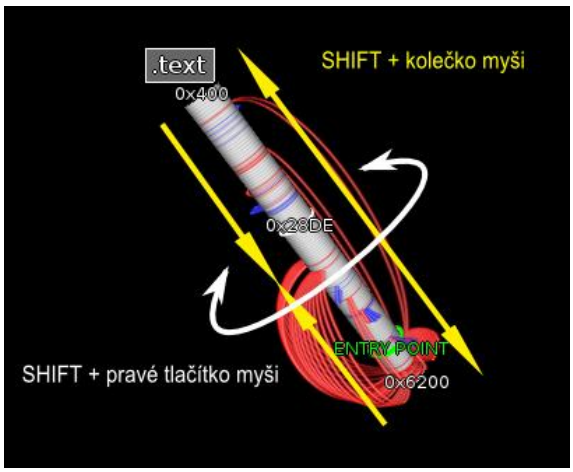


Režim zobrazenia jednej sekcie

Zobrazuje len jednu sekciu s ktorou je takto možné lepšie manipulovať. Aktuálnu zobrazenú sekciu je možné meniť pomocou šípok vpravo/vľavo.

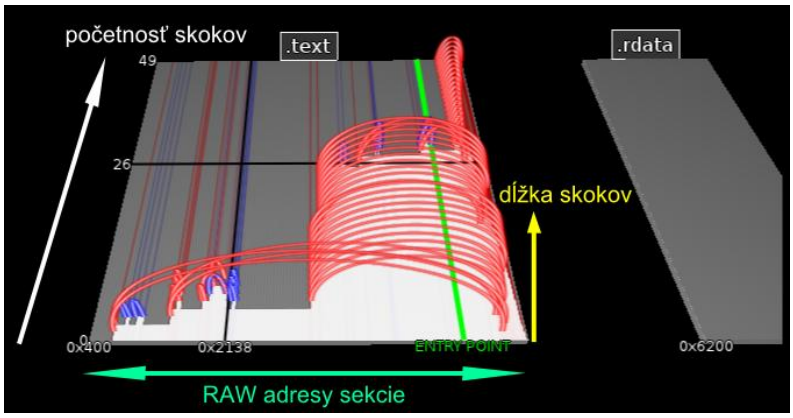
Ovládanie:

- **SHIFT + pravé tlačidlo myši** - rotácia sekcie okolo osi valca
- **SHIFT + kolečko myši** - zoom (predĺžiť/skrátiť sekciu)



Graf početnosti (quad plot)

Početnosť je množstvo (hustota) skokov, ktoré prebiehajú "ponad" danú adresu. Graf početnosti zobrazuje sekciu v tvare obdĺžnika. Početnosť je vyznačená ako histogram, kde X os sú RAW adresy sekcie EXE súboru a Y os je početnosť skokov na danej adrese. Skoky sú reprezentované modrými a červenými čiarami.



Je možný len jeden režim zobrazenia (všetky sekcie naraz). Manipulovať sa dá len s aktuálnou zvolenou sekciou, ktorú je možné meniť pomocou šípok vpravo/vľavo.

Ovládanie (použiteľné v oboch režimoch):

- **CTRL + kolečko myši** - fish eye zoom
  - Posunúť kurzor nad pozíciu, kde chceme mať najväčšie priblíženie
  - Kolečko myši - priblížiť/oddialiť
- **M** - prepnúť farebnú schému (tmavá / svetlá)

Inštalácia programu

Nie je potrebná inštalácia. Potrebné súbory na spustenie sú:

- **JumpGraph.exe** - samotný program
- **settings.xml** - XML súbor s nastaveniami (ak neexistuje, vytvorí sa nový s implicitnými hodnotami)

Požiadavky:

- operačný systém Windows XP / Vista / Windows 7
- podpora 3D grafickej akcelerácie pomocou **OpenGL 2.0** alebo novšie

