KAPITOLA 7

Pokročilé techniky vývoje kódu a generátory počítačových virů

"V matematice nemůžete porozumět věcem. Můžete je pouze používat."

– John von Neumann

V této kapitole se dozvíte o vylepšených obranných technikách, které autoři počítačových virů za dlouhá léta vyvinuli, aby odrazili útok antivirových skenerů.

Podrobně se dozvíte o zakódovaných, oligomorfních, polymorfních[1] a vylepšených metamorfních počítačových virech[2]. Nakonec se podíváme na generátory[3] počítačových virů, které používají podobné techniky k vytvoření odlišně vypadajících variant virů.

7.1 Úvod

Zde prozkoumáme různé cesty, po kterých se autoři virů za poslední dekádu vydali, aby porazili antivirové skenery. Ačkoliv byla většina těchto technik použita na zamaskování souborových infektorů, můžeme s jistotou očekávat, že se tyto techniky objeví v budoucích počítačových červech.

Za celá ta léta vývoje ušly binární viry velmi dlouhou cestu. Pokud by se někdo vydal po stopách výsledků vývoje virů, došel by k přesvědčení, že takřka všechno myslitelné už bylo učiněno a že se problémy s viry nebudou zvyšovat. Zbývají modely distribuovaných výpočtů, které ještě ve virech spatřeny nebyly.

7.2 Vývoj virového kódu

Autoři virů neustále bojují s antivirovými produkty. Jejich největším nepřítelem jsou nejpopulárnější antivirové skenery. Generická antivirová řešení, jako třeba kontrolory integrity dat a monitory blokující podezřelé chování (behavior-blockers) se (narozdíl od nich) nikdy neměla za cíl stát tak populárními.

Ve skutečnosti takové generické modely detekce virů vyžaduji na systémech Windows mnohem větší promyšlení návrhu, protože tyto technologie byly na systémech DOS několikrát překonány DOSovými viry. Výsledkem je, že si mnoho lidí začalo myslet, že tyto techniky jsou k ničemu.

Skenování je řešení, které trh přijal, bez ohledu na jeho nevýhody. Musí tedy čelit rostoucí složitosti a vzrůstajícímu počtu škodlivého softwaru, který se navíc dokáže sám rozšiřovat.

Ačkoliv se moderní výpočetní technika výrazně zrychlila, po dlouhý čas se daly binární kódy virů současnými technikami stěží dohonit. DOSové viry se do roku 1996 vyvíjely do velmi složitých úrovní, od té doby začaly na trhu dominovat 32bitové systémy Windows. Výsledkem bylo, že se autoři virů ve vývoji binárních virů vrátili o několik let zpět. Složitost polymorfismu v DOSu vyvrcholila v roce 1996, když byl představen virus Ply s novým permutačním enginem (metamorfní virus ACG byl představen v roce 1998). Tento vývoj už nicméně nemohl dále pokračovat a tak se pionýři v psaní počítačových virů zaměřili na vyvinutí 32bitových technik pro Win32 platformy.

Někteří autoři virů stále chápou platformu Windows jako příliš složitou, zvlášť v případě systémů Windows NT/2000/XP/2003. Základní techniky infekce už nicméně byly k vidění, nehledě na to, že zdrojové kódy samotných virů byly distribuovány na Internetu. Tyto zdrojové kódy poskytují základy pro masově se šířící červy. Navíc nevyžadují nějaké programátorské zkušenosti – pouze schopnost zkopírovat a vložit text.

V následující části prozkoumáme základní matoucí techniky virového kódu, zakódovanými viry počínaje a moderními metamorfními technikami konče.

7.3 Zakódované viry

Už od samého počátku se autoři virů pokoušeli implementovat do kódu virů nějakou evoluci. Jedním z nejjednodušších způsobů k zamaskování fungování virového kódu bylo kódování. Prvním známým virem, který tuto techniku implementoval, byl DOSový Cascade[4]. Virus začíná neměnným decryptorem, po kterém následuje zakódované tělo viru. Podívejte se na ukázku decryptoru viru Cascade.1701, který je zobrazen ve výpise 7.1.

Výpis 7.1

Decryptor viru Cascade.

```
lea     si, Start   ; začátek zakódovaného těla (nastaveno dynamicky)
mov     sp, 0682    ; délka zakódovaného těla (1666 bajtů)
Decrypt:
xor     [si],si     ; první dekódování
xor     [si],sp     ; druhé dekódování
inc     si          ; inkrementace prvního čítače
dec     sp          ; dekrementace druhého
jnz     Decrypt     ; smyčka proběhne tolikrát, než se dekódují všechny bajty
Start:              ; Začátek zakódovaného/dekódovaného těla viru
```

Tento decryptor používá trik proti ladění, protože používá jako jeden z dekódovacích klíčů registr SP (stack pointer, ukazatel na zásobník). Dekóduje se vždy směrem dopředu, registr SI se inkrementuje o jedničku.

