МАТЕРИАЛЫ САЙТА

Поиск адресов API

Журнал «Хакер» , 22.07.2004 9 мин на чтение 0 0 856

Часто, когда мы пишем какую-нибудь программу и вызываем что-то типа FindFile, мы даже не задумываемся как компилятор определяет какую именно процедуру ему надо запустить. Это удобно и вам, и

разработчикам языка, и авторам этих API функций. Конечно все прекрасно, но только для «добропорядочных» программистов, а те кто хотят написать свой Win32 вирус получают гемор на долгое время (если кто-нить не поможет).

Немного истории вперемешку с теорией. Были такие золотые времена, когда не было еще окошек, и властвовал DOS. Золотыми они были для вирусмейкеров, которые могли творить в ОСи что угодно и не особо напрягаться. Как тебе наверное известно, в ДОСе все работало на прерываниях. Они были чем-то вроде предков современных API, но, в отличи от последних, точки входа к прерываниям лежали по фиксированным адресам. Их легко можно было вызвать, заменить своей программой и вообще творить много веселых вещей. Но когда вышла Windows 95 все VX-кодеры попадали со стульев, в которые, за это время, успели основательно врасти их задницы. И, как казалось в то время, не зря. Мелкомягкие объявили, что благодаря их окну вирусы навсегда побеждены и на земле будет царить любовь и добро. Действительно, они обрубили все что раньше использовали вирмейкеры. Одним из этих лишений оказалась скрытость адресов API функций. Многие ушли со сцены в то время, но остались лучшие и стали изучать win32.

Как я уже говорил, когда вы пишете свою программу компилятор сам заботиться о том, чтобы ваше творение вызывала правильные API. Их адреса «вшиваются» навсегда в таблицу импорта. Но вирус сам должен встраиваться в чужой код и тогда он не будет знать какие адреса ему нужны. Конечно, можно сделать эти адреса фиксированными, но тогда он будет работать только под определенным окном, а это не есть good. Нам надо каждый раз определять их. Существует много способов сделать это. Я расскажу об одном из них.

Как вы знаете, когда мы запускаем приложением, код вызывается откуда-то из KERNEL32 (т.е. KERNEL делает вызов нашего кода), а

потом,
когда вызов сделан, адрес возврата лежит в стеке (адрес памяти в ESP). Получив этот адрес нам надо будет найти место с которого загрузился KERNEL32, то есть там где есть MZ. Так как это dll PE формата, то мы обратимся к таблице экспорта и получим нужные нам функций по их именам. Теперь подробнее.

```
.code
start:
mov esi,[esp]
```

Вот таким кодом в самом начале проги мы получим значение адреса где-то внутри CreateProcess. Теперь мы получим начало страницы из которой был вызван код и подготовимся к поиску сигнатуры ZM. Будем искать 50 страниц. Этого достаточно и мы не залезем хрен знает куда.

```
and esi,0FFFF0000h
mov ecx,5
```

Сравним слово по адресу esi с ZM. Если оно совпадет, то проверим PE ли это файл. Если же нет, то отнимем 10 страниц от значения в esi и продолжим поиск. Если не нашли вернем ноль.

```
find:
cmp word ptr [esi],”ZM”
jz check_pe
find_next:
sub esi,10000h
failed:
mov esi,0h
got_k32:
xchg eax,esi
```

Проверка на PE формат тоже очень проста. Мы просто сравниваем

слово по смещению 3Ch от MZ с сигнатурой PE. Если все совпадает то мы получили адрес загрузки KERNEL32.

```
check_pe:
mov edi,[esi+3Ch]
add edi,esi
cmp word ptr [edi],"EP"
jz got_k32
jmp find_next
```

