VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

TECHNIKY NAPADENÍ ELF/PE SOUBORŮ A JEJICH DETEKCE

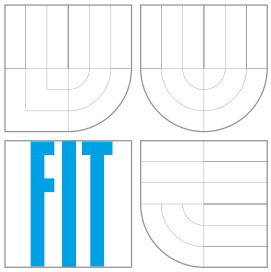
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ADAM BRUNAI

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

TECHNIKY NAPADENÍ ELF/PE SOUBORŮ A JEJICH DETEKCE

ATTACK TECHNIQUES ON ELF/PE FILES AND DETECTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ADAM BRUNAI

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MAROŠ BARABAS

BRNO 2012

Abstrakt

Tato práce se zabývá technikami napadení spustitelných souborů v OS Windows. Jejím hlavním cílem je analyzovat techniky infekce v kontextu složitosti jejich implementace a detekce. Před samotnou analýzou technik infekce bude čtenář obeznámen s problematikou spustitelných souborů. Součástí práce je demonstrační nástroj „pein“, který řeší implementaci infekce. V závěru se práce zabývá analýzou malwaru a technikami detekce.

Abstract

This thesis deals with the attack techniques on executable files in Windows OS. Its main goal is to analyze the file infection techniques in terms of their implementation and detection. Before the analysis, the reader will become familiar with executable files. Part of the thesis is demonstration tool named “pein”that solves implementation of infection. In conclusion, the work deals with the malware analysis and detection techniques.

Klíčová slova

bezpečnost, malware, analýza, trojský kůň, infekcia, heuristika, Windows, Portable Executable, IA-32

Keywords

security, malware, analysis, trojan horse, infection, heuristic, Windows, Portable Executable, IA-32

Citace

Adam Brunai: Techniky napadení ELF/PE souborů a jejich detekce, bakalářská práce, Brno, FIT VUT v Brně, 2012

Techniky napadení ELF/PE souborů a jejich detekce

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Maroša Barabasa. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Adam Brunai
10. května 2012

Poděkování

Rád bych poděkoval vedoucímu této práce panu Ing. Marošovi Barabasovi za pomoc při výběru tématu bakalářské práce, odborné konzultace, rady a připomínky.

Obsah

Úvod

V dnešnej dobe sa stávajú počítače stále bežnejšou súčasťou nášho života. Tento trend spôsobuje, že výrazná časť nášho súkromia a financií je závislá od bezpečnosti používaných zariadení. Veľkým problémom v tejto oblasti je malware. S rozšírením internetu nastalo i rozšírenie malwaru a jeho výskyt pretrváva aj v dnešnej dobe. Napriek snahe vyvinúť vhodnú obranu voči malwaru, nebolo doteraz nájdené definitívne riešenie.

Pretože výrazná časť malwaru využíva k svojmu šíreniu spustiteľné súbory, bude hlavným cieľom našej práce analyzovať techniky infekcie týchto súborov. K napísaniu tejto práce ma viedli nasledujúce otázky: Môže útočník infikovať ľubovoľne rozsiahly spustiteľný súbor ľubovoľnou infekciou? Je možné tento postup zovšeobecniť? Je možné odhaliť infekciu spustiteľného súboru? Je možné chrániť software distribuovaný po internete voči modifikácii? Odpoveďami na tieto a ďalšie otázky sa budeme zaoberať v nasledujúcich kapitolách.

V tejto práci sa primárne zameriame na platformu Windows, architektúru IA-32 a formát spustiteľných súborov PE. Jedná sa o častú konfiguráciu, na ktorú sa orientuje väčšina popredných dodávateľov softwaru. Skúsenosti a znalosti v tejto oblasti sú však použiteľné i na iných operačných systémoch a formátoch.

V prvých dvoch kapitolách sa čitateľ oboznámi s problematikou malwaru a formátmi spustiteľných súborov. Tieto prerekvizitné znalosti sú nutné k pochopeniu nasledujúcich kapitol. Tretia kapitola sa zaoberá technikami infekcie, z ktorých jedna bude implementovaná v kapitole štyri. Posledná piata kapitola sa zaoberá možnosťami analýzy a detekcie malwaru s ohľadom na techniky predstavené v tretej kapitole.

Kapitola 1

Škodlivý software (malware)

Už od počiatku prvých osobných počítačov vznikal software, ktorý môžeme označiť za nežiadúci. Spočiatku sa jednalo o neškodné experimenty a žarty. Škodlivý software (*malware*), tak ako ho poznáme dnes, je určený k narušeniu funkcie počítača, zbieraniu citlivých informácií, alebo na získanie prístupu k napadnutému systému.

Jeho autori sú často finančne motivovaní. Napadnutý systém sa môže stať súčasťou siete napadnutých počítačov (botnet), ktorý útočník využíva k rozosielaniu spamu[1] alebo k DDoS[2] útokom. Medzi software vytvorený za účelom zisku radíme i *spyware*, ktorého úlohou je odosielanie citlivých dát užívateľa jeho autorovi.

Podľa správy spoločnosti Symantec z roku 2008 môže množstvo vznikajúceho malwaru presahovať množstvo legitímneho softwaru[19]. Táto tendencia podnecuje k výskumu malwaru a obrany voči nemu.

1.1 Základná klasifikácia

Malware môžeme na základe jeho účelu, chovania a spôsobu šírenia rozdeliť do nasledujúcich kategórií[20]:

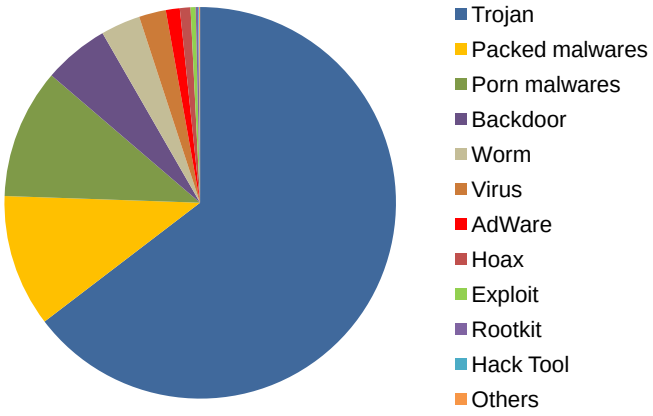
Vírusy – počítačové programy, ktoré sa samé replikujú a šíria z jedného systému na iný. Infikujú súbory na disku, väčšinou sú platformovo závisle.

Červy – Šíria sa po sieti samostatne využívaním zraniteľnosti cieľových systémov. Nie sú teda závislé na infekcií súborov na disku a môžu sa šíriť bez interakcie užívateľa.

Trójske kone – názov pochádza z gréckej mytológie. Trójsky kôň je program navonok vyzerajúci ako užitočný, no v pozadí vykonáva škodlivú činnosť. *Na rozdiel od vírusov sa sám nereplikuje.*

Podľa správy[2] publikovanej spoločnosťou AegisLabs, ktorá skúmala 14 miliónov unikátnych vzoriek malwaru, tvorili v roku 2011 väčšinu malwaru práve trójske kone. Grafické znázornenie môžeme vidieť na obr. 1.1.

[1] Nevyžiadaná správa masovo šírená internetom.
[2] Distribuovaný útok, pri ktorom dochádza k zahlteniu cieľa útoku požiadavkami.



Obr. 1.1: Rozšírenie malwaru v roku 2011 podľa typu

1.2 Trójsky kôň

Problematika *trójskych koní* je veľmi široká oblasť. Poznáme trójske kone, ktoré sa spoliehajú na techniky sociálneho inžinierstva a ich implementácia je triviálna. V minulosti sa často vyskytovali trójske kone vo forme spustiteľného súboru, ktorý neobsahoval okrem samotného tela trójskeho koňa nič iné. Ojedinelé však nie sú ani komplexné exempláre využívajúce pokročilé techniky infekcie a maskovania typické pre počítačové vírusy.

Medzi najznámejšie trójske kone patri AIDS z roku 1989, program zašifroval dáta na disku pomocou jednoduchých substitučných tabuliek a za dešifrovanie dát požadoval poplatok. Jedná sa o jeden z prvých prípadov použitia *škodlivej kryptografie*[3]. Autor tohoto programu bol neskôr chytený a odsúdený.

1.2.1 Vznik trójskeho koňa

Trójsky kôň môže vzniknúť[20]:

1. Ako pôvodná aplikácia (telo trójskeho kôňa bez inej funkčnosti).
2. Vytvorením z existujúceho programu – máme k dispozícii zdrojové kódy.
3. Vytvorením z existujúceho programu – bez prístupu k zdrojovým kódom (*infekcia*).

Prvá možnosť pre nás z hľadiska výskumu nie je zaujímavá. Taktiež pokiaľ máme k dispozícii zdrojové kódy existujúceho programu, nie je pre skúseného programátora problém takýto software upraviť k ľubovolnému účelu. Toto je typické pre open-source systémy. Napríklad úprava nástroja `ps` v UNIX systémoch umožňujúca skrytie PID určitého procesu.

V ďalších kapitolách tejto práce sa budeme zaoberať s *technikami infekcie*, bez znalosti zdrojových kódov hostiteľskej aplikácie. Cieľom nášho snaženia bude získať dostatok informácii, aby sme boli schopní analyzovať možnosti ich detekcie. Takýmto spôsobom môžeme detekovať i dovtedy neznáme infiltrácie. To čím sa zaoberať nebudeme, sú obfuskačné techniky[20][17], pretože presahujú rozsah tejto práce.

1.2.2 Popis činnosti

Pre trójske kone, s ktorými sa budeme stretávať v tejto práci, je typické, že sú vytvorené z existujúcej aplikácie. Táto aplikácia následne obsahuje deštruktívny kód, ktorý sa spúšťa pred samotnou aplikáciou. Často sa maskuje ako antivírusový software, alebo iný druh užitočnej aplikácie. Užívateľ inštalujúci túto aplikáciu v domnení, že slúži na boj s malwarom, tak paradoxne nainštaluje i trójskeho koňa, ktorý sa spustí pri každom štarte operačného systému.

Trójsky kôň môže taktiež umožniť jeho autorovi prístup do vzdialeného systému, limitovaný len prístupovými právami užívateľa. Útočník potom môže napríklad ukradnúť citlivé dáta (heslá, čísla platobných kariet atď), modifikovať alebo mazať súbory, zaznamenávať stisnuté klávesy, sledovať obrazovku užívateľa atď.

1.2.3 Typy trójskych koní

Rozoznávame nasledujúce základné typy trójskych koní[1]:

Backdoor – jedná sa o program, ktorý umožňuje útočníkovi získať vzdialený prístup na napadnutý systém, bez toho aby o ňom užívateľ vedel. Implementácia je podobná ako u legitímnych produktov umožňujúcich vzdialený prístup na báze TCP/IP (VNC atď).

Password-stealing – úlohou tejto infiltrácie je získať dôverné údaje z napadnutého systému. Obvykle sa jedná o heslá zachytené pomocou sledovania stlačených kláves (keylogger). Tento typ sa niekedy označuje ako spyware.

Logická bomba – typ trójskeho koňa, ktorého deštruktívna činnosť je viazaná na určitý čas alebo udalosť.

Downloader – jeho úlohou je stiahnuť ďalší malware do napadnutého systému. Stiahnutá infiltrácia môže byť znova downloader atď, čo môže spôsobiť výrazné zaťaženie procesoru a siete.

Dropper – tento typ je podobný downloaderu, ale malware obsahuje v sebe a nepotrebuje tak pripojenie k internetu pre jeho šírenie.

Veľa trójskych koní umožňuje vzdialený prístup k napadnutému systému bez *autentizácie*[3], čo býva často využívané útočníkmi (hackermi) k ovládnutiu napadnutého systému i keď infiltráciu do systému zaviedol niekto iný. K tomuto účelu sa využíva metóda *skenovania portov*. Pre určité trójske kone sú typické určité čísla portov. Na základe tejto informácie môže útočník zistiť, aká infiltrácia sa v systéme nachádza a akému komunikačnému protokolu rozumie.