Protože registr SI zprvu ukazuje na začátek zakódovaného těla viru, jeho počáteční hodnota závisí na relativní pozici virového těla v souboru. Cascade se připojuje na konec souboru, takže SI bude mít stejnou hodnotu v případě souborů o stejné velikosti. SI (dekryptovací klíč 1) se bude ovšem lišit, pokud mají hostitelské programy různou velikost. Registr SP je prostý čítač počtu bajtů k dekódování – jde tedy směrem dopředu a dekóduje se po slovech (2 bajty), ale místo dekódování se posunuje o jeden bajt. To decryptovací smyčku trochu komplikuje, ale nic nemění na její reverzibilitě. Operace XOR je pro viry velmi praktická, protože "XORování" stejné hodnoty dvakrát vrátí původní hodnotu.

Uvažte kódování písmena P (0x50) klíčem 0x99. Tedy 0x50 XOR 0x99 je 0xC9 a 0xC9 XOR 0x99 vrátí 0x50. Proto mají autoři virů v oblibě tuto techniku zakódování – protože jsou líní! Omezí tak nutnost implementovat dva rozdílné algoritmy, jeden pro zakódování a druhý pro dekódování.

Kryptograficky řečeno – takové kódování je slabé, ačkoliv první antivirové programy měly možnost získat krátký skenovací řetězec ze samotného decryptoru. To ovšem vedlo k několika problémům. Různé viry mohou mít například stejný decryptor, ale kompletně jiné vlastnosti. Detekcí viru podle jeho decryptoru se pak může stát, že antivirus nedovede správně identifikovat příslušnou variantu. A dále – nevirové programy, třeba různé obálky (wrappers) s ochranami proti ladění, mohou mít na začátku svého kódu podobný decryptor. Virus, který používá stejný kód pro svoje dekódování, tak může antivirový program zmást.

Taková jednoduchá metoda vývoje kódu se záhy objevila i v 32bitových virech pro Windows. W95/Mad a W95/Zombie používají stejnou techniku jako virus Cascade, jediný rozdíl spočívá v 32bitové implementaci. Podívejte se na decryptor z viru W95/Mad.2736 ve výpise 7.2.

Výpis 7.2

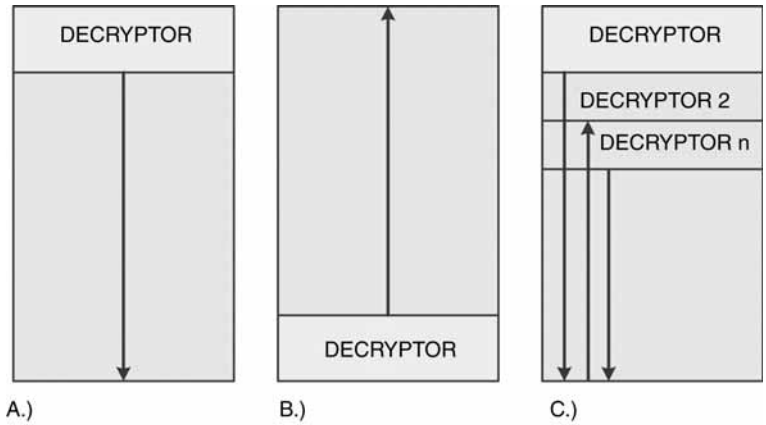
Decryptor viru W95/Mad.2736.

```
mov           edi,00403045h        ; EDI = začátek
add           edi,ebp              ; přidej bázovou adresu
mov           ecx,0A6Bh            ; délka zakódované části těla viru
mov           al,[key]             ; načti klíč
Decrypt:
xor           [edi],al             ; dekóduj jeden bajt těla
inc           edi                  ; inkrementuj místo dekódování
loop          Decrypt              ; dekóduj všechny bajty
jmp           Start                ; Skoč na Start (přes data)
DB            key     86           ; proměnný klíč o velikosti jeden bajt
Start:                             ; zakódované/dekódované tělo viru
```

Jedná se o ještě jednodušší implementaci metody XOR. Detekce takových virů je stále možná bez nutnosti dekódovat tělo viru. Ve většině případů je vzorek kódu decryptoru těchto virů dostatečně unikátní, aby se tyto viry daly podle něj detekovat. Tato detekce sice není exaktní, nicméně stále existuje možnost dekódovat zakódované tělo viru a detekovat i různé varianty.

Útočník může implementovat některé speciální techniky, aby proces kódování a dekódování více zkomplikoval a zmátl detekci, popř. léčení zavirovaných souborů antivirovými programy:

- Směr dekódovací smyčky se může měnit, je možné kódovat směrem dopředu i dozadu (viz obrázek 7.1).
- Použití vícenásobných vrstev kódování. První decryptor dekóduje druhý, ten dekóduje třetí a tak dále (viz obrázek 7.1c). Viry Hare[5] od Demon Emperora, W32/Harrier[6] od TechnoRata, {W32,W97M}/Coke od Vecny a W32/Zelly od ValleZe jsou příklady virů, které používají tuto metodu.
- Některé decryptovací smyčky získávají řízení jedna po druhé s náhodně zvoleným směrem dekódování. Tato technika kód zpřehází nejvíc (viz obrázek 7.1c).
- Existuje pouze jedna decryptovací smyčka, ale používá více než dva klíče k dekódování všech částí informace. V závislosti na implementaci decryptoru mohou být takové viry mnohem obtížněji detekovatelné. Dost záleží na velikosti klíče – čím je klíč větší (8, 16, 32 bitů nebo ještě více), tím déle bude dekódování hrubou výpočetní silou trvat (v případě, že klíče nemohou být jednoduše z viru vyextrahovány).



