Misc

Misc development

Вызов функции без импорта

Сегодня посмотрел очередную свежую серию MLP:FiM и понял, что мне нечем себя занять. В связи с этим решил включить трек моей любимой группы (как вы, наверное, догадались - это Ранетки :D) и написать что-нибудь эдакое на ассемблере. По совету друзей, которые маются всякой фигней, вместо того, чтобы заняться чем-нибудь полезным и написать нормальную статью в блог, выбор пал на написание нескольких макросов, которые позволяют вызывать библиотечные функции без использования таблицы импорта.
Минусы: макросы базозависимые.
Плюсы: макросы потокобезопасные.
Anyway: на базе этих макросов можно достаточно быстро построить новые, удобные вам.

Пример использования получившихся макросов:

```
1  .486
2  .model flat, stdcall
3  option casemap :none
4
5  include \masm32\include\windows.inc
6  include \masm32\macros\macros.asm
7  include \masm32\macros\no_import.asm
8
9
10 .code
11 start PROC
12     LOCAL buf[128]:BYTE
13     LOCAL sz:DWORD
14     mov sz, 127
15     ;подготовка возможности вызовов без таблицы импорта
16     noimport_call_prepare
17
18     noimport_invoke_load chr$("GetUserNameA"), chr$("advapi32.dll"), addr buf, addr s
19
20     ;вызов MessageBoxA с предварительной загрузкой библиотеки user32.dll
21     noimport_invoke_load chr$("MessageBoxA"), chr$("user32.dll"), 0, addr buf, chr$("