Na tieto účely sa často používajú trójske kone ako *NetBus*, *Downloader-EV*, *Back Orifice* atď. Ochranu pred týmito a inými infiltráciami zabezpečuje antivírusový software.

[3] Autentizácia – overovanie identity užívateľa.

Kapitola 2

Formáty spustiteľných súborov

Infekcia spustiteľných súborov patrí stále medzi najčastejší spôsob šírenia malwaru. Od samotného vzniku počítačových systémov prešli spustiteľné súbory dlhou cestou. Dnes existujú rôzne formáty, na rôznych platformách, ktoré sa od sebe navzájom líšia. Obecne považujeme za spustiteľný súbor taký, ktorý obsahuje inštrukcie, pomocou ktorých môže počítač vykonať určitú činnosť. Z nášho pohľadu sú pre nás zaujímavé dva najpoužívanejšie binárne formáty:

- *Executable and Linkable Format (ELF)*
- *Portable Executable (PE)*

Formát ELF je typický pre unixové systémy, ale používajú ho i platformy ako PlayStation 3 alebo Wii. Tento formát sa často vyskytuje bez prípony súboru. V systémoch Windows sa už dlhú dobu používa formát PE, pre ktorý sú typické prípony `.exe` a `.dll`. Okrem týchto formátov sa môžeme stretnúť s jednoduchými binárnymi súbormi s koncovkou `.COM`, alebo s interpretovanými súbormi (`.bat`, `.sh`, `.py` atď).

Pochopenie štruktúry týchto súborov a spôsobu, akým s nimi operačný systém zaobchádza, je kľúčové pre pochopenie metód infekcie týchto súborov. Ďalej sa zameriame na často napádaný 32-bitový formát PE.

2.1 Reprezentácia algoritmu

Počítače slúžia primárne k riešeniu problémov, zjednodušovaniu bežných činností a k vykonávaniu iných úloh. Aby sme mohli od počítača vyžadovať vykonanie určitého algoritmu, musíme ho reprezentovať vo forme, ktorej rozumie. Bežne sa k tomuto účelu používajú programovacie jazyky:

Interpretované – zdrojový kód je nepriamo vykonaný interpretom (software). Používajú sa rôzne prístupy k implementácii.

Kompilované – zdrojový kód je prekladaný priamo do natívneho kódu procesoru (podľa toho akú inštrukčnú sadu používa).

Medzi interpretované jazyky radíme i jazyky *skriptovacie* (Python, PHP, atď), alebo jazyky prevádzané do *bytecodu*[1] (Java). Infekcia súborov, ktoré sú interpretované je možná, nie je však obvyklá. Je špecifická podľa programovacieho/skriptovacieho jazyka, ktorý sa v súbore nachádza.

Budeme sa zaoberať iba infekciou súborov, ktoré sú výsledkom kompilácie pre architektúru IA-32 (známa tiež ako i386, alebo x86). Jedná sa o veľmi rozšírenú architektúru, s ktorou je kompatibilná väčšina 32-bitových procesorov pre osobné počítače a notebooky. Získané skúsenosti môžeme uplatniť i na odvodené architektúry.

2.2 Architektúra IA-32

Aby sme boli schopný porozumieť jednotlivým metódam infekcie, oboznámime sa so základmi práce s architektúrou IA-32. Je to 32-bitová registrová architektúra. Používa inštrukčnú sadu typu CISC[2]. Prvýkrát bola táto architektúra použitá v mikroprocesore Intel 80386[13][8].

2.2.1 Registre

Výpis dostupných registrov nájdeme v tabuľke 2.1.

Typ	Registre	Primárne určenie
Univerzálne	`EAX, EBX, ECX, EDX`	Výpočty a uloženie medzivýsledkov
Indexové	`ESI, EDI, EBP, ESP, EIP`	Adresácia pamäte
Segmentové	`CS, DS, ES, SS, FS, GS`	Uloženie adresy segmentu

Tabuľka 2.1: Rozdelenie registrov podľa typu

Niektoré registre majú určené ich typické alebo výhradné použitie. Napríklad register `ESP` ukazuje na aktuálny vrchol zásobníku a `EIP` ukazuje na nasledujúcu inštrukciu. Registre `ESI` a `EDI` sa často používajú pri práci s poliami (SI – Source Index, DI – Destination Index). Špeciálnym typom je register príznakov `EFLAGS`, ktorý obsahuje jednobitové príznaky indikujúce stav aritmeticko-logickej jednotky procesoru.

2.2.2 Zásobník

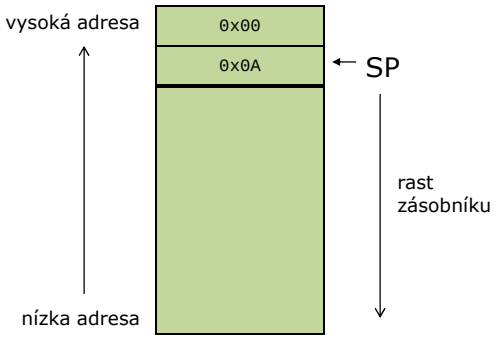
Zásobník je dátová štruktúra typu LIFO[3]. Procesory využívajúce architektúru IA-32 nemajú hardwarový zásobník, preto zásobník tvorí vyhradené miesto v pamäti. Zásobník rastie smerom od vyšších adries k nižším (viz obr. 2.1).

So zásobníkom môžeme pracovať buď priamo cez ukazovateľ `SP`, pomocou inštrukcie pre uloženie operandu na zásobník (`PUSH` operand) alebo výber zo zásobníka (`POP` operand).

[1] Inštrukčná sada navrhnutá pre efektívne vykonanie interpretom.

[2] Inštrukcie pokrývajú široký rozsah funkcií, ktoré by sa dali naprogramovať pomocou jednoduchších, už existujúcich inštrukcií.

[3] Last In Firs Out – posledný vložený prvok je vybratý ako prvý.



Obr. 2.1: Princíp zásobníku

Príklady použitia:

```
PUSH EAX ;uloží obsah registra EAX na zásobník
POP EBX  ;vyberie prvú položku na zásobníku a uloží ju do EBX
```

2.2.3 Inštrukcie

Inštrukčná sada IA-32 obsahuje veľké množstvo inštrukcií. My sa budeme zaoberať iba tými, ktoré sú relevantné k pochopeniu skúmanej problematiky. Obecný formát inštrukcie je nasledujúci[9]:

`názov inštrukcie` *nápoveda* operandy

Názov inštrukcie je krátke jednoznačné označenie (napríklad `MOV`) a nápoveda určuje veľkosť operandu v prípade, že by sa inak nedala jednoznačne určiť. Operandom inštrukcie môže byť hodnota, register alebo pamäť. Pokiaľ je operandom pamäť uzatvárame adresu do hranatých zátvoriek, aby sme ju odlíšili od číselnej hodnoty. Príklady použitia:

```
MOV EAX,5            ;uloží číslo 5 (desiatkovo) do registra EAX
MOV EAX,EBX          ;uloží obsah EBX do EAX
MOV EAX,[0xA0000000] ;uloží obsah pamäte na adrese 0xA0000000 do EAX
```

2.2.4 Volanie podprogramu

Podprogram (tiež funkcia alebo procedúra) označený návestím, môžeme volať pomocou inštrukcie `CALL`, tá zároveň uloží na zásobník adresu miesta, z ktorého bol podprogram volaný. Návrat z podprogramu zabezpečuje inštrukcia `RET`. Konvencia predávania parametrov závisí na kompilátore, alebo môže byť preddefinovaná (volania Windows API[4]). Zásadné je, či je za uvoľnenie parametrov zo zásobníka zodpovedný volaný podprogram, alebo volajúci program (viz kód 2.1).

[4]Rozhranie pre programovanie aplikácií systému Windows, používa konvenciu volania *stdcall*.

Kód 2.1 Volanie podprogramu

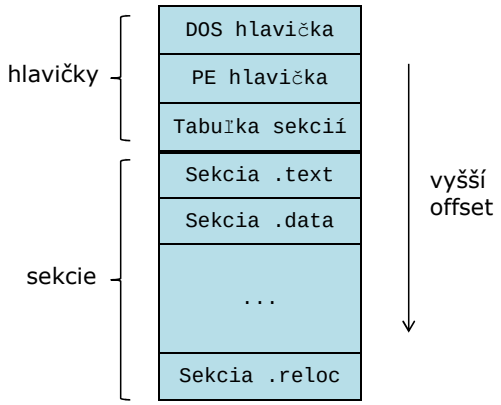
```
podprogram:      ;tu začína podprogram
...
RET              ;návrat z podprogramu

start:           ;začiatok programu
PUSH param1      ;prvý parameter o veľkosti 4 byty
PUSH param2      ;druhý parameter o veľkosti 4 byty
CALL podprogram ;volanie podprogramu
ADD SP,8         ;inštrukcia pričíta k SP č. 8 - odstráni parametre
```

2.3 Portable Executable (PE)

Súborový formát PE používajú systémy Windows už od verzie Windows NT 3.1. Cieľom Microsoftu bolo vyvinúť formát, ktorý by mohli používať všetky platformy používajúce ich OS. Je určený pre spustiteľné súbory (`.exe`), dynamicky linkované knižnice (`.dll`) a objektové súbory (`.obj`). Vychádza z Unixového formátu COFF. Momentálne podporuje viacero procesorov a inštrukčných sád (IA-32, IA-64, PowerPC, MIPS, ARM atď). Jeho 64-bitová verzia sa označuje ako PE32+. V tejto práci sa budeme venovať iba 32-bitovej verzii, obe sú však konceptuálne rovnaké a rozdiely sú minimálne.

Pri spustení súboru mapuje zavádzač OS jeho obsah do pamäti. Ak je štruktúra súboru v poriadku, inicializuje dáta[5], ktoré nebolo možné určiť v dobe prekladu a spustí príslušný kód. Štruktúra súboru pozostáva z hlavičiek a sekcií tak, ako môžeme vidieť na obrázku 2.2. Je potrebné si uvedomiť, že obraz súboru, ktorý zavádzač operačného systému mapuje do pamäti, nie je úplne zhodný s obsahom súboru. Toto je spôsobené rozdielnym zarovnaním sekcií v pamäti a v súbore, o ktorom si povieme neskôr.

Ďalší dôležitý pojem je *RVA* (Relative Virtual Address). Ako názov napovedá, jedná sa o offset v mapovanom súbore (po nahratí do pamäte). Keď bude v ďalšom texte spomenutá *virtuálna adresa*, je tým myslená vždy adresa v pamäti.



Obr. 2.2: Štruktúra PE súboru

[5] Adresy importovaných funkcií, adresy pri relokácii ...

2.3.1 Hlavička

Pre prácu s PE súbormi poskytuje Windows sadu štruktúr, tá sa nachádza v hlavičkovom súbore `WINNT.H`. Každý PE súbor začína (z dôvodu zachovania kompatibility) krátkym úsekom kódu pre MS DOS. Pri spustení PE pod MS DOS informuje užívateľa, že daný program nemôže pod týmto OS bežať. Táto štruktúra sa volá `IMAGE_DOS_HEADER` a obsahuje pre nás jedinú zaujímavú položku, a to hodnotu `e_lfanew`.