Obrázek 7.1 *Příklady dekódovacích smyček.*

- Začátek decryptoru je utajen. Mezi decryptorem a zakódovaným tělem nebo zakódovaným tělem a koncem souboru jsou umístěny náhodné bajty.
- Použije se nelineární dekódování. Některé viry, jako třeba W95/Fono používá jednoduchý nelineární algoritmus s tabulkou klíčů. Kódování viru je založeno na substituční tabulce, virus se například rozhodne nahradit písmena A za Z, P za L atd., takže takové slovo APPLE bude po takovém zakódování vypadat jako ZLLPE.

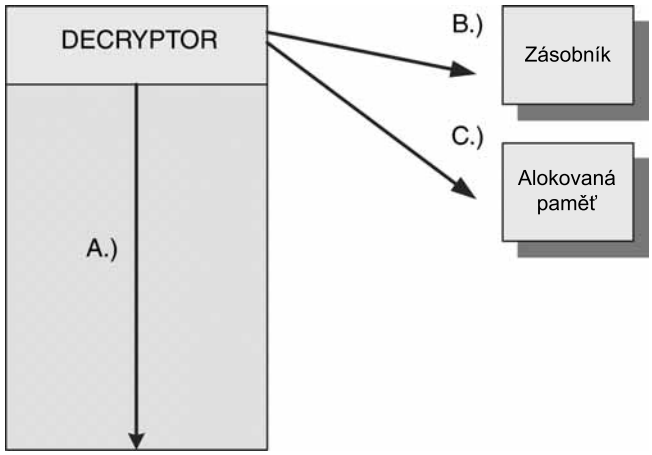
  Protože dekódování viru není lineární, virové tělo se nedekóduje bajt po bajtu. To může jednoduše oklamat virové analytiky-začátečníky, protože virus nemusí vypadat, že je zakódovaný. Pokud se potom použije některý řetězec pro detekci viru, bude detekce fungovat jen částečně. Tato technika dovede přelstít i vylepšené techniky detekce, které používají emulátor. Ačkoliv by v normálním případě emulace pokračovala tak dlouho, dokud by se nenarazilo na lineární dekódování, za což se může považovat postupná změna bajtů v paměti virtuálního stroje skeneru, bude emulace pokračovat dál, dokud se nenarazí na limit.

  Varianta viru W32/Chiton ("Efish") používá podobnou metodu jako Fono, ale vždy se ujistí, že je každý bajt těla viru nahrazen jiným bajtem použitím kompletní substituční tabulky. Chiton navíc pro každý bajt kódu používá různé hodnoty, což znatelně komplikuje proces dekódování.

  Viry jako třeba W95/Drill a {W32,Linux}/Simile.D reprezentují vrchol umění nelineárního zakódování, kde se dekóduje tělo viru v pseudonáhodném pořadí a každé místo v kódu se dekóduje jen jednou[7].
- Útočník se může rozhodnout neukládat kódovací klíč a místo toho se pokusí ke svému dekódování použít hrubou sílu, pomocí které získá kódovací klíče po svém. Takové viry používající RDA techniky (Random Decryption Algorithm, náhodný dekódovací algoritmus) jsou mnohem hůře detekovatelné. Příkladem viru, který používá takovou techniku, je RDA.Fighter.
- Útočník může použít silný algoritmus pro zakódování těla viru. Rodina viru IDEA napsaná autorem, který si říká Spanska, využívá tuto metodu. Jeden z několika decryptorů používá šifru IDEA[8]. Protože virus sebou nese dekódovací klíč, zakódování se nedá považovat za silné, avšak

léčení takto napadnutých souborů je problém, protože antivirové programy musí implementovat kódovací algoritmus, který si s použitým zakódováním poradí. Druhá dekódovací vrstva viru IDEA[9] navíc používá RDA.