Теперь в eax так называемая база. Дальше я приведу код двух процедур которые будут искать все требуемые функции. Первая из них называется GetAPIs. Она задает параметры второй процедуре GetAPI, которая ищет каждую API отдельно, основываясь на данных переданных первой процедурой. Они хорошо откомментированы и не должны вызвать особых вопросов если вы знаете формат PE заголовка. Также описаны их интерфейсы, т.е. как с ними взаимодействует вышестоящая процедура. Несколько оговорок: код адаптирован под вирусы, поэтому кода вы видите что-то типа [ebp+xxh] не смущайтесь, в ebp дельта смещение, в обычных программах его можно убрать (оно равно нулю); в переменной kernel содержится база полученная выше в eax.

```
;————————;
;Эта процедура получает адрес требуемой API функции по ее имени ;
;ВХОДНЫЕ ДАННЫЕ: ESI — указатель на имя функции с учетом регистра ;
;ВЫХОДНЫЕ ДАННЫЕ: EAX — адрес требуемой функции ;
;ECX — длина имени функции ;
;————————;
GetAPI proc
mov edx,esi ;сохраняем указатель имя

mov edi,esi ;для проверки длины
xor al,al ;будем сравнивать посимвольно с 0
```

```
@_1: scasb
jnz @_1

sub edi,esi ;EDI = размер имени функции
mov ecx,edi ;в ECX тоже самое
xor eax,eax
mov esi,3Ch ;смещение на начало PE заголовка
add esi,[ebp+kernel]

lodsw ;значение по адресу ESI в EAX
add eax,[ebp+kernel] ;нормализуем смещение PE
mov esi,[eax+78h] ;RVA таблицы экспорта
add esi,1Ch ;плюс смещение на RVA таблицы адресов

add esi,[ebp+kernel] ;нормализуем, и получаем ссылку на RVA т.адр.
lea edi,[ebp+ATVA] ;готовимся к пересылке

lodsd ;RVA табл. адресов в EAX
add eax,[ebp+kernel] ;нормализуем
stosd ;сохраняем в переменной ATVA

lodsd ;RVA табл. имен в EAX
add eax,[ebp+kernel] ;нормализуем
push eax ;сохраняем в стеке
stosd ;и в переменной NTVA

lodsd ;RVA табл. ординалов в EAX
add eax,[ebp+kernel] ;нормализуем
stosd ;сохраняем в OTVA

pop esi ;в ESI адрес табл. имен

xor ebx,ebx ;mov ebx,0
@_3: lodsd ;[ESI]==>EAX, RVA на имя функции
```

```
push esi ;сохраняем указатель на RVA имени функции
add eax,[ebp+kernel] ;нормализуем
;готовимся к сравнению
mov esi,eax ;в ESI адрес имени функции
mov edi,edx ;в EDI адрес образца имени
push ecx ;сохраняем длину образца имени
cld
rep cmpsb ;сравниваем побайтово
pop ecx ;восстанавливаем длину образца имени
jz @_4 ;переходим сюда если совпали имена
pop esi ;нет,
восстанавливаем указатель на RVA им.ф-и (уже следующей)
inc ebx ;увеличиваем счетчик
jmp @_3 ;и опять начинаем
с начала
@_4:
pop esi ;очищаем стек
xchg eax,ebx ;в EAX значение счетчика
shl eax,1 ;умножаем на
2 (тк ординалы это wordы)
add eax,dword ptr [ebp+OTVA];прибавляем к началу
таблицы орд. счетчик
xor esi,esi
xchg eax,esi ;в ESI адрес ординала
lodsw ;в EAX получаем сам ординал
shl eax,2 ;умножаем его на 4 (тк dword) и получаем смещение
относительно табл. адресов
add eax,dword ptr [ebp+ATVA];нормализуем
mov esi,eax ;в ESI адрес на RVA API функции
lodsd ;получаем это RVA в EAX
add eax,[ebp+kernel] ;нормализуем
ret ;и на выходе адрес требуемой ф-и

GetAPI endp
```

```
;————————;
Public GetAPIs
;————————;
;Эта процедура получает все желаемые API функции ;
;ВХОДНЫЕ ДАННЫЕ: ESI — указатель на имя первой функции в
формате
ASCIIz;
;EDI — указатель на переменную которая будет содержать ;
;адрес API функции ;
;EAX — адрес базы kernel32 ;
;ВЫХОДНЫЕ ДАННЫЕ: НИЧЕГО ;
;Сама функция предполагает использование следующих структур ;
; ;
; ESI указывает на —. db «FindFirstFileA»,0 ;
; db «FindNextFileA»,0 ;
; db «CloseHandle»,0 ;
; […] ;
; db 0BBh ; Отмечает конец массива ;
; ;
; EDI указывает на —. dd 00000000h ; Будущий адрес FFFA ;
; dd 00000000h ; Будущий адрес FNFA ;
; dd 00000000h ; Будущий адрес CH ;
; […] ;
;————————;
GetAPIs proc
jmp next
kernel dd 0
ATVA dd 0
NTVA dd 0
OTVA dd 0
next:
mov dword ptr [ebp+kernel],eax ;сохраняем адрес базы
__1: ;начинаем поиск функций
push esi ;сохраняем нужные регистры
```

```
push edi
call GetAPI ;в EAX получаем адрес функции
pop edi ;восстанавливаем регистры
pop esi
stosd ;сохраняем по адресу на который указывает EDI

add esi,ecx ;переходим к следующему имени
cmp byte ptr [esi],0BBh ;проверяем, не конец ли массива имен
jz exit ;если да то выход
jmp __1 ;нет, ищем следующую функцию
exit:
ret
GetAPIs endp
```

Приведенный выше код всего лишь один из методов получения адресов API функци. Существуют еще множество более или менее надежных методов,
например с использованием SEH, или поиска нужных API в таблице импорта самой жертвы вируса. В интернете много информации на это тему, так что смотрите сами.

Журнал «Хакер»

Теги: Софт Статьи

ДАЛЕЕ ПО ЭТОЙ ТЕМЕ РАНЕЕ ПО ЭТОЙ ТЕМЕ

Полиморфизм: новые техники №1

Снова о сокрытии процессов: теперь и XP

Полиморфизм: новые техники №2

01.09.2004

24.09.2004

02.09.2004

Прячем Pinch в один прием

08.10.2004

Перехват вызовов API-функций на Delphi №2

02.08.2004

Новые веяния в бэкдорах

28.02.2005