Táto položka obsahuje offset *PE hlavičky* v súbore. Tá je pomenovaná `IMAGE_NT_HEADERS` a je definovaná nasledovne[15][16]:

```
typedef struct _IMAGE_NT_HEADERS {
    DWORD Signature;
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

Položka `Signature` validného PE súboru obsahuje ASCII hodnotu „PE00“. Štruktúra `FileHeader` obsahuje základné informácie:

WORD Machine – nesie identifikátor cieľového procesoru (architektúry). Pre IA-32 je to `0x12CH` (existuje makro `IMAGE_FILE_MACHINE_I386`, ktoré túto hodnotu definuje).

WORD NumberOfSections – indikuje koľko sekcií sa nachádza v PE súbore. Túto hodnotu menia techniky infekcie, ktoré pridávajú novú sekciu.

WORD Characteristics – obsahuje bitové príznaky indikujúcich atribúty súbory. Na základe tejto hodnoty vieme rozlíšiť napríklad, či sa jedná o spustiteľný súbor alebo DLL.

Napriek tomu, že nasledujúca štruktúra sa volá `OptionalHeader`, je povinná a obsahuje nemenej dôležité informácie:

DWORD AddressOfEntryPoint – táto položka obsahuje RVA vstupného bodu. Ukazuje na miesto, odkiaľ sa začne vykonávať program. Adresa vstupného bodu je pre nás kľúčová, pokiaľ chceme zmeniť beh programu.

DWORD ImageBase – pri linkovaní PE súboru sa predpokladá, že bude mapovaný na konkrétnu adresu. Táto adresa je uložená v tejto položke a obvykle má hodnotu `0x40000000`. Po pričítaní akejkoľvek RVA k `ImageBase`, dostaneme skutočné umiestnenie v pamäti. Viac sa dozvieme v sekcii 2.3.5.

DWORD SectionAlignment – každá sekcia musí byť zarovnaná na násobok tejto hodnoty, aby bolo možné uplatniť ochranu pamäte. Obvyklá veľkosť je `0x1000` (veľkosť stránky).

DWORD FileAlignment – jednotlivé sekcie sú v súbore zarovnané na násobok tejto hodnoty. Obvyklá veľkosť je `0x200` (veľkosť sektoru).

DWORD CheckSum – táto položka obsahuje kontrolný súčet súboru. Pri spustiteľných súboroch ho zavádzač nekontroluje, ale má význam pri DLL.

IMAGE_DataDirectory[16] – pole `IMAGE_DATA_DIRECTORY` štruktúr. Každá obsahuje RVA nejakej významnej časti spustiteľného súboru (importy, exporty, zdroje atď).

2.3.2 Tabuľka sekcií

Sekcie v PE súbore slúžia na logické rozčlenenie jeho obsahu (napríklad kód a dáta). Zároveň však môžeme jednotlivým sekciám nastaviť rôznu ochranu pamäti, čo pomáha odhaliť chyby behu programu. Preto musí byť každá sekcia zarovnaná na násobok veľkosti stránky pamäte.

Aby sa ušetrilo miesto v súbore, sú v ňom obsiahnuté sekcie zarovnané na menšiu hodnotu (veľkosť sektoru disku). Veľkosť sekcie v súbore je teda menšia, alebo rovnaká ako veľkosť sekcie v pamäti. Chýbajúce dáta v pamäti zavádzač nahradí hodnotou `0x00`.

Väčšina metód infekcie PE súborov vyžaduje úpravu tabuľky sekcií. Pre prácu s *tabuľkou sekcií* môžeme využiť pole štruktúr (pre každú sekciu jeden prvok) `IMAGE_SECTION_HEADER`, štruktúra je definovaná nasledovne[15][16]:

```
typedef struct _IMAGE_SECTION_HEADER {
    BYTE Name[IMAGE_SIZEOF_SHORT_NAME];
    union {
        DWORD PhysicalAddress;
        DWORD VirtualSize;
    } Misc;
    DWORD VirtualAddress;
    DWORD SizeOfRawData;
    DWORD PointerToRawData;
    ...
    DWORD Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

Okrem pola `Name`[6], ktoré obsahuje názov sekcie, sú pre nás dôležité nasledujúce položky:

VirtualSize – skutočná veľkosť sekcie v pamäti bez zarovnania.

VirtualAddress – RVA začiatku sekcie v pamäti.

SizeOfRawData – veľkosť sekcie v súbore (násobok zarovnania v súbore).

PointerToRawData – offset začiatku sekcie v súbore.

Characteristic – atribúty sekcie (zapisovateľná, spustiteľná atď).

Typický názov pre sekciu obsahujúcu kód je `.text`. Linkery od Borlandu používajú názornejší názov `.CODE`. Pre dátovú sekciu sú typické názvy `.data` (neinicializované dáta) a `.idata` (inicializované dáta). Ďalšie často používané názvy sú `.rscr`, `.reloc`, atď. Na tomto mieste je potrebné uvedomiť si, že názvy týchto sekcií slúžia len pre lepšiu orientáciu v súbore a zavádzač ich nepoužíva – nie sú záväzné.

2.3.3 Tabuľka importov

Programy pre Windows neustále používajú DLL, pretože samotné Windows API je založené na používaní týchto knižníc a preto obsahuje každý program takzvanú *tabuľku importov (IAT)*. Importy systémových knižníc `Kernel32.dll` a `User32.dll` nájdeme takmer v každej aplikácii a tvoria základ Windows API.

[6] Maximálne 8 bytov dlhý reťazec. Nemusí byť ukončený nulovým bytom `0x00`.

Importovať *symboly* (takto budeme označovať importované funkcie alebo premenné) môžeme zo systémových, ale aj z vlastných DLL. V praxi sa môžeme stretnúť s dvoma spôsobmi[14][5]:

Implicitné linkovanie – programátor uvedie hlavičkový súbor, ktorý obsahuje definíciu požadovanej funkcie alebo dát a linker sa postará o správne zahrnutie importovaných symbolov do IAT. Úlohou zavádzača je v tomto prípade správne inicializovať IAT pri zavádzaní programu tak, aby obsahovala aktuálne adresy importovaných symbolov.

Explicitné linkovanie – programátor použije funkcie `LoadLibrary()` a `GetProcAddress()`, pričom získa adresy importovaných symbolov za behu programu, ako návratovú hodnotu funkcie `GetProcAddress()`. Zavolanie menovaných funkcií nemusí nastať vôbec a preto sa tento spôsob používa v situáciach, kedy vieme potrebné DLL určiť až v čase behu programu. IAT sa v tomto prípade nepoužíva.

Všetky klasické aplikácie používajú niektorú z uvedených metód, aby i v prípade zmeny DLL (zmena adries symbolov v pamäti) bolo možné získať správne adresy importovaných symbolov. Implementácia volania importovanej funkcie (implicitné linkovanie) môže vyzerať napríklad nasledovne[14]:

```
.text
CALL 0x0040100C ;volanie importovanej funkcie, napríklad MessageBox()
...
0x0040100C:
JMP DWORD PTR [0x00405030] ;pozrieme sa do IAT kam skočiť
```

Kde inštrukcia `JMP` je nepriamy skok na adresu, ktorá je uložená na `0x00405030`. Adresa `0x00405030` leží v IAT a na jej pozícii sa nachádza skutočná adresa importovanej funkcie. O inicializáciu IAT sa postará zavádzač pri spustení súboru. Tento spôsob implementácie sa používa kvôli zjednodušeniu kompilácie, funkčný ekvivalent je:

```
.text
CALL DWORD PTR [0x00405030] ;na 0x00405030 leží adresa funkcie MessageBox()
```

2.3.4 Tabuľka exportov

Opakom importovania symbolov je ich export. *Tabuľku exportov* obsahujú iba DLL súbory a používa ju zavádzač pri hľadaní adresy exportovaného symbolu. Symboly môžu byť exportované/importované na základe ich mena, alebo ordinárneho čísla. Metódy infekcie tabuľku exportov priamo nepoužívajú a preto sa ňou nebudeme ďalej zaoberať.

2.3.5 Relokácia

PE súbory sú pri mapovaní do pamäte mapované na preferovanú bázovú adresu. Adresy, ktoré súbor obsahuje (adresy dát, funkcií, skokov atď) sú tak absolútne, určené v dobe kompilácie. Takýto kód nazývame *pozične závislý kód*. Každý `.exe` súbor má pridelený vlastný adresný priestor, takže nemôže nastať konflikt medzi jednotlivými procesmi. Súbory `.dll` sú však zdieľané a môže nastať situácia, kedy je preferovaná nahrávacia adresa obsadená

inou knižnicou[7]. V tomto prípade je potrebné vykonať *relokáciu*.

K tomuto účelu slúži sekcia `.reloc`, ktorá obsahuje RVA všetkých absolútnych adries v programe. Zavádzač môže vďaka tomu upraviť všetky adresy v programe tak, aby odpovedali novému umiestneniu programu. Rozdiel novej bázovej adresy a pôvodnej bázovej adresy nazývame *hodnota delta*. Zavádzaču stačí pričítať túto hodnotu ku každej absolútnej adrese v programe, aby vykonal relokáciu.

Predpokladajme nasledujúci úsek kódu[14]:

```
0x00401020: mov ecx,dword ptr [0x0040D434] ;8B 0D 34 D4 40 00
```

Prvé dva bajty tvoria operačný kód inštrukcie a zvyšné štyri adresu. Ak bude nová bázová adresa väčšia o hodnotu `0x1000` stačí túto hodnotu prirátať k hodnote typu `DWORD` na adrese `0x00401022`. Sekcia `.reloc` obsahuje štruktúry typu `IMAGE_BASE_RELOCATION`, tá je definovaná nasledovne:

```
typedef struct _IMAGE_BASE_RELOCATION {
    DWORD   VirtualAddress;
    DWORD   SizeOfBlock;
} IMAGE_BASE_RELOCATION;
```

Hodnota `VirtualAddress` určuje začiatok úseku pamäte, ku ktorému sa relokácie vzťahujú. `SizeOfBlock` obsahuje veľkosť bloku hodnôt typu `DWORD`, ktoré nasledujú za touto štruktúrou. Každá hodnota prislúcha jednej relokácii. Prvé 4 bity obsahujú typ relokácie (my budeme spracovávať iba typ `IMAGE_REL_BASED_HIGHLOW`), nasledujúcich 12 bitov obsahuje offset vzťahujúci sa k hodnote `VirtualAddress`, ich sčítaním získame umiestnenie adresy, ktorú chceme upraviť.

[7] Tejto situácii môžeme predchádzať mechanizmom nazývaným *binding*.

Kapitola 3

Techniky infekcie v OS Windows

Prvé techniky infekcie používané v systéme MS DOS sa výrazne líšili od tých, ktoré sa používajú dnes v systéme Windows. Súvisí to s faktom, že súbory PE sú omnoho komplexnejšie ako spustiteľné súbory používané v systéme MS DOS. Z tohoto dôvodu sa nimi nebudeme zaoberať a bez historického úvodu sa zameriame rovno na tie súčasné[20].

Špeciálna pozornosť bude venovaná využitiu sekcie `.reloc`, ktorá umožňuje integrovať infekciu do PE súboru a výrazne sťažuje detekciu.

3.1 Infekcia konca súboru

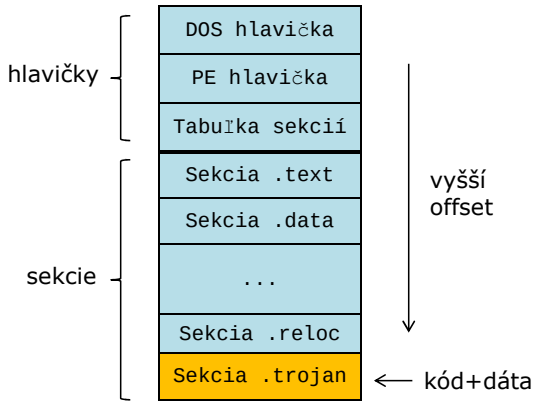
Existuje dobrý dôvod prečo pridávať škodlivý kód na koniec PE súboru. PE súbor obsahuje pozične závislý kód a aj keď to nebol pôvodný zámer, ten plní ochrannú funkciu. Do pozične závislého kódu totiž nemôžeme ľubovoľne zasahovať. Ak do neho pridáme, alebo odoberieme určitý úsek kódu, ten spôsobí posun zvyšných inštrukcií a ich adries. Adresy, ktoré predtým na tieto inštrukcie odkazovali, sa stanú neplatnými. Preto je najjednoduchším riešením zapisovať za koniec úseku kódu.