- Český virus W32/Crypto od Prizzyho demonstroval použití Microsoft Crypto API v počítačových virech. Crypto za běhu kódoval DLL knihovny na infikovaném systému s použitím dvojice klíčů (soukromý a veřejný klíč). Jiné počítačové červy a backdoory (tzv. zadní vrátka) také používají Crypto API k dekódování zakódovaného obsahu, což činí práci antivirových skenerů obtížnější. Příkladem počítačového červa, který používá Crypto API, je například W32/Qint@mm, který zakódovává EXE soubory.
- Samotný decryptor někdy není součástí viru. Viry jako třeba W95/Resur[10] a W95/Silcer jsou představiteli této metody. Tyto viry donutí zavaděč systému Windows k přesunutí (relocate) infikovaných modulů, jakmile se načítají do paměti. Samotná relokace modulu je zodpovědná za dekódování virového těla, protože do něj virus vložil speciální relokační záznamy. Bázová adresa spustitelného modulu v tomto případě funguje jako kódovací klíč.
- Virus Cheeba demonstroval, jak může být kódovací klíč uložen mimo tělo viru. Cheeba byl vypuštěn v roce 1991 a jeho tzv. payload (aktivační rutina viru) se zakóduje jménem souboru a dekóduje se správně pouze tehdy, když virus přistupuje k souboru[11]. Pokud není šifra viru slabá, antiviroví výzkumníci jednoduše nemohou popsat payload viru. Dmitry Gryaznov zredukoval (s použitím kryptoanalýzy těla viru) velikost klíče potřebného k útoku na šifru Cheeby na pouhých 2150400 možných klíčů, za předpokladu, že zašifrovaný kód je napsaný podobným stylem jako zbytek kódu viru[12]. Výsledkem bylo nalezení správného jména souboru – "user.bss". Tento soubor patřil k populárnímu softwaru pro BBS. Je pravděpodobné, že se časem objeví více takových triků (tzv. "clueless agents"[13]), které znesnadní získávání znalostí o daném nebezpečí, které hrozí ze strany útočníka.
- Kódovací klíče mohou být generovány různými způsoby, např. konstantně, náhodně, fixně, s posunutím atd.
- Samotný klíč může být uložen v decryptoru, v hostiteli nebo nikde. V některých případech kód decryptoru funguje jako dekódovací klíč, což může způsobovat problémy tehdy, pokud debugger nějak poškodil decryptor. Tato technika může rovněž posloužit jako útok na emulátory, které používají techniky optimalizace kódu pro efektivnější běh decryptorů (příkladem takového viru je například Tequila).
- Náhodnost klíče je také důležitým faktorem. Některé viry generují nové klíče pouze jednou za den a označují se jako pomalé generátory. Jiní dávají přednost generování klíčů pokaždé, když infikují soubor – ty se označují jako rychlé generátory. Útočník může k dosažení náhodnosti použít mnoho různých metod. Jednoduché příklady zahrnují časovač nebo datum a čas uložený v CMOS a CRC32. Složitějším příkladem je generátor pseudonáhodných čísel Mersenne Twister[14] použitý ve virech W32/Chiton a W32/Beagle.
- Útočník může označit některá místa jako místa určená dekódování zakódovaného obsahu. Nejčastější metody jsou zobrazeny na obrázku 7.2.



Obrázek 7.2 *Možná místa pro dekódování. A) Decryptor dekóduje data na místě zakódovaného těla viru. Tato metoda je nejběžnější, zakódovaná data musí být ovšem v paměti zapisovatelná, což je věc konkrétního operačního systému. B) Decryptor čte zakódovaný obsah a buduje dekódované tělo viru na zásobníku. To je pro útočníka velmi praktické, protože je zásobník zapisovatelný automaticky. C) Virus alokuje paměť pro dekódované tělo a data. To může být určitá nevýhoda, protože útočník potřebuje před samotným dekódováním prvně alokovat paměť.*

Poznámka

Některé metamorfní viry, jako třeba Simile, tuto nevýhodu vyřešily tak, že kód viru, který se stará o alokaci paměti, je dostatečně variabilní na to, aby se z něj dal vybrat vyhledávací řetězec.

Předchozí techniky fungují velmi efektivně, když se zkombinují s variabilními decryptory, které se postarají o změny v nových generacích viru. V následujících částech jsou probrány oligomorfní a polymorfní decryptory, které se ve virech používají.

7.4 Oligomorfní viry

Autoři virů velmi rychle přišli na to, že se detekce zakódovaných virů pro antivirový software odvíjí od toho, jak dlouhý a jedinečný je samotný decryptor. Aby antivirové produkty překonali, rozhodli se implementovat techniky, které umí vytvářet mutované decryptory.

Narozdíl od zakódovaných virů ty oligomorfní mění v nových generacích svůj decryptor. Nejjednodušší technikou ke změně decryptorů je použití více decryptorů namísto jednoho. Prvním takovým známým virem byl Whale, který si sebou nesl pár tuctů rozdílných decryptorů, přičemž si vždy jeden z nich náhodně vybral.

W95/Memorial měl schopnost vystavět až 96 různých decryptorů. Proto byla detekce viru, která byla založena na hledání kódu decryptoru, nepraktickým, ačkoliv stále použitelným řešením. Většina antivirových produktů se pokusila tento problém řešit dynamickým dekódováním zakódovaného těla, čímž byla ovšem detekce stále založena na konstantním kódu dekódovaného těla viru.

Podívejte se na příklad z viru Memorial zobrazený ve výpisu 7.3, jedná se o jeden z 96ti možných případů decryptoru.