Príloha C

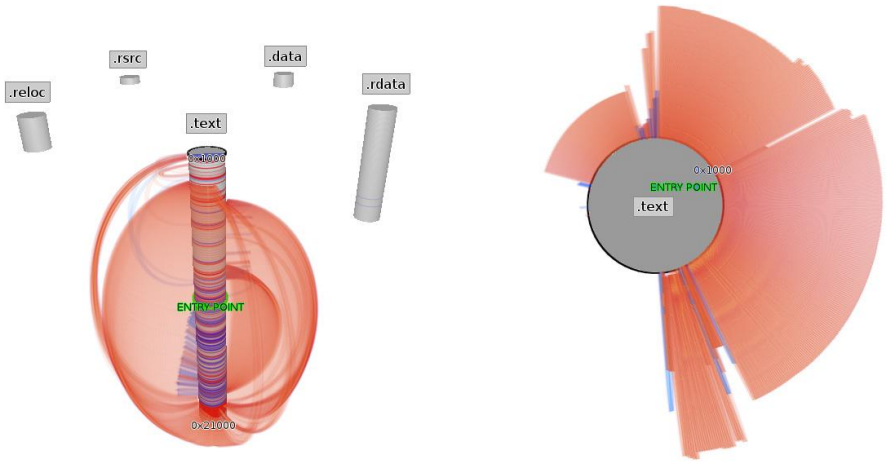
Ukážky vizualizácie programového toku

V tejto prílohe sú niektoré ukážky súborov, ktoré boli použité na testovanie programu formou dotazníku (kompletný súbor použitý na testovanie je priložený na DVD).

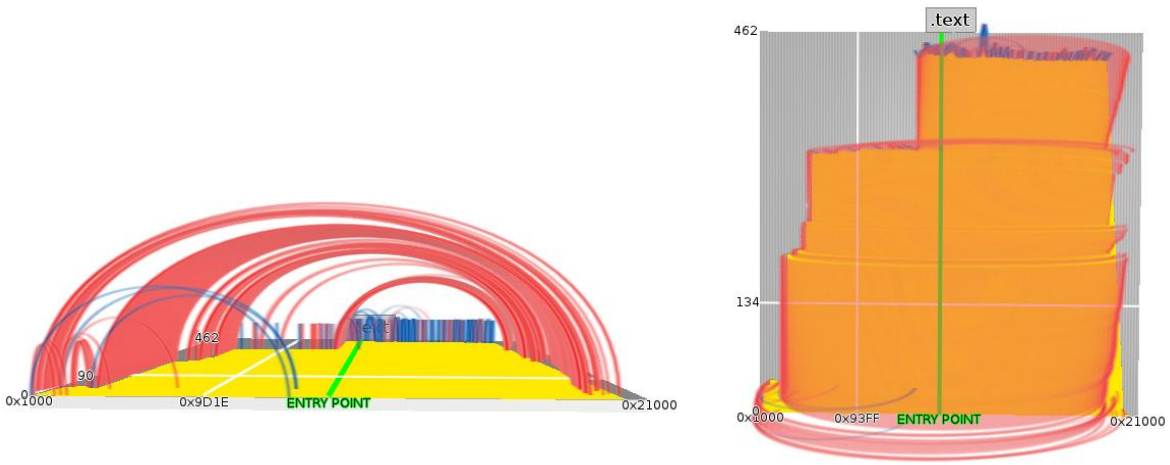
Vzorky normálnych PE EXE súborov

Pdfsettings.dll

(súčasť aplikácie *Acrobat Reader* od *Adobe*)
Binárny kód sa nachádza v sekcii *text*, skoky majú pravidelnú štruktúru. Zhluky skokov rovnakého typu sa dajú identifikovať aj v grafe početnosti (na základe podobnej dĺžky).