Jedná sa o najčastejšie používanú techniku infekcie PE súborov. Umožňuje zachovať pôvodnú funkčnosť infikovaného programu a jej implementácia je nenáročná. Rozoznávame dva nasledujúce spôsoby implementácie.

3.1.1 Pridanie novej sekcie

Telo škodlivého kódu i dáta sú umiestnené v novej sekcii vytvorenej na konci súboru (viz obr. 3.1). Nová sekcia sa zapíše do tabuľky sekcií a hodnota `NumberOfSections` sa zväčší o jedna. RVA vstupného bodu uložená v `AddressOfEntryPoint` sa obvykle nahradí adresou prvej inštrukcie vloženého kódu.

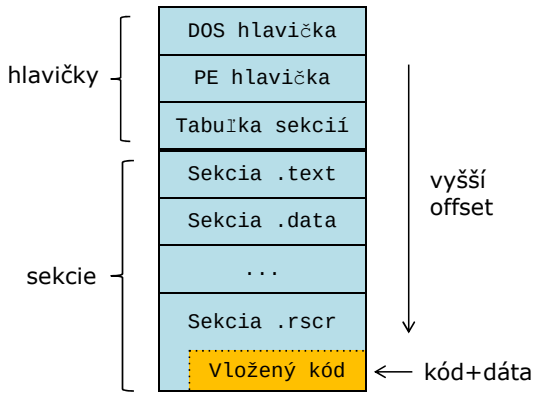
Nevýhodou tejto techniky z pohľadu útočníka je to, že pri ručnej analýze je infekcia ľahšie odhalená. Automatizované techniky analýzy sú schopné upozorniť na prítomnosť podozrivej sekcie a taktiež na to, že beh programu prebieha v dvoch sekciách.



Obr. 3.1: Pridanie novej sekcie

3.1.2 Rozšírenie poslednej sekcie

Alternatívou k pridaniu novej sekcie je rozšírenie poslednej sekcie (viz obr. 3.2). Tou je obvykle sekcia `.rscr`[1], ktorej atribúty neumožňujú spustenie kódu. Je preto potrebné tieto atribúty upraviť a zväčšiť veľkosť sekcie (položky `VirtualSize` a `SizeOfRawData`) tak, aby sa do nej vošiel vkladaný kód a jeho dáta. Podozrivé atribúty sekcie `.rscr` sú jedným zo spôsobov, ako túto techniku detektovať.



Obr. 3.2: Rozšírenie poslednej sekcie

3.2 Dutinová technika (cavity technique)

V sekcii 2.3.2 sme sa dozvedeli, že medzi jednotlivými sekciami vzniká voľné miesto v dôsledku ich zarovnania. Veľkosť sekcie sa zarovnáva na násobky minimálne 4 KiB. Voľné miesto, ktoré môže vzniknúť na konci sekcie označíme ako S, kde $S \in \{0\,\mathrm{B}, 1\,\mathrm{B}, 2\,\mathrm{B} \ldots 4095\,\mathrm{B}\}$.

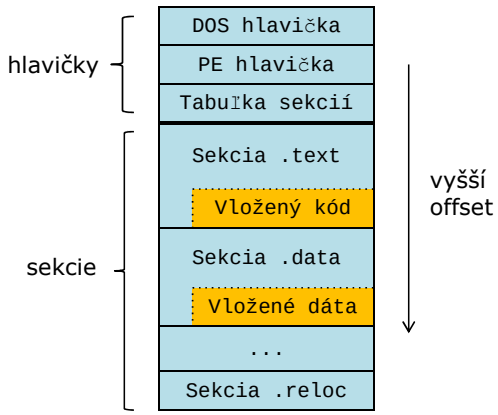
[1] Sekcia obsahuje nezapisovateľné dáta aplikácií, kurzory, ikony, dialógy atď.

Pretože rozdelenie pravdepodobnosti je rovnomerné a hodnoty tvoria aritmetickú postupnosť s $d = 1$, môžeme priemernú teoretickú hodnotu voľného miesta pre infekciu vyrátať podľa vzorca

$$\overline{S} = S_{\min} + \frac{S_{\max} - S_{\min}}{2} = 0\,\mathrm{B} + \frac{4095\,\mathrm{B} - 0\,\mathrm{B}}{2} = 2047{,}5\,\mathrm{B}. \tag{3.1}$$

Toto miesto vyplní zavádzač nulami a program ho nepoužíva. Samotná infekcia tak spočíva v zmene hodnoty `VirtualSize`. Ďalej je potrebné rozšíriť súbor o potrebný kód a dáta. V prípade potreby je možné využiť viacero medzier (dutín), rozdeliť kód na viacero častí a vhodným spôsobom ich za behu programu prepojiť.

Špeciálny prípad tejto techniky môžeme vidieť na obrázku 3.3. V tomto prípade je vkladaný kód vložený do sekcie `.text` a jeho dáta do sekcie `.data`. Nastane tak úzka integrácia pôvodnej aplikácie a vkladaného kódu.



Obr. 3.3: Dutinová technika

3.2.1 Využitie sekcie `.reloc`

Sekcia `.reloc` umožňuje relokáciu kódu a tým jeho zavedenie na ľubovolnú adresu. V kontexte infekcie spustiteľných súborov má však jedno ďalšie využite. Umožňuje vkladať dáta i do pozične závislého kódu a pomocou nej následne opraviť adresy odkazujúce na posunutú časť kódu. Toto umožňuje rozšíriť predchádzajúcu techniku dutinovej infekcie a vložiť tak kód ľubovolnej veľkosti, samozrejme za predpokladu, že infikovaná aplikácia túto sekciu obsahuje.

Operačné systémy od Microsoftu staršie ako Windows 98 túto sekciu používali v DLL a v `.exe` súboroch. S príchodom Windows 98 sa táto sekcia stala v `.exe` súboroch nepotrebná a jej výskyt je preto ojedinelý. Súborový formát PE32+ túto sekciu obsahuje, aby bolo možné použiť ASLR[2] a jej výskyt je povinný.

[2]Bezpečnostná technika, ktorá umožňuje stanoviť pozície kľúčových dát náhodne (pozícia knižníc, zásobníku, haldy atď).

Využitie sekcie `.reloc` môže byť preto v budúcnosti častejšie. Infekcie, ktoré týmto spôsobom vzniknú, budú komplexnejšie a ťažšie detekovateľné. Viac o použití sekcie `.reloc` je uvedené v kapitole 4.

3.3 Dynamické zavedenie DLL

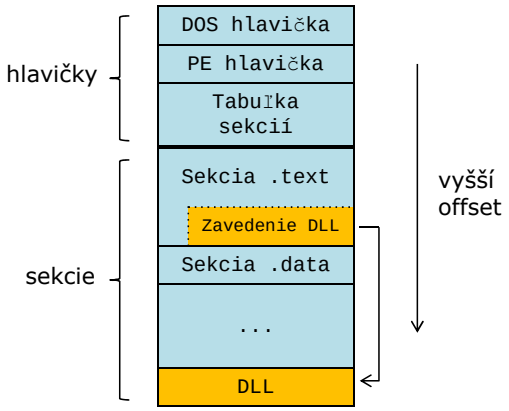
Princíp tejto techniky spočíva vo vytvorení DLL, ktorá obsahuje škodlivý kód, v jej následnom zavedení do pamäte a spustení (viz obr. 3.4). Samotná DLL môže byť uložená na konci spustiteľného súboru, alebo na pevnom disku v samostatnom súbore. V praxi sa môžeme stretnúť s dvoma spôsobmi implementácie:

Zavedenie pomocou volania LoadLibrary() – Windows API poskytuje túto funkciu, ktorá umožňuje dynamické zavedenie DLL za behu programu. Funkcia `LoadLibrary()` vyžaduje ako vstupný parameter umiestnenie DLL. Preto je potrebné tento súbor vytvoriť (napríklad v priečinku *tmp*), ak je DLL umiestnená v spustenom súbore.

Reimplementácia LoadLibrary() – táto technika[4] vyžaduje viac miesta pre kód, ktorý DLL zavedie do pamäte a spustí. Preto býva tento kód spravidla umiestnený na konci súboru.

Oba spôsoby implementácie vyžadujú vloženie krátkeho úseku kódu (stub), aby mohla byť knižnica DLL zavedená do pamäte. Ak tento úsek kódu nepoužíva techniky zamerané proti detekcii, môžeme túto techniku jednoducho detektovať na základe skenovania[3].

Za výhodu tejto techniky z pohľadu útočníka môžeme považovať to, že súbory DLL tvoria prirodzené rozhranie pre umiestnenie škodlivého kódu (autor môže použiť vyššie programovacie jazyky, vývojové prostredie atď) a hlbšie znalosti architektúry nie sú nevyhnutné.



Obr. 3.4: Dynamické zavedenie DLL

[3] Technika detekcie založená na vyhľadávaní postupnosti bajtov identifikujúcich infekciu.

3.4 Problémy techník infekcie

V nasledujúcej sekcii sa budeme venovať problémom a témam, ktoré nezávisia od zvolenej techniky infekcie. Ich riešeniam sa budeme podrobnejšie venovať v kapitole 4.

3.4.1 Spustenie vloženého kódu

Samotná infekcia obsiahnutá v spustiteľnom súbore nie je nebezpečná. Nebezpečnou sa stáva až keď sa začne vykonávať jej kód. Toho sa dosahuje rôznymi spôsobmi. Napríklad spustenie kódu infekcie môžeme dosiahnuť priamo, zmenou hodnoty `AddressOfEntryPoint`.

Existujú spôsoby ako sťažiť detekciu už známych infekcií (zavesenie sa na Windows API, EPO techniky atď), ale sú typické skôr pre počítačové vírusy.

3.4.2 Vlastnosti vloženého kódu

Kód infekcie sa líši od typického kódu generovaného kompilátormi. Čím je technika infekcie jednoduchšia, tým viac obmedzení musí vložený kód prekonať. Častým obmedzením je adresovanie v rámci vloženého kódu. Jednoduché techniky infekcie sú schopné vkladať iba kód, ktorý nepoužíva absolútne adresy (pozične nezávislý kód). Dáta využívane infekciou môžu byť umiestnené v inej sekcii, ale častejšie sú pevnou súčasťou vkladaného kódu.

Typický kód pre platformu Windows využíva pre väčšinu bežných úloh volania jeho API. Prvé infekcie tohoto typu používali pevne zadané adresy. Tieto adresy sa však stávajú neplatné pri zmenách knižnice, ktorá ich obsahuje. Ďalšou možnosťou je dynamické zistenie adresy pomocou volania `GetProcAddress()`.

Najsofistikovanejšia technika je pridanie importov infekcie do tabuľky importov hostiteľskej aplikácie a úprava adries importov v kóde. Pri štúdiu literatúry som na použitie tejto techniky nenarazil. Jej implementáciu umožňuje práve sekcia `.reloc` a je možné jej použitie v budúcnosti.

3.5 Zhrnutie a porovnanie techník infekcie

Všetky uvedené techniky si na záver zhrnieme vo forme tabuľky (viz tab. 3.1).

Technika	**Obtiažnosť implementácie**	**Obtiažnosť detekcie**	**Max. veľkosť kódu**	**Detekovateľná skenovaním**
Pridanie novej sekcie	jednoduchá	stredná	∞	×
Rozšírenie poslednej sekcie	jednoduchá	stredná	∞	×
Dutinová technika	stredná	zložitá	4 KiB	×
Dutinová technika s využitím sekcie `.reloc`	zložitá	zložitá	∞	×
Dynamické zavedenie DLL	stredná	jednoduchá	∞	✓

Tabuľka 3.1: Zhrnutie a porovnanie techník infekcie

Kapitola 4

Infekcia spustiteľných súborov

V tejto kapitole sa pozrieme bližšie na implementáciu infekcie PE súborov. Ako vhodnú techniku pre demonštráciu som sa rozhodol zvoliť *dutinovú techniku* infekcie. Jej detekcia nie je triviálna a budeme sa ňou zaoberať v kapitole 5.