Výpis 7.3

Příklad decryptoru viru W95/Memorial.

```
mov     ebp,00405000h       ; načti bázi
mov     ecx,0550h           ; počet bajtů
lea     esi,[ebp+0000002E]  ; offset "Start"
add     ecx,[ebp+00000029]  ; plus tento počet bajtů
mov     al,[ebp+0000002D]   ; vyber první klíč
Decrypt:
nop                         ; smetí
nop                         ; smetí
xor     [esi],al            ; dekóduj bajt
inc     esi                 ; přesuň ukazatel na další
nop                         ; smetí
inc     al                  ; posuň klíč
dec     ecx                 ; jsou tam nějaké další bajty k dekódování?
jnz     Decrypt             ; skákej, dokud nejsou všechny bajty dekódované
jmp     Start               ; dekódování u konce, spusť tělo
                            ; datová oblast
Start:
; zakódované/dekódované tělo viru
```

Povšimněte si posunu klíče. Pořadí instrukcí může být lehce měněno a decryptor může pro smyčku používat odlišné instrukce.

Porovnejte tento příklad s jiným decryptorem z výpisu 7.4.

Výpis 7.4

Lehce odlišný decryptor viru W95/Memorial.

```
mov     ecx,0550h           ; počet bajtů
mov     ebp,013BC000h       ; načti bázi
lea     esi,[ebp+0000002E]  ; offset"Start"
add     ecx,[ebp+00000029]  ; plus tento počet bajtů
mov     al,[ebp+0000002D]   ; vyber první klíč
Decrypt:
nop                 ; smetí
```

```
nop                 ; smetí
xor     [esi],al    ; dekóduj bajt
inc     esi         ; přesuň ukazatel na další bajt
nop                 ; smetí
inc     al          ; posuň klíč
loop    Decrypt     ; skákej dokud nejsou všechny bajty dekódované
jmp     Start       ; dekódování u konce, spusť tělo
                    ; datová oblast
Start:
;     zakódované/dekódované tělo viru
```

Všimněte si výskytu instrukce "loop" a prohození instrukcí na začátku decryptoru. Virus je oligomorfní tehdy, pokud je schopný měnit svůj decryptor (v omezeném rozsahu).

Je zajímavé, že některé produkty, které jsme testovali, nebyly schopny zachytit všechny instance viru Memorial. To proto, že pro nalezení a pochopení generátoru oligomorfního decryptoru je potřeba prozkoumat takové viry do nejmenších detailů. Bez patřičné ruční analýzy se nedají pomalé oligomorfní viry spolehlivě detekovat. Například decryptor viru Badboy[15] se mění po jedné instrukci – a velmi mimořádně. Jedná se o velkou výzvu pro centra zabývající se automatickou analýzou virů.

Jiným příkladem oligomorfních virů je ruská rodina virů nazvaná jako WordSwap.

7.5 Polymorfní viry

Dalším krokem v úrovni obtížnosti je polymorfní útok. Polymorfní viry umějí měnit svůj decryptor do vysokého počtu odlišných instancí, které mohou mít až milióny různých forem.

7.5.1 Virus 1260

Prvním známým polymorfním virem byl virus 1260, který byl naprogramován ve Spojených Státech Markem Washburnem v roce 1990[16]. Tento virus používal mnoho zajímavých technik, jejichž příchod už dříve předpověděl Fred Cohen. Virus používal k dekódování svého těla dva posunující se klíče, a co je ještě důležitější – vkládal nepotřebné instrukce do svého decryptoru. Tyto instrukce jsou v podstatě obyčejným "smetím" a nemají jinou funkci než vytvoření variabilnějšího decryptoru.

Antivirové skenery byly virem 1260 poraženy, protože nebylo možné najít jednoznačný řetězec, podle kterého by bylo možné virus detekovat. Ačkoliv je decryptor viru 1260 velmi jednoduchý, může se zvětšovat a zmenšovat v závislosti na množství vložených nepotřebných instrukcí a náhodného zarovnání na konci decryptoru až po 39 bajtů smetí. Navíc každá skupina instrukcí (prolog, dekódování a inkrementace) vně decryptoru může být obměňována v libovolném pořadí. Kostra decryptoru se tedy může měnit.

Podívejte se na příklad jedné instance decryptoru vyextrahovaného z viru 1260 (viz výpis 7.5).

Výpis 7.5

Příklad decryptoru viru 1260.

```
; Skupina 1 – Instrukce prologu
inc     si              ; volitelné, náhodné smetí
mov     ax,0E9B         ; nastav klíč 1
clc                     ; volitelné, náhodné smetí
mov     di,012A         ; offset "Start"
nop                     ; volitelné, náhodné smetí
mov     cx,0571         ; počet bajtů – klíč 2

; Skupina 2 – Dekódovací instrukce
Decrypt:
xor     [di],cx         ; dekóduj první slovo klíčem 2
sub     bx,dx           ; volitelné, náhodné smetí
xor     bx,cx           ; volitelné, náhodné smetí
sub     bx,ax           ; volitelné, náhodné smetí
sub     bx,cx           ; volitelné, náhodné smetí
nop                     ; nevolitelné smetí
xor     dx,cx           ; volitelné, náhodné smetí
xor     [di],ax         ; dekóduj první slovo klíčem 1

; Skupina 3 – Dekódovací instrukce
inc     di              ; další bajt
nop                     ; nevolitelné smetí
clc                     ; volitelné, náhodné smetí
inc     ax              ; posuň klíč 1
; smyčka
loop    Decrypt         ; skákej, dokud nejsou všechny bajty dekódovány –
                        ; posuň klíč 2
                        ; náhodné zarovnání až do 39 bajtů
Start:
;     zakódované/dekódované tělo viru
```

V každé skupině instrukcí je umístěno až 5 instrukcí smetí (INC SI, CLC, NOP a další, nic nedělající instrukce) bez opakování. Jsou tam vždy dvě instrukce smetí NOP.