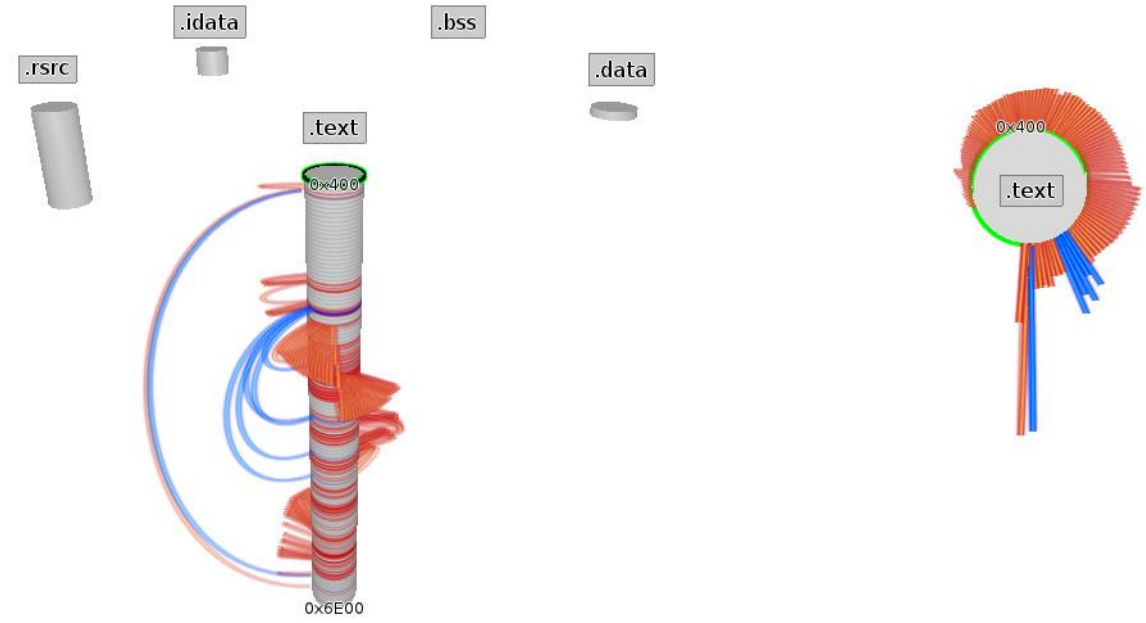
Obrázok C.1: Pdfsettings.dll, graf skokov, vpravo pohľad na sekciu *text* zhora



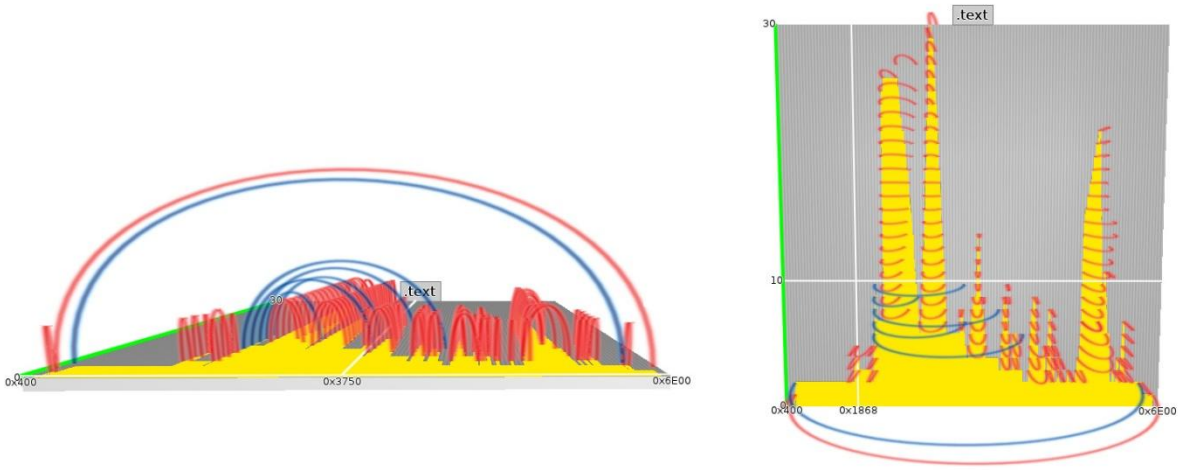
Obrázok C.2: Pdfsettings.dll, graf početnosti (vyznačená žltou farbou)

Cjpeg.exe

(nástroj na komprimovanie obrázkov vo formáte JPEG od *The Independent JPEG Group*)
Binárny kód sa nachádza v sekcii *.text*, skoky majú pravidelnú štruktúru. To, že skupiny skokov začínajú alebo končia na tých istých adresách spôsobuje, že sa hodnota početnosti skokov nemení postupne, ale prudko na viacerých miestach (viď obrázok C.4 vpravo, histogram početnosti je vyznačený žltou farbou).