Infikovať spustiteľné súbory sa dá pracným postupom pomocou vhodných nástrojov (hex editor, PE editor atď). Tento spôsob je relatívne rýchly, ale pre experimenty a detekciu nevhodný. Okrem toho sa dá použiť iba pri jednoduchých technikách infekcie a pri vkladaní jednoduchých úsekov kódu.

Preto vytvoríme nástroj s názvom `pein` (PE injector/infector), ktorý nám umožní túto činnosť automatizovať. `pein` ušetrí čas pri experimentovaní s PE súbormi a umožní testovať detekciu i na netriviálnych infekciách.

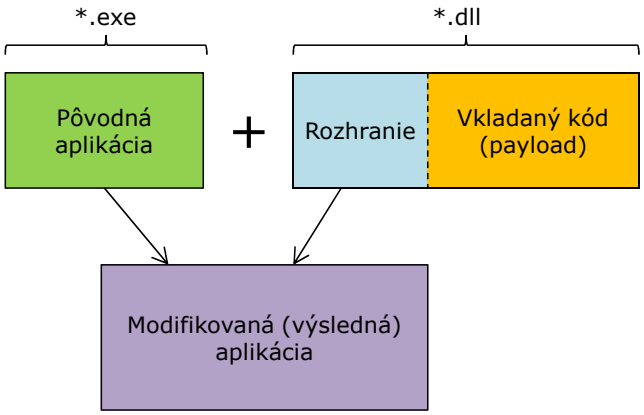
4.1 Koncept nástroja `pein`

Úlohou nástroja `pein`, implementovaného v jazyku C, je umiestniť kód infekcie uložený vo vhodnom formáte do ľubovoľného spustiteľného súboru. Ako nosič vkladaného kódu bude slúžiť DLL knižnica špeciálne skompilovaná za týmto účelom (viz obr. 4.1). **Pozor – nejedná sa o techniku dynamického zavedenia DLL**. DLL slúži iba ako nosič kódu, to znamená, že kód bude analyzovaný a extrahovaný. V infikovanom súbore teda neostanú žiadne pozostatky pôvodného DLL súboru.

Rozhranie bude v našom prípade tvoriť funkcia `DllMain()` obsiahnutá v DLL súbore. Táto funkcia bude tvoriť vstupný bod vkladaného kódu a bude volaná pred spustením kódu pôvodnej aplikácie. Alternatívou k tomuto prístupu by mohlo byť volanie na základe udalostí. Aj keď funkcia `DllMain()` neprijíma žiadne parametre, je možné tento koncept rozšíriť a vytvoriť komplexnejšie rozhranie umožňujúce napríklad run-time dekomprimáciu infekcie.

4.1.1 Payload

Pojem payload označuje v kontexte analýzy malwaru škodlivý výsledok jeho činnosti (mazanie dát, zobrazovanie urážlivého textu atď). Náš payload bude neškodný a jeho jedinou úlohou bude zobraziť informačné dialógové okno, oznamujúce jeho prítomnosť (viz kód 4.1).



Obr. 4.1: Princíp funkcie nástroja pein

Kód 4.1 Zdrojový kód payloadu v jazyku C

```
#include <windows.h> //obsahuje deklaráciu funkcie MessageBoxA()

void WINAPI DllMain(void)
{
    //ukazovateľ na funkciu MessageBoxA()
    int (WINAPI * pMessageBox)(HWND,LPCSTR,LPCSTR,UINT);
    //získame adresu funkcie MessageBoxA()
    pMessageBox = GetProcAddress(LoadLibrary("User32.dll"), "MessageBoxA");

    //vytvoríme dialógové okno pomocou volania MessageBoxA()
    pMessageBox(NULL, "Payload executed successfully.", "Trojan horse",
    MB_OK | MB_ICONWARNING | MB_SYSTEMMODAL);
}
```

Všímavejší programátori si iste všimnú, že sme použili explicitné linkovanie (viz sekcia 2.3.3). Implicitné linkovanie by vyžadovalo úpravu tabuľky importov hostiteľskej aplikácie s použitím jej sekcie `.reloc`. Táto implementácia nevyžaduje aby hostiteľská aplikácia túto sekciu obsahovala.

Aby sme získali čo najjednoduchší a najmenší[18] výsledok kompilácie, musíme zabezpečiť kompiláciu bez *C Runtime Library*[1]. To najjednoduchšie dosiahneme použitím kompilátoru a linkeru dodávanými spolu s vývojovým prostredím Visual Studio 2010 (MinGW je príliš naviazaný na C Runtime Library):

```
cl /nologo /c /O1 payload.c
link /nologo /DLL /ENTRY:DllMain /EXPORT:DllMain,@1 /NODEFAULTLIB
/SUBSYSTEM:WINDOWS /ALIGN:16 payload.obj Kernel32.lib
```

Výsledkom kompilácie je minimálny kód (viz kód 4.2).

Adresy označené výkričníkom sa po vložení kódu do hostiteľskej aplikácie stávajú neplatnými. Takto vytvorený DLL súbor obsahuje aj sekciu `.reloc`, ktorú využijeme pri opravách

[1] Implementácia štandardnej knižnice jazyka C, obvykle linkovaná staticky.

Kód 4.2 Zdrojový kód payloadu v jazyku assembler

```
10000240 D>/$ PUSH payload.100002B8!                ;/ProcNam.="MessageBoxA"
10000245   |. PUSH payload.100002AC!                ;|/FileName="User32.dll"
1000024A   |. CALL DWORD PTR [IAT:LoadLibraryA!]    ;|\LoadLibraryA
10000250   |. PUSH EAX                              ;|hModule
10000251   |. CALL DWORD PTR [IAT:GetProcAddress!] ;\GetProcAddress
10000257   |. PUSH 1030
1000025C   |. PUSH payload.1000029C!                ;ASCII "Trojan horse"
10000261   |. PUSH payload.1000027C!                ;ASCII "Payload exec..."
10000266   |. PUSH 0
10000268   |. CALL EAX
1000026A   \. RETN
```

adries dát vo vkladanom kóde. Sekcia `.reloc` v payloade teda umožňuje i vkladanie pozične závislého kódu, čo je jeden z dôvodov prečo je práve DLL súbor dobrým nosičom vkladaného kódu.

4.2 Implementácia nástroja `pein`

Jadrom nástroja `pein` je modul `PEinject.c`, ktorý obsahuje funkciu

```
int pe_inject(const char * source_name, //zdroj infekcie (*.dll)
    const char * target_name //cieľ infekcie (*.exe)
    );
```

Táto funkcia realizuje infekciu požadovaného PE súboru. Nad touto funkciou je potrebné vybudovať užívateľské rozhranie. Jednoduché textové rozhranie je použitie v module `main.c`. Funkcia `pe_inject()` vykonáva nasledujúce činnosti:

1. Získa a overí obrazy vstupných súborov.
2. Alokuje pamäť pre výstupný súbor.
3. Opraví adresy vo vstupnom kóde.
4. Skopíruje nezmenené hlavičky pôvodnej aplikácie do výstupného obrazu.
5. Skopíruje do výstupného obrazu pôvodné sekcie, pričom do nich priebežne pridáva vkladaný kód a jeho dáta.
6. Aktualizuje položky hlavičky.
7. Uloží výsledný obraz.

Na niektoré z týchto činností sa pozrieme v nasledujúcich kapitolách.

4.2.1 Obraz PE súboru

Windows API žiaľ neposkytuje vhodné rozhranie pre prácu s obrazom (image) PE súboru. Preto implementujeme funkcie `load_image()` a `save_image()`. Budú pracovať so štruktú-

rou `t_image` a budú plniť nasledujúce funkcie:

- Nahratie požadovaného PE súboru do pamäte
- Inicializácia ukazovateľov na hlavičky a tabuľky súboru
- Uloženie obrazu na pevný disk

Tieto a ostatné pomocné funkcie pre prácu s PE súborom sa nachádzajú v module `PEimage.c`.

4.2.2 Práca s adresami

Pri práci s PE súbormi budeme používať 2 typy adries:

- RVA (viz sekcia 2.3)
- Adresa v súbore

Keďže PE obraz bude nahratý v pamäti pomocou `load_image()`, budeme pod adresou v súbore rozumieť súčet adresy, na ktorej je obraz nahratý a offsetu v tomto súbore. Funkcie pre prepočet týchto adries sa volajú `rva2adr()` a `adr2rva()`. Nachádzajú sa tiež v module `PEimage.c`.

Adresy Windows API Aby sme mohli využívať funkcie Windows API aj vo vkladanom kóde, musíme opraviť adresy, ktoré ukazujú do IAT. Adresy vo vkladanom kóde ukazujú do IAT pôvodného súboru. Našou úlohou je nájsť všetky tieto výskyty a nahradiť ich novou adresou ukazujúcou do IAT hostiteľskej aplikácie.

Umiestnenie adresy požadovanej funkcie v IAT môžeme zistiť nasledujúcim spôsobom:

```
//zistíme umiestnenie v IAT zdrojového súboru (*.dll)
sourceGetProcRVA = get_symbol_rva("KERNEL32.dll", "GetProcAddress", source);

//zistíme umiestnenie v IAT hostitelského súboru (*.exe)
targetGetProcRVA = get_symbol_rva("KERNEL32.dll", "GetProcAddress", target);
```

Následne musíme prejsť všetky hodnoty v sekcii `.reloc`. Ak niektorá hodnota ukazuje na starú adresu (adresa v zdrojovom programe), musíme ju nahradiť novou adresou (adresa v hostiteľskom programe):

```
if(*value == source.pe_header->OptionalHeader.ImageBase+sourceGetProcRVA)
{
    *value = target.pe_header->OptionalHeader.ImageBase+targetGetProcRVA;
}
```

Adresy skokov a dát Podobným spôsobom spracujeme aj adresy skokov a dát. Ako príklad si ukážeme opravu adries dát. Najprv skontrolujeme či práve spracovávaná hodnota ukazuje do sekcie dát, ak áno, vypočítame offset položky, na ktorú ukazuje (offset v rámci sekcie). Ďalší krok je výpočet novej adresy v hostiteľskej aplikácii.

Dáta vkladaného kódu zapisujeme na koniec sekcie dát hostiteľskej aplikácie. Preto môžeme novú adresu vyrátať ako súčet začiatku hostiteľskej sekcie, veľkosti hostiteľskej sekcie a offsetu dát (viz kód 4.3).

Kód 4.3 Spracovanie adries skokov a dát

```
source_data_start = source.pe_header->OptionalHeader.ImageBase
+ source.section_header[SRC_DATA_SECTION].VirtualAddress;
target_data_start = target.pe_header->OptionalHeader.ImageBase
+ target.section_header[SRC_DATA_SECTION].VirtualAddress;

if(*value >= source_data_start && *value < source_data_start
+ source.section_header[SRC_DATA_SECTION].Misc.VirtualSize)
{
    data_offset = *value - source.pe_header->OptionalHeader.ImageBase
    - source.section_header[SRC_DATA_SECTION].VirtualAddress;

    *value = target_data_start +
    target.section_header[SRC_DATA_SECTION].Misc.VirtualSize+data_offset;
}
```

4.2.3 Volanie `DllMain()`

Vieme už, že funkcia `DllMain()` slúži ako vstupný bod vkladaného kódu, ktorý sa spustí pred pôvodnou aplikáciou. Existuje viacero spôsobov ako v prostredí architektúry IA-32 odovzdať vykonávanie kódu funkcii. V tomto prípade je použitá konštrukcia `PUSH-RETN` z dôvodu jednoduchej implementácie:

```
     01 ...  ;pôvodný kód hostiteľskej aplikácie
EP-> 02 PUSH <pôvodný vstupný bod (EP)>          \
     03 ...  ;telo funkcie DllMain()              | vložené
     04 RETN ;skok na EP hostiteľskej aplikácie   /  dáta
```

Vykonávanie programu začne na riadku `02`, kde leží inštrukcia `PUSH`. Tá uloží na zásobník adresu pôvodného EP hostiteľskej aplikácie. Potom sa začne vykonávať telo funkcie `DllMain()`, ktoré končí inštrukciou `RETN`. Posledná inštrukcia `RETN` vyberie zo zásobníka pôvodný EP a vykoná skok na túto adresu.