1260 nezvládá nahrazování registrů, ale složitější polymorfní útoky tento trik používají. Virus 1260 má nicméně efektivní polymorfní engine, který umí generovat vysoký počet rozličných decryptorů.

7.5.2 Dark Avengerův mutovací engine (MtE)

Dalším důležitým milníkem v historii vývoje polymorfních virů byl MtE[17], mutační engine napsaný Dark Avengerem z Bulharska. První verze MtE byla vypuštěna během léta roku 1991, druhá následovala

počátkem roku 1992. Myšlenka mutovacího enginu byla založena na modulárním vývoji. Pro nováčky bylo obtížné napsat polymorfní virus, a proto jim chtěli schopnější autoři pomoci. Motor MtE byl vypuštěn jako objekt, který se dá připojit k libovolnému viru.

MtE je navržen jako volání funkce mutovacího enginu, kdy se předávají parametry v předdefinovaných registrech. Engine se pak už sám postará o vybudování polymorfní obálky nad virem.

Parametry enginu zahrnují následující:

- Pracovní segment.
- Ukazatel na kód, který se má zakódovat.
- Délku těla viru.
- Báze decryptoru.
- Adresu vstupního bodu hostitele.
- Cílové umístění zakódovaného těla.
- Velikost decryptoru (drobný, malý, střední nebo velký).
- Bitové pole registrů, které se nemají používat.

Výstupem MtE je polymorfní dekódovací rutina se zakódovaným tělem viru umístěným v zadaném bufferu (viz výpis 7.6).

Výpis 7.6

Příklad decryptoru vygenerovaného pomocí MtE.

```
mov     bp,A16C       ; Tento blok inicializuje registr BP
                      ; na "Start"-delta
mov     cl,03         ; (delta je v tomto případě 0x0D2B)
ror     bp,cl
mov     cx,bp
mov     bp,856E
or      bp,740F
mov     si,bp
mov     bp,3B92
add     bp,si
xor     bp,cx
sub     bp,B10C       ; Tak, registr BP je konečně inicializován,
                      ; ovšem dále obsahuje matoucí ukazatel
                      ; na zakódované tělo
Decrypt:
mov     bx,[bp+0D2B]  ; vyber další záznam
                      ; (poprvé na"Start")
add     bx,9D64       ; dekóduj ho
xchg    [bp+0D2B],bx  ; ulož dekódovanou hodnotu na místo
```

```
mov     bx,8F31       ; inkrementuj BP o 2
sub     bx,bp
mov     bp,8F33
sub     bp,bx         ; a oprav délku
jnz     Decrypt       ; jsou už všechny bajty dekódované?
Start:
      ; zakódované/dekódované tělo viru
```

Tento příklad demonstruje, jak je takový engine komplikovaný. Přišli byste na to, jak jej detekovat?

Zdá se být logické, abychom se prvně pokusili shromáždit dostatečně velké množství vzorků. Poprvé mi to trvalo celkem 5 dní, než jsem byl schopen pro něj napsat spolehlivý detektor. MtE dovedlo produkovat některé decryptory, které se objevily jen v 5% (nebo ještě méně) případech. Engine měl nicméně několik malých omezení, které umožnily (s pomocí disassembleru délky instrukcí a stavového stroje) virus spolehlivě detekovat. Ve skutečnosti existuje pouze jeden jediný konstantní bajt v MtE decryptoru, a to 0x75 (JNZ), který je následován záporných offsetem – i tento skok je umístěný na variabilním místě (na konci decryptoru, jehož délka není konstantní).

Poznámka

MtE nepoužívá v decryptoru instrukce smetí, jako to dělá například virus 1260. MtE tedy útočí na techniky, které se snaží optimalizovat decryptory, aby překonaly polymorfismus.

Dopad MtE na antivirový software byl jasný – většina enginů antivirových programů musela projít bolestivým přeprogramováním a nutností implementovat ve skenovacím enginu virtuální stroj. Jak pravil Frans Veldman, "prostě necháme virus, aby odvedl špinavou práci za nás."

Po MtE rychle přišly další podobné enginy, jako třeba TPE (Trident Polymorphic Engine), který vytvořil Masouf Khafir v Holandsku v roce 1993.

V dnešní době jsou známy stovky polymorfních enginů. Většina z nich byla použita k vytvoření pouze několika virů. Jakmile lze detekovat polymorfní decryptor, jeho další používání se stává nevýhodou, protože další nové viry se dají zachytit stejnou detekcí. Taková detekce ovšem sebou nese jistá úskalí ve formě falešných poplachů. Spolehlivější techniky detekce spočívají v rozpoznání samotného těla viru.

To umožňuje autorům virů úspěšně používat stejné polymorfní enginy pro různé viry – až do doby, než budou takové viry odhalitelné heuristickými nebo generickými metodami detekce.