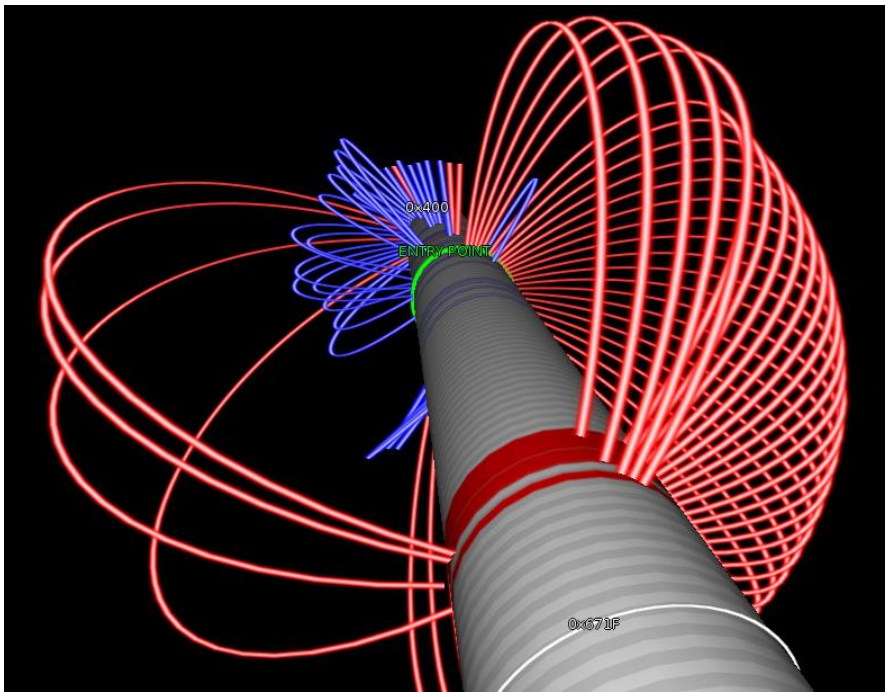
Obrázok C.3: Cjpeg.exe, graf skokov, vpravo pohľad na sekciu *text* zhora



Obrázok C.4: Cjpeg.exe, graf početnosti (vyznačená žltou farbou)

D3dx.dll

(súčasť *Direct3D* na OS Windows)
Demonštrácia efektu „žiariace čiary“ implementovaného pomocou falošných volumetrických čiar, ktorý lepšie vynikne v tmavej farebnej schéme určenej pre použitie na monitore. V tejto práci sa kvôli tlači na papier využíva hlavne svetlá farebná schéma.