Implementácia tejto konštrukcie teda spočíva v pridaní jednej inštrukcie `PUSH` za telo kódu pôvodnej aplikácie a pred telo funkcie `DllMain()`.

4.3 Použitie nástroja `pein`

Nástroj `pein` je určený pre použitie v prostredí príkazového riadku. Zdrojový súbor typu DLL musí byť skompilovaný s ohľadom na použitú techniku (explicitné linkovanie, absencia štandardnej knižnice atď). Cieľový súbor musí byť 32-bitový PE spustiteľný súbor.

`pein` [*-h*] *zdrojový_súbor cieľový_súbor*

-h : Výpis nápovedy.

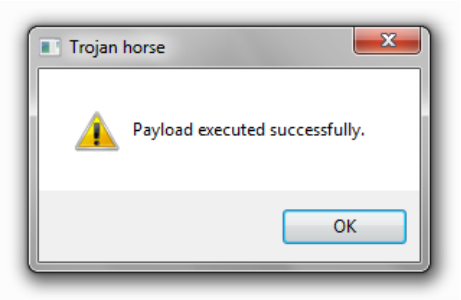
zdrojový_súbor : Zdrojový súbor (*.dll).

cieľový_súbor : Cieľový súbor (*.exe).

Výstupný súbor má prednastavený názov `out.exe`. Činnosť toho nástroja budeme demonštrovať na aplikácii `NOTEPAD.EXE` (poznámkový blok), ktorá je štandardnou súčasťou systému Windows:

```
C:\demo>pein payload.dll NOTEPAD.EXE
File was successfully infected.
```

Pred spustením výslednej aplikácie sa zobrazí dialógové okno payloadu (viz obr. 4.2). Po kliknutí na OK alebo po zavretí okna sa spustí pôvodná aplikácia. S aplikáciou je možné ďalej pracovať a vložený kód zostáva až do ukončenia aplikácie pasívny.



Obr. 4.2: Payload vložený pomocou nástroja pein

Pre otestovanie reakcie antivírusového softwaru na našu aplikáciu využijeme webovú službu `virustotal.com`. Ako môžeme vidieť na obr. 4.3, pozitívnu zhodu oznámil iba antivírusový software *Jiangmin*.

Toto zistenie dokazuje, že technika samotná nie je súčasným antivírusovým softwarom detekovateľná. Jedným z dôvodov, prečo žiadne z popredných antivírusových riešení neoznačilo túto aplikáciu za podozrivú, je marketingová politika. Úspešný antivírusový software nemôže užívateľa zaťažovať hláseniami o každom podozrivom súbore, pokiaľ nepovažuje riziko za dostatočné.

V každom prípade software určený pre bezpečnostných analytikov (profesionálov) by mal disponovať možnosťou dôkladnejšej analýzy, ktorou sa budeme zaoberať v nasledujúcej kapitole.

SHA256:	04fbbbd20de7f7b6ed613398d0dddf1254667faf659ecf02e23605974a964c3c
File name:	out.exe
Detection ratio:	1 / 42
Analysis date:	2012-05-06 19:17:54 UTC (16 minút ago)

More details

Antivirus	Result	Update
AhnLab-V3	-	20120506
AntiVir	-	20120506
Antiy-AVL	-	20120506
Avast	-	20120506
AVG	-	20120506
BitDefender	-	20120506
ByteHero	-	20120505
CAT-QuickHeal	-	20120505
ClamAV	-	20120506
Commtouch	-	20120506
Comodo	-	20120506
DrWeb	-	20120506
Emsisoft	-	20120506
eSafe	-	20120506
eTrust-Vet	-	20120504
F-Prot	-	20120506
F-Secure	-	20120506
Fortinet	-	20120506
GData	-	20120506
Ikarus	-	20120506
Jiangmin	Trojan/JboxGeneric.eqq	20120506
K7AntiVirus	-	20120505
Kaspersky	-	20120506
McAfee	-	20120506
McAfee-GW-Edition	-	20120506

Obr. 4.3: Výsledok analýzy na stránke `virustotal.com`.

Kapitola 5

Možnosti analýzy a detekcie

Malware môže spôsobiť významné škody nielen v súkromnom sektore, ale i v oblasti biznisu. Rôzne spoločnosti investujú nemalé prostriedky do bezpečnostného softwaru, s cieľom zabrániť prieniku do firemnej infraštruktúry. Prácou ľudí zaoberajúcich sa analýzou malwaru je nielen zabrániť, ale i predchádzať opätovným útokom.

V poslednej kapitole tejto práce sa budeme venovať technikám analýzy spustiteľných súborov a detekcii prípadnej infekcie. Touto oblasťou sa budeme zaoberať s ohľadom na techniky infekcie uvedené v kapitole 4. Zoznámime sa s technikami, ktoré sa dnes pri analýze bežne používajú a na záver kapitoly sa oboznámime s možnosťami prevencie a ochrany spustiteľných súborov.

Medzi najčastejšie ciele podrobnej analýzy malwaru patrí[10]:

- Odhalenie zraniteľnosti, pomocou ktorej sa malware dostal do systému.
- Zistenie rozsahu poškodenia a identifikácia napadnutých počítačov.
- Oprava bezpečnostných chýb, slabých miest systému a prípadné určenie spôsobov a signatúr, na základe ktorých je možné malware identifikovať.

5.1 Pracovné prostredie

Pokiaľ analyzujeme neznámy malware, je bezpodmienečne nutné pre tento účel vytvoriť bezpečné a izolované prostredie. Za bezpečné prostredie v tomto prípade považujeme taký počítač, alebo počítačovú sieť, ktorá neobsahuje žiadne citlivé alebo nezálohované dáta, nie je pripojená k internetu atď. Zároveň môže byť nutné sprístupniť malwaru falošné prostriedky (sieťové, súborové), aby sme mohli sledovať jeho činnosť. Riešením týchto požiadaviek sú dve možnosti:

Skutočný systém – skutočná počítačová sieť, ktorá je izolovaná od iných sietí. Stav tejto siete, teda zálohy jednotlivých počítačov je možné obnoviť pomocou vhodného softwaru (Norton Ghost atď). Nevýhodou tohoto prístupu je komplikovaný proces obnovy po jednotlivých experimentoch.

Virtuálny systém – virtuálna počítačová sieť vytvorená pomocou vhodného nástroja (VMWARE, Microsoft Virtual PC, Oracle VM VirtualBox atď). Výhodou je predovšetkým rýchlejšia obnova systému a väčšia flexibilita. Prácu zjednodušuje i mechanizmus snímok (snapshots)[1]. Nevýhodou je fakt, že niektoré druhy malwaru môžu zmeniť svoje chovanie, pokiaľ zistia, že bežia na virtuálnom stroji, alebo zneužiť bezpečnostnú chybu VM softwaru a rozšíriť sa mimo virtuálne prostredie.

V praxi sa častejšie stretneme s použitím virtuálnych systémov pre analýzu malwaru. Väčšina malwaru využíva z rôznych príčin pripojenie k internetu. Pokiaľ analyzujeme tento typ malwaru, môžeme v sieti vytvoriť falošný server.

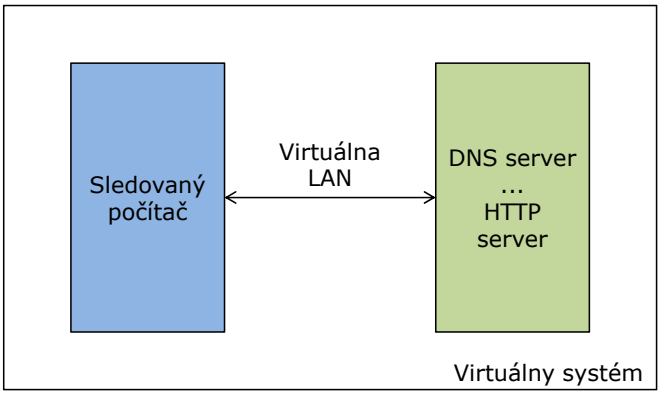
K tomuto účelu v jednoduchších prípadoch stačí jeden virtuálny počítač, úprava súboru `hosts` a vytvorenie serveru na localhoste pomocou nástroja `nc` (Netcat). Predpokladajme, že chceme zachytiť HTTP požiadavok na adresu `domena.com`. Obsah súboru `hosts` bude v tomto prípade nasledujúci:

```
# This is an example of the hosts file ...
127.0.0.1  domena.com
```

TCP server počúvajúci na porte 80 vytvoríme príkazom:

```
nc -l 80
```

Taktiež môžeme vytvoriť dva virtuálne počítače, pričom jeden z nich bude slúžiť ako server (viz obr. 5.1). V tomto prípade môžeme analyzovať i zložitejšiu komunikáciu. Obdobným spôsobom môžeme vybudovať ďalšie počítačové siete a analyzovať tak ľubovoľnú komunikáciu.



Obr. 5.1: Základné sieťové usporiadanie pre analýzu malwaru

5.2 Statická analýza

Prvým krokom analýzy malwaru je statická analýza. Techniky statickej analýzy pracujú s PE súbormi, pričom program obsiahnutý v PE súboroch nebeží. Úlohou tejto analýzy je získať prehľad o štruktúre programu a pokúsiť sa zistiť jeho funkciu.

[1] Umožňuje ukladať viacero stavov systému a v prípade potreby obnoviť vybraný stav.

5.2.1 Skenovanie antivírusovým softwarom

Táto jednoduchá metóda nám umožní zistiť, či je malware v databáze poskytovateľa antivírusového softwaru. Na identifikáciu malwaru používa antivírusový software databázu signatúr[2] a kombináciu iných metód (heuristika, algoritmická detekcia, emulácia atď).

V prípadoch kedy nepracujeme zo známym malwarom, môžeme vytvoriť vlastnú signatúru (pokiaľ je možné malware týmto spôsobom detekovať) a využiť ju pri detekcii. Môžeme využiť napríklad nasledujúce riešenia:

virustotal.com – webová služba, ktorá umožňuje analyzovať odoslaný spustiteľný súbor pomocou antivírusových produktov (momentálne 43). Obsahuje tiež verejné API pre použitie v skriptoch.

ClamAV – open source antivírusový skener, ktorý umožňuje vytvárať vlastné signatúry a používať ich spolu s dodávanou databázou malwaru.
Je dostupný na adrese: `http://www.clamav.net/`.

YARA – nástroj zameraný na výskumníkov malwaru, určený k identifikácii a klasifikácii vzoriek malwaru. Umožňuje definovať komplexné signatúry a flexibilne ich používať.
Je dostupný na adrese: `http://code.google.com/p/yara-project/`.

V sekcii 4.1.1 bol predstavený kód payloadu a bol zároveň použitý pri demonštrácii funkcie programu `pein`. Na detekciu tohoto payloadu môžeme využiť nástroj `yara` (viz kód 5.1).