7.5.3 32bitové polymorfní viry

W95/HPS a W95/Marburg[18] byly prvními viry, které začaly používat 32bitové polymorfní enginy. Tyto dva viry byly vytvořeny známým španělským autorem virů, který si říkal GriYo, a to v roce 1998. Ten také vytvořil vysoce polymorfní viry pro DOS, jako třeba virus Implant[19].

Stejně jako polymorfní engine Implantu je i engine viru HPS velmi výkonný a vyspělý. Podporuje subrutiny s použitím instrukcí CALL/RET a podmíněné skoky s nenulovým offsetem. Kód polymorfního enginu zabírá zhruba polovinu kódu viru a mezi vygenerované bloky decryptoru se umisťují náhodné bajty dat. Celý decryptor se vybuduje pouze během fáze inicializace viru, čímž je zajištěn pomalý polymorfismus, což znamená, že antiviroví výzkumníci nemohou efektivně testovat detekční spolehlivost skenerů, protože musí infikované PC vždy restartovat, aby jim virus vytvořil nový decryptor.

Decryptor se skládá z instrukcí procesoru Intel 386. Tělo viru je zakódováno a dekódováno různými metodami, včetně instrukcí XOR/NOT a INC/DEC/SUB/ADD s 8, 16 nebo 32bitovými klíči. Z pohledu detekce toto drasticky snižuje použitelné možnosti. Bohužel musím uznat, že tento polymorfní engine je velmi dobře napsaný, stejně jako zbytek viru a zjevně nebyl vytvořen nějakým začátečníkem.

Třeba následující příklad decryptoru, který jsem pro názornost zjednodušil. Polymorfní decryptor viru je umístěn za variabilně zakódovaným tělem viru. Decryptor je rozdělený na dva menší ostrůvky kódu, které se mohou objevit v promíchaném pořadí. V příkladu ve výpisu 7.7 decryptor začíná na návěští Decryptor_Start a dekódování pokračuje až do doby, než kód konečně skočí na dekódované tělo viru.

Výpis 7.7

Ukázka dekryptoru viru W95/Marburg.

```
Start:
                          ; zde je umístěné zakódované/dekódované tělo viru
Routine-6:
dec     esi               ; dekrementuj čítač smyčky
ret
Routine-3:
mov     esi,439FE661h     ; nastav čítač smyčky v ESI
ret
Routine-4:
xor     byte ptr [edi],6F ; dekóduj konstantním bajtem
ret
Routine-5:
add     edi,0001h         ; posuň ukazatel dekódovaní na další bajt
ret
Decryptor_Start:
call    Routine-1         ; nastav EDI na "Start"
call    Routine-3         ; nastav čítač smyčky
Decrypt:
call    Routine-4         ; dekóduj
call    Routine-5         ; přejdi na další bajt
call    Routine-6         ; dekrementuj čítač smyčky
cmp     esi,439FD271h     ; je už všechno dekódované?
jnz     Decrypt           ; ještě ne, pokračuj v dekódování
jmp     Start             ; skoč na dekódovaný začátek viru
```

```
Routine-1:
call    Routine-2           ; trik CALL-POP!
Routine-2:
pop     edi
sub     edi,143Ah           ; EDI ukazuje na"Start"
ret
```

Předchozí decryptor je vysoce strukturovaný a každý jeho funkční celek je umístěn ve speciální rutině. Výsledkem jsou miliony možných vzorků kódu s náhodnými daty mezi ostrůvky.

Polymorfní viry mohou vytvořit bezpočet nových decryptorů, které používají různé metody pro zakódování konstantních částí virového těla (kromě svých datových oblastí).

Některé polymorfní viry, jako třeba W32/Coke používají vícenásobné vrstvy zakódování. Varianta Coke dokonce umí polymorfně infikovat dokumenty Microsoft Wordu. Coke mění své makro (prostřednictvím svého binárního kódu) přímo, místo používání samotných maker. Normálně jsou polymorfní makro viry velmi pomalé, protože vyžadují mnoho interpretací (interpretation). Protože Coke generuje změněná makra pomocí svého binárního kódu, nepotýká se s žádným zpomalováním počítače a jeho přítomnost je tedy hůře rozpoznatelná. Podívejte se na následující příklad viru Coke, které jsem vybral ze dvou instancí změněného makra AutoClose() ve výpisu 7.8.

Výpis 7.8

Krátký příklad polymorfního makra viru Coke.

```
'BsbK
Sub AuTOclOSE()
oN ERROr REsuMe NeXT
SHOWviSuAlBASIcEditOr = faLsE
If nmnGG > WYff Then
For XgfqLwDTT = 70 To 5
JhGPTT = 64
KjfLL = 34
If qqSsKWW < vMmm Then
For QpMM = 56 To 7
If qtWQHU = PCYKWvQQ Then
If lXYnNrr > mxTwjWW Then
End If
If FFnfrjj > GHgpE Then
End If
```

Druhý příklad je trochu delší (kvůli zanesenému smetí). V příkladu jsem vyznačil některé podstatné instrukce – povšimněte si, že ani ty nejsou zobrazeny ve stejném pořadí. Předchozí instance viru například vypíná Visual Basic Editor, takže už jej v menu Wordu nespatříte. Ve výpisu 7.9 se toto odehraje taky, ale až za hromadou vloženého smetí a jiných instrukcí.