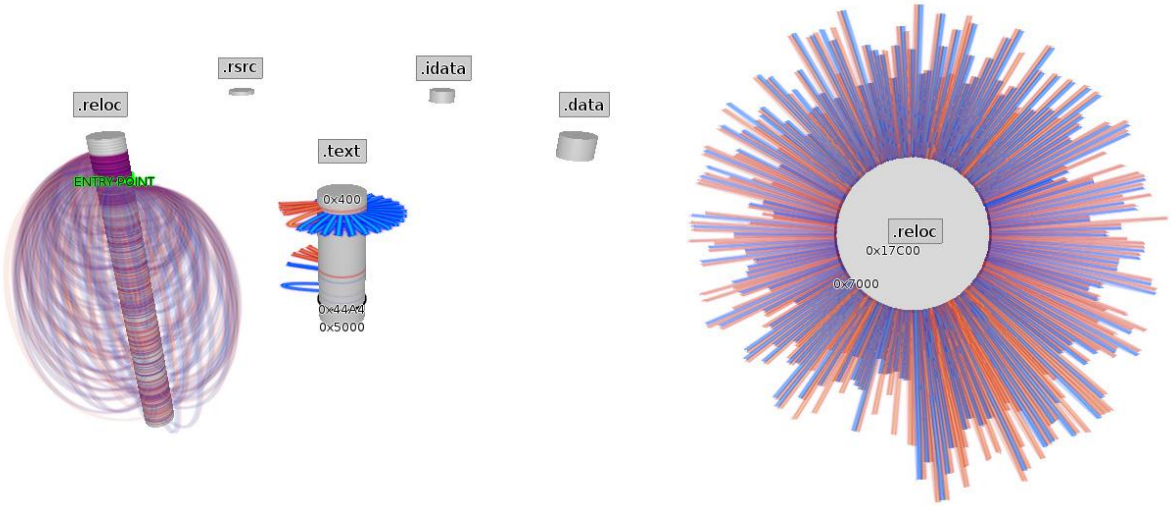


Obrázok C.5: D3dx.dll, ukážka efektu volumetrických čiar zblízka

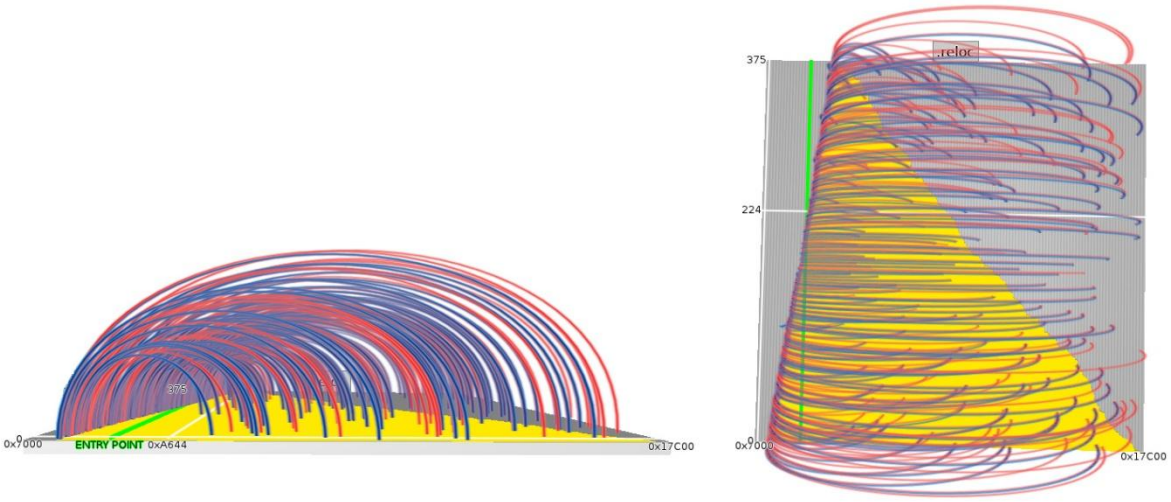
Vzorky malware

W32/ZPerm

Metamorfný vírus je typický svojou nepravidelnou štruktúrou skokov (ako sa striedajú oba druhy skokov značené červenou a modrou farbou, tak sa zlievajú do fialovej). Pôvodný program je v sekcii *text*, telo vírusu je v sekcii *reloc*. V grafe početnosti (početnosť skokov je vyznačená v grafe žltou farbou) sa na rozdiel od normálnych súborov nedajú identifikovať samostatne oddelené skupiny skokov rovnakého typu. V histograme početnosti nie sú rozoznateľné viaceré zmeny („impulzy“) ako to býva u normálnych EXE súborov (napríklad *cjpeg.exe* vyššie), početnosť plynule stúpa a potom klesá.