Kód 5.1 Signatúra payloadu pre nástroj `yara`

```
//file payload.yara
rule DemoTrojanPayload
{
  meta:
    description = "Detection of trojan horse payload."
  strings:
    $code = {68[4]68[4]68[4]FF15[4]50FF15[4]683010000068[4]68[4]6A00FFD0C3}
    $str1 = "Trojan horse"
    $str2 = "Payload executed successfully."
  condition:
    $code at entrypoint and $str1 and $str2
}
```

Reťazec `$code` obsahuje binárny kód extrahovaný z výstupného súboru nástroja `pein` pomocou debuggeru OllyDbg (`http://www.ollydbg.de/`). Zápis `68[4]` reprezentuje bajt s hodnotou `0x68` nasledovaný štyrmi bajtmi (ľubovolná adresa) a je ekvivalentný častejšiemu zápisu `68????????`. Podmienka na konci pravidla `DemoTrojanPayload` určuje, že detekcia nastane iba v prípade, ak sa kód nachádza na vstupnom bode programu a skenovaný súbor zároveň obsahuje reťazce `$str1` a `$str2`. Nástroj `yara` môžeme použiť na rekurzívne skenovanie ľubovolného priečinku nasledovne:

```
C:\demo>yara -r payload.yara .
DemoTrojanPayload .\out.exe
```

[2] Jedinečný binárny reťazec, ktorý jednoznačne identifikuje malware.

Pri vytváraní signatúr malwaru je dôležité zvoliť jedinečný a dostatočne dlhý úsek (alebo viacero) programu, aby sme sa vyhli falošným hláseniam. Tento úsek by však nemal byť dlhší ako je nevyhnutné.

Taktiež je potrebné si uvedomiť, že uvedené pravidlo sa dá aplikovať iba na konkrétny payload. Zmena i jediného bajtu payloadu môže spôsobiť, že nový payload nebude možné detektovať. Technika infekcie použitá v nástroji `pein` nevkladá žiadny dodatočný kód. Preto samotnú techniku infekcie nie je možné detekovať pomocou skenovania.

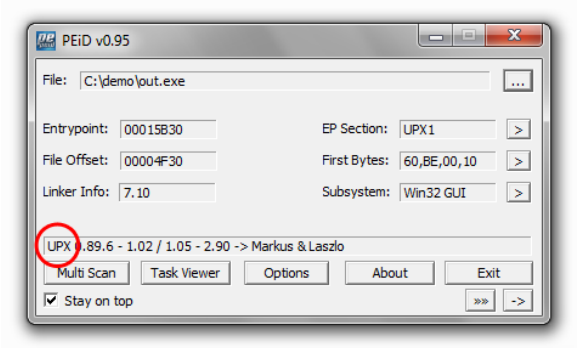
5.2.2 Komprimovaný malware

Autori malwaru využívajú komprimáciu spustiteľných súborov, aby sťažili alebo znemožnili ich analýzu. Nástroje ako UPX[3] a podobné nielenže zmenšia veľkosť spustiteľného súboru, ale zároveň znemožnia analýzu pôvodných sekcií (výsledok podobný šifrovaniu). Všetky techniky analýzy preto nemusia byť použiteľné na komprimované PE súbory.

Niektoré nástroje umožňujúce komprimáciu, disponujú i možnosťou dekomprimácie. Identifikovať použitý nástroj nám umožní napríklad program PEiD (jeho vývoj je už zastavený). Jeho použitie na výstupný súbor programu `pein` skomprimovaný pomocou UPX môžeme vidieť na obr. 5.2. Malware skomprimovaný pomocou nástroja UPX môžeme dekomprimovať pomocou programu CFF Explorer (`http://www.ntcore.com/exsuite.php`) alebo príkazom:

```
upx -d [komprimovaný program]
```

Nástroj PEiD je dostupný na adrese: `http://www.peid.info/`.



Obr. 5.2: Nástroj PEiD identifikoval UPX kompresiu

Autor malwaru môže taktiež použiť modifikovaný alebo vlastný komprimačný algoritmus. O použití komprimácie svedčí viacero faktorov (nezrozumiteľné sekcie, málo importov). Jedným z nich je úroveň *informačnej entropie*[4], ktorá sa označuje ako H, ďalej nech S je konečná množina všetkých stavov systému $S \in \{s_1, s_2, ..., s_n\}, n \leq \infty$ s rozdelením pravdepodobnosti $P(s_i)$. Potom informačnú entropiu vypočítame podľa vzorca

[3] Obľúbený komprimačný nástroj dostupný na adrese: `http://upx.sourceforge.net/`.
[4] Stredná hodnota informácie jedného kódovaného znaku.

$$H(S) = -\sum_{i=1}^{n} P(s_i) \log_2 P(s_i). \tag{5.1}$$

Ak počítame informačnú entropiu sekcie v PE súbore, počítame ju zo sekvencie bajtov. Bajt reprezentuje stav systému a na základe početnosti konkrétnej hodnoty v sekvencii, vieme vypočítať jej pravdepodobnosť. Pre výslednú entropiu platí $H(S) \in \langle 0, 8 \rangle$. Čím je hodnota vyššia, tým je pravdepodobnejšie, že sekcia je komprimovaná. Empirické zistenia ukazujú, že za podozrivú môžeme považovať úroveň 7,174 a viac[12].

5.2.3 Analýza reťazcov

Reťazce obsiahnuté v PE súbore môžu veľa napovedať o jeho funkcii. Bežne sa používa viacero typov reťazcov. Najčastejšie sa môžeme stretnúť s reťazcami ASCII (1 bajt na znak) alebo s reťazcami Unicode (obvykle 2 bajty na znak) označovanými aj ako *wide strings*. Existuje viacero aplikácii, ktoré umožňujú zobraziť dáta v PE súbore považované za reťazce. Ďalšia možnosť je použiť automatickú analýzu napríklad pomocou programu `yara`. Vhodné ukážky nájdeme v knihe *Malware Analyst's Cookbook and DVD*[11].

5.2.4 Heuristika

Pomocou techniky nazývanej heuristika môžeme nájsť podozrivé charakteristiky PE súboru a odhaliť tak i doteraz neznámy malware. Pokročilá heuristika môže využívať i emuláciu kódu a analyzovať chovanie potenciálneho malwaru. V tejto sekcii sa zameriame na statickú heuristiku. Najvýznamnejšie indikátory malwaru sú[20][11]:

EP smeruje do poslednej sekcie – väčšina dostupných linkerov pre OS Windows generuje usporiadanú štruktúru PE súboru a vykonávanie kódu nezačína v poslednej, ale spravidla v prvej sekcii. Techniky infekcie uvedené v sekcii 3.1 tak môžeme úspešne odhaliť týmto spôsobom.

Podozrivé príznaky sekcie – každá sekcia v PE súbore má svoje typické príznaky. Napríklad sekcia dát musí byť čitateľná (zapisovateľná), ale nemusí byť spustiteľná. Netypické príznaky známych sekcií tak indikujú potenciálnu infekciu.

Sekcia s nulovou dĺžkou v súbore – sekcia, ktorá neobsahuje žiadne dáta obvykle slúži k umiestneniu dekomprimovanému kódu za behu programu.

Nesprávna hodnota SizeOfImage – položka SizeOfImage musí vyť zarovnaná na násobok zarovnania sekcie. Nekorektná hodnota tejto položky môže byť dôsledkom infekcie a je považovaná za podozrivú.

Extrémne nízka alebo vysoká entropia – na základe entropie dát môžeme určiť, ktoré sekcie obsahujú komprimované alebo neobvyklé dáta.

Skok do inej sekcie – niektoré techniky infekcie nemodifikujú vstupný bod ale riadenie odovzdajú škodlivému kódu pomocou inštrukcie `JMP` do infikovanej sekcie. Obecne skok medzi jednotlivými sekciami môžeme považovať za podozrivý (hlavne v blízkosti EP).

Podozrivý názov sekcie kódu – typické názvy sekcie kódu sú `.text` a `.code` (prípadne ďalšie špecifické). Na základe zoznamu typických názvov môžeme odhaliť podozrivé PE súbory.

Viacnásobné PE hlavičky – ak sa v PE súbore nachádza viacero PE hlavičiek, môže sa jednať o malware typu dropper (viz sekcia 1.2.3).

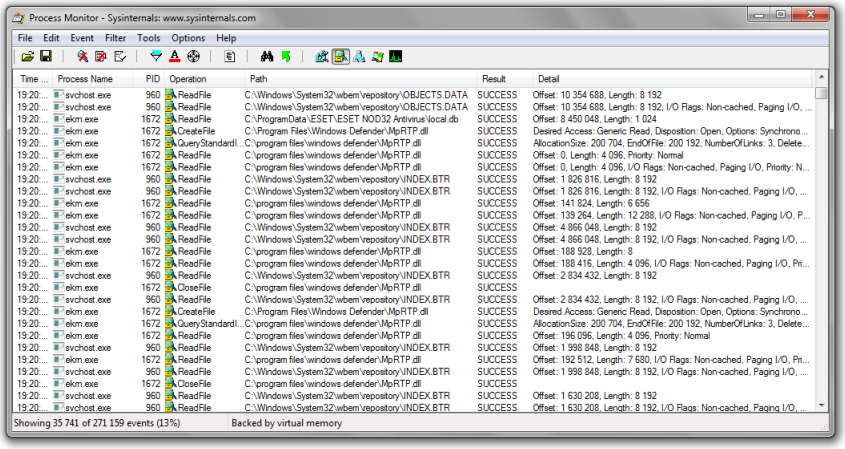
Pri implementácii heuristického analyzátora treba dať pozor na početnosť falošných hlásení. Pokiaľ je heuristický analyzátor určený pre výskum malwaru, môže byť počet hlásení výrazne vyšší ako pri software určenom pre bežného používateľa.

5.3 Dynamická analýza

Druhým krokom analýzy malwaru je dynamická analýza. Na rozdiel od statickej analýzy sa vykonáva po spustení malwaru (za jeho behu). Dynamická analýza je tak obecne menej bezpečná a zložitejšia ako statická analýza, pri analýze komplexného malwaru však poskytuje lepšie výsledky. Jej úlohou je zistiť skutočnú funkciu malwaru a jeho vplyv na sledovaný systém. S použitím vhodného softwaru zistíme aké súbory malware vytvára, akým protokolom komunikuje po sieti, aké algoritmy používa atď.

5.3.1 Monitorovanie súborov a registrov

Vhodný nástroj pre monitorovanie zásahov malwaru do systému je nástroj Process Monitor. Umožňuje zobraziť a filtrovať udalosti v systéme. Predovšetkým aktivity v súborovom systéme, v registroch a v sieti (viz obr. 5.3). Jednotlivé udalosti dokáže filtrovať na základe viacerých parametrov. Napríklad názov procesu a typ udalosti.



Obr. 5.3: Process Monitor s filtrovaním udalostí

Počet udalostí v OS Windows sa pohybuje v rádoch stoviek za sekundu. Pretože Process Monitor eviduje všetky udalosti a filtrovanie sa uplatní iba pri ich zobrazení, je dôležité udržovať ich počet v prijateľnej miere. Nástroj je zdarma dostupný na adrese: `http://technet.microsoft.com/en-us/sysinternals/bb896645`.

5.3.2 Monitorovanie siete

Analýza sieťovej činnosti a použitého protokolu nám nielenže priblíži jeho funkciu, ale je vhodná i pre vytváranie signatúr pre IDS[5] systémy. Pre monitorovanie siete môžeme použiť open source nástroje typu tcpdump, Wireshark (`http://www.wireshark.org/`) atď.

Zvládnutie analýzy sieťovej komunikácie patrí v dnešnej dobe k základným predpokladom úspešnej analýzy malwaru. Pri experimentovaní s malwarom a analýze jeho sieťovej komunikácie sa prejaví kvalita návrhu pracovného prostredia. Musíme byť schopní simulovať všetky služby, ku ktorým malware pristupuje a obmedziť sieťovú komunikáciu na minimum. Toto nám zjednoduší orientáciu vo výpise komunikácie (okrem iného Wireshark neumožňuje filtrovanie podľa názvu procesu).

5.3.3 Debugger

Debugger je software alebo hardware používaný k testovaniu alebo skúmaniu behu iného programu. Techniky analýzy, ktorými sme sa doteraz zaoberali, nám umožňujú získať podrobný prehľad o skúmanom malware. Ale až použitie debuggeru nám umožní skutočný pohľad do jeho vnútra.