Výpis 7.9

Další příklad polymorfního makra viru Coke.

```
‘fYJm
Sub AUtOcLOse()
oN ERRor REsUME NexT
optIOns.saVenorMALPrOmpT = FAlsE
DdXLwjjVlQxU$ = "TmDKK"
NrCyxbahfPtt$ = "fnMM"
If MKbyqtt > mHba Then
If JluVV > mkpSS Then
jMJFFXkTfgMM$ = "DmJcc"
For VPQjTT = 42 To 4
If PGNwygui = bMVrr Then
dJTkQi = 07
‘wcHpsxllwuCC
End If
Next VPQjTT
quYY = 83
End If
DsSS = 82
bFVpp = 60
End If
tCQFv=1
Rem kJPpjNNGQCVpjj
LyBDXXXGnWW$ = "wPyTdle"
If cnkCvCww > FupJLQSS Then
VbBCCcxKWxww$ = "Ybrr"
End If
opTiONS.COnFIrmCOnvErsiOnS = faLSe
Svye = 55
PgHKfiVXuff$ = "rHKVMdd"
ShOwVisUALbaSiCEdITOR = fALSe
```

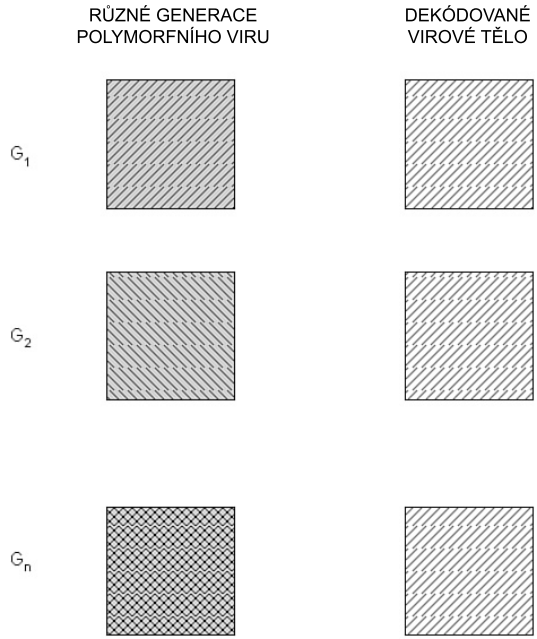
Novější polymorfní enginy používají decryptor založený na RDA, který implementuje útok hrubou výpočetní silou proti svému konstantnímu, avšak variabilně zakódovanému tělu. Ruční analýza takových virů může vést k velkým překvapením. Často v nich lze nalézt neefektivitu v náhodnosti generovaných algoritmů, která se dá zneužít k protiútoku. Někdy může dokonce jeden jediný "wildcard" řetězec zajistit perfektní detekci.

Většina skenerů měla už před léty emulátor kódu, který uměl emulovat 32bitové PE soubory. Jiní antiviroví výzkumníci pouze implementovali dynamické dekódování, aby si s takovými viry poradili. To

fungovalo v předchozích případech, protože tělo viru bylo po dekódování konstantní. Jak vyplývá z různých AV testů, někteří dodavatelé stále nemají podporu detekce složitějších virů.

Autoři virů použili kombinaci technik zatajení vstupního bodu s 32bitovým polymorfismem, aby učinili práci antivirových skenerů ještě obtížnější. Navíc se pokusili implementovat techniky obrany proti emulaci, aby vyřadili emulátory kódu z provozu.

Všechny polymorfní viry si nicméně stále sebou nesou konstantní kód svého těla, které je možné pomocí mnoha vylepšených technik dekódovat a identifikovat. Pro názornost se podívejte na obrázek 7.3.



Obrázek 7.3 *Instance zakódovaných a dekódovaných těl polymorfního viru.*

7.6 Metamorfní viry

Autoři virů stále tráví týdny a měsíce vytvářením nových polymorfních virů, které díky svým chybám v kódu nemají šanci se dále rozšířit. A na druhou stranu – antiviroví výzkumníci jsou schopni si s detekcí takových virů poradit během několika minut nebo dnů, za což může nízký existující počet efektivních polymorfních enginů.

Samozřejmě, že se autoři virů snaží implementovat různé nové techniky vývoje kódu, aby práci výzkumníkům co nejvíce ztížili. W32/Apparition byl prvním známým 32bitovým virem, který ke svému vývoji v dalších generacích nepoužíval polymorfní decryptory. Virus si v sobě nese zdrojový kód a vypustí jej kdykoliv, když najde na počítači nějaký nainstalovaný kompilátor. Virus vkládá a odebírá ze svého zdrojového kódu smetí a znovu se rekompiluje. Tímto způsobem se virus v dalších generacích naprosto liší od předešlých. Je jen náhoda, že se W32/Apparition nestal velkým problémem. A taková technika by