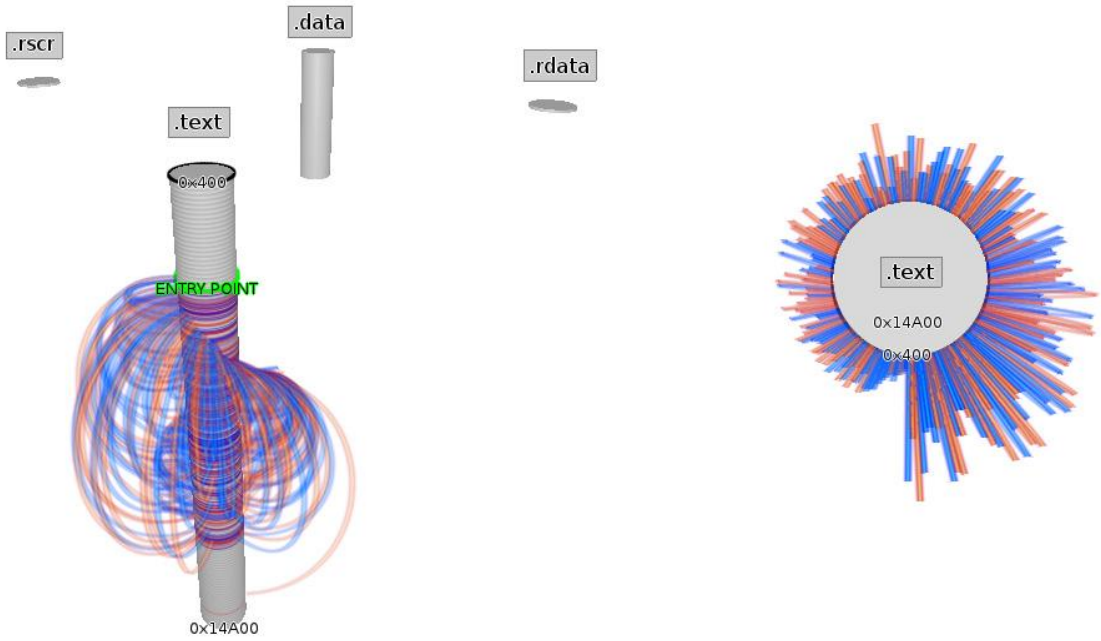
Obrázok C.6: W32/ZPerm, graf skokov, vpravo pohľad na sekciu *reloc* zhora



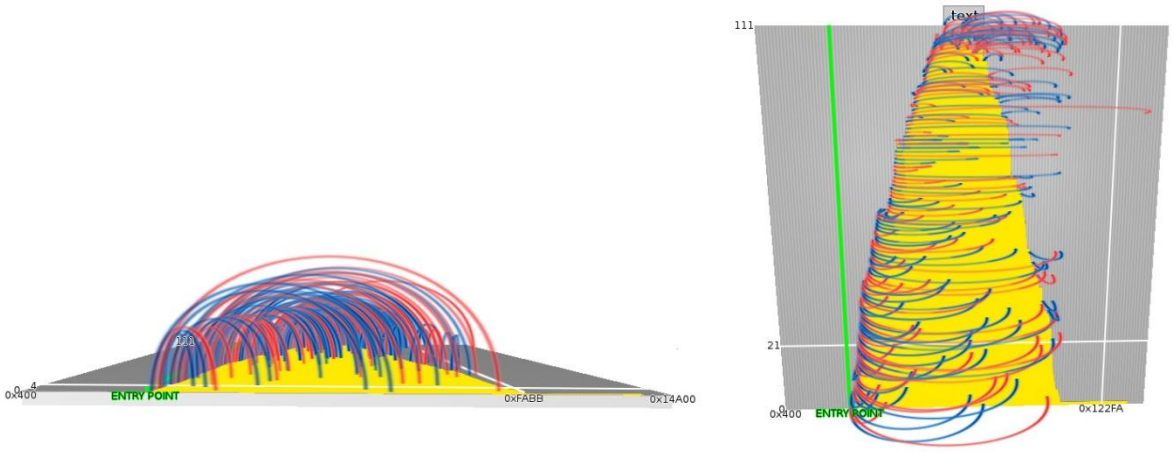
Obrázok C.7: W32/ZPerm, graf početnosti, pohľad na sekciu *reloc* obsahujúcu telo vírusu

Backdoor.Cycbot

Trójsky kôň má nepravidelnú štruktúru skokov v sekcii *text*, podobne ako telo vírusu ZPerm. Početnosť (je vyznačená v grafe početnosti žltou farbou) na rozdiel od normálnych súborov postupne stúpa aj klesá k stredu oblasti, kde sa nachádza programový kód, nedajú sa podľa nej identifikovať nejaké samostatné zhluky skokov rovnakého typu.



Obrázok C.8: Backdoor.Cycbot, graf skokov



Obrázok C.9: Backdoor.Cycbot, graf početnosti (vyznačená žltou farbou)