Väčšina programátorov je oboznámená so *source-level* debuggermi, tie sa používajú pri vývoji aplikácií a umožňujú sledovať beh programu na úrovni (vysokoúrovňového) zdrojového kódu. Použitie tohoto typu debuggeru vyžaduje, aby bol sledovaný program preložený tak, aby obsahoval ladiace informácie.

Pri analýze malwaru sa takmer výhradne používajú *assembly-level* debuggery. Tento typ debuggeru nepotrebuje prístup k zdrojovému kódu skúmaného programu, ale zobrazuje ho na úrovni jednotlivých strojových inštrukcií (assembler).

Debuggery ďalej delíme na[17]:

Debuggery v režime jadra – tento typ obvykle vyžaduje 2 OS. Výnimkou bol program SoftICE, ten ale nie je podporovnaý od roku 2007. V súčasnosti je možné použiť WinDbg.

Debuggery v užívateľskom režime – v tomto režime debugger beží pod rovnakým OS ako sledovaný program. Tento režim podporujú debuggery OllyDbg, WinDbg alebo IDA Pro.

Software IDA Pro slúži primárne ako disassembler, ktorý sa používa pri statickej analýze.

5.3.4 Sandbox

Sandbox (tiež Sand-Boxing) je bezpečnostný mechanizmus, ktorý umožňuje beh nedôveryhodného programu v bezpečnom prostredí – virtuálny subsystém v rámci OS. V tomto prípade nehrozí poškodenie reálneho systému. Aktivity vykonané programom v tomto prostredí sú zaznamenané.

[5] Obranný systém, ktorý monitoruje sieťovú prevádzku a snaží sa odhaliť podozrivé aktivity.

Medzi najznámejšie sandboxy patria Norman SandBox, GFI Sandbox, Anubis, Joe Sandbox, ThreatExpert atď. Niektoré je možné použiť pre nekomerčné použitie bezplatne alebo poskytujú bezplatné webové rozhranie. Výsledkom analýzy pomocou sandboxu je výstupný log s rôznou rozsiahlosťou.

Medzi nevýhody sandboxov patrí nemožnosť interakcie s malwarom (malware je spustený bez parametrov), obmedzená doba behu malwaru, nemožnosť nastavenia prostredia a možná nekompatibilita malwaru s OS sandboxu.

5.4 Prevencia a generické techniky

Počet existujúceho malwaru od jeho vzniku neustále stúpa. V dobách, keď existovalo len niekoľko desiatok prvých vzoriek, nebol problém venovať ich analýze celé týždne. Napriek silnej automatizácii tejto činnosti v dnešnej dobe, vyžaduje analýza malwaru veľa času a finančných zdrojov. Aj z týchto dôvodov sa neustále hľadajú nové spôsoby a prístupy k boju s malwarom.

V tejto sekcii sa pozrieme na prístupy, ktorých úlohou nie je identifikovať konkrétny druh malwaru, ale predovšetkým zabrániť poškodeniu systému, či identifikovať celé rodiny malwaru. Spoločným znakom niektorých techník je to, že prenechávajú väčšiu zodpovednosť užívateľovi a vyžadujú väčšiu dôslednosť pri práci s OS.

5.4.1 Viacvrstvový bezpečnostný model

Dôsledné riadenie prístupu a zmysluplné používanie užívateľských práv môže výrazne zmierniť škody spôsobené malwarom alebo im úplne zabrániť. Zaujímavou implementáciou tohoto prístupu je bezpečnostný model v OS Android. Za zmienku stojí beh každej aplikácie pod vlastným UID – riadenie prístupu k zdrojom a dátam. Tento princíp je prezentovaný ako sandbox v rámci každej aplikácie, ktorý tak umožňuje ich izoláciu[7].

Prideľovanie práv (čo aplikácia môže a čo nie) prebieha pri inštalácii a zaniká odinštalovaním tejto aplikácie. Tento prístup sa líši napríklad od OS Windows, kde je súhlas užívateľa typicky vyžadovaný pri každom spustení programu. Viacero štúdií užívateľského rozhrania preukázalo, že opakované vyžadovanie podobného alebo rovnakého potvrdenia od užívateľa spôsobí, že užívateľ sa o ne prestane zaujímať a začne ich potvrdzovať bez čítania[7].

5.4.2 Kontrola integrity

Táto technika je typickým zástupcom skupiny generických techník. Dokáže detektovať a zabrániť zmenám v súboroch na základe kontrolného súčtu súboru (CRC, MD5, SH1 atď). Pred použitím tejto techniky je typicky vytvorená databáza kontrolných súčtov na chránenom systéme alebo online. Na základe tejto databázy je možné zistiť zmeny známych súborov.

Zaujímavé je použitie podobného princípu vo firemnom prostredí. Existujú bezpečnostné softwarové riešenia, ktoré umožňujú centrálnu správu povolených aplikácií. Na každej pra-

covnej stanici je tak možné použiť iba ten software, ktorý schválil dodávateľ bezpečnostného riešenia, administrátor alebo firemný bezpečnostný analytik (princíp white listu).

5.4.3 Podpisovanie kódu

Tento mechanizmus využíva asymetrickú kryptografiu k digitálnemu podpísaniu softwaru jeho autorom. Je teda možné jednoducho overiť, či bol software od odoslania autorom pozmenený a či skutočne pochádza od uvedenej osoby. Podpis je možné vytvoriť buď len na základe súkromného kľúča, alebo na základe certifikátu vydaného certifikačnou autoritou.

Samotná prítomnosť digitálneho podpisu však neznamená, že software nemôže obsahovať malware. Ďalším problémom je skutočnosť, že OS Windows (vo verzii 7) na prítomnosť neplatného podpisu pri spustení neupozorňuje. To, že je podpis neplatný, užívateľ zistí len vo vlastnostiach súboru.

5.4.4 Behaviorálna detekcia

Jedná sa o relatívne novú techniku, ktorá sa pomaly začína presadzovať v niektorých antivírusových riešeniach. Je založená na analýze systémových volaní a vychádza zo zistenia, že autor malwaru nemôže jednoducho zmeniť poradie systémových volaní bez zmeny sémantiky kódu[6]. Na základe behaviorálnych signatúr je preto možné detekovať i príbuzný malware.

Záver

V tejto práci boli analyzované štyri techniky infekcie spustiteľných súborov v OS Windows. Predmetom analýzy boli predovšetkým špecifiká implementácie, obmedzenia vloženého kódu a obtiažnosť detekcie. Pri tejto analýze bolo zistené, že súbory obsahujúce sekciu `.reloc` sú náchylnejšie k infekcii, pri ktorej dochádza k zväčšovaniu jednotlivých sekcií.

Spolu s nástrojom `pein` bol predstavený univerzálny koncept infekcie. Tento koncept demonštruje, že je možné payload vyvíjať v jazyku C a s použitím vhodného rozhrania ho integrovať do cieľovej aplikácie. Nástroj `pein` implementuje techniku dutinovej infekcie. Modul `PEimage.h` nástroja `pein` obsahuje rozhranie pre prácu s PE súbormi, ktoré môže byť použité i pri analýze PE súborov.

Výstupný súbor nástroja `pein` bol skenovaný pomocou webovej služby `virustotal.com` (43 antivírusových produktov). Bol nájdený jediný pozitívny nález, čím sme dokázali, že väčšina súčasných antivírusových produktov nedetekuje techniku dutinovej infekcie (tak ako bola predstavená v kapitole 3). V poslednej kapitole tejto práce boli podrobnejšie diskutované možnosti detekcie malwaru na základe použitej techniky infekcie.

Práca by mohla ďalej pokračovať vývojom softwaru pre analýzu malwaru, ktorý bude implementovať techniky detekcie uvedené v poslednej kapitole. Ďalšou možnosťou je napríklad rozšírenie rozhrania nástroja `pein` tak, aby umožňoval implementovať ochranu spustenia programu heslom, šifrovanie, komprimáciu programu atď.

Literatúra

[1] Trojan horse (computing) [online]. http://en.wikipedia.org/wiki/Trojan_horse_(computing), 2012-02-29 [cit. 2012-03-04].

[2] AEGISLAB: Distribution of Malware Types in 2011 [online]. http://blog.aegislab.com/index.php?op=Default&postCategoryId=1&blogId=1, 2012 [cit. 2012-03-03].

[3] Bates, J.: Trojan Horse: AIDS Information Introductory Diskette Version 2.0. *Virus Bulletin*, 1990-01: str. 3.

[4] Bauch, J.: Loading a DLL from memory [online]. http://www.joachim-bauch.de/tutorials/loading-a-dll-from-memory/, 2010 [cit. 2012-03-15].

[5] Danehkar, A.: Injective Code inside Import Table [online]. http://ntcore.com/files/inject2it.htm, 2007 [cit. 2012-03-12].

[6] Forrest, S.; Hofmeyr, S. A.; Somayaji, A.; aj.: A sense of self for Unix processes. *Security and Privacy, Symposium on*, 1996-05: s. 120–128.

[7] GOOGLE: Android Security Overview [online]. http://source.android.com/tech/security/index.html, 2012 [cit. 2012-04-13].

[8] Ing. Filip Orság, P.: Pokročilé asemblery (IPA) [online]. http://www.fit.vutbr.cz/study/course-l.php.cs?id=7526, 2012 [cit. 2012-03-12].

[9] INTEL: Intel 64 and IA-32 Architectures Software Developer Manuals [online]. http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html, 2011 [cit. 2012-03-06].

[10] Kendall, K.: PRACTICAL MALWARE ANALYSIS [online]. http://www.blackhat.com/presentations/bh-dc-07/Kendall_McMillan/Paper/bh-dc-07-Kendall_McMillan-WP.pdf, 2007 [cit. 2012-04-03].

[11] Ligh, M. H.; Adair, S.; Hartstein, B.; aj.: *Malware Analyst's Cookbook and DVD.* Wiley, 2010, iSBN 0470613033.

[12] Lyda, R.; Hamrock, J.: Using Entropy Analysis to Find Encrypted and Packed Malware. *IEEE Security & Privacy*, ročník 5, 2009-07: s. 40–45.

[13] Marek, R.: *Učíme se programovat v jazyce Assembler pro PC*. Computer Press, 2003, iSBN 80-722-6843-0.

[14] MICROSOFT: An In-Depth Look into the Win32 Portable Executable File Format (MSDN magazine) [online]. `http://msdn.microsoft.com/en-us/magazine/cc301805.aspx`, 2002 [cit. 2012-03-12].

[15] MICROSOFT: PE Structures (MSDN magazine) [online]. `http://msdn.microsoft.com/en-us/magazine/bb985997.aspx`, 2004 [cit. 2012-03-08].

[16] MICROSOFT: Microsoft PE and COFF Specification [online]. `http://msdn.microsoft.com/en-us/windows/hardware/gg463119`, 2010 [cit. 2012-03-12].

[17] Sikorski, M.; Honig, A.: *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, 2012, iSBN 1593272901.

[18] Sotirov, A.: Tiny PE [online]. `http://www.phreedom.org/research/tinype/`, 2006 [cit. 2012-03-19].

[19] SYMANTEC: Symantec Internet Security Threat Report: Trends for July–December 2007 (Executive Summary) [online]. `http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_exec_summary_internet_security_threat_report_xiii_04-2008.en-us.pdf`, 2008-04-29 [cit. 2012-03-03].

[20] Szor, P.: *Počítačové víry – analýza útoku a obrana*. ZONER SOFTARE, 2006, iSBN 80-86815-04-8.

Dodatok A

Obsah CD

Priložené CD obsahuje nasledujúce súbory a priečinky:

- `Payload-source/` – Zdrojové kódy payloadu
- `Pein-source/` – Zdrojové kódy nástroja `pein`
- `bakalarska-praca.pdf` – Bakalárska práca vo formáte PDF
- `NOTEPAD.EXE` – Testovací spustiteľný súbor
- `payload.dll` – Skompilovaný payload
- `payload.yara` – Signatúra na detekciu payloadu
- `pein.exe` – Skompilovaný nástroj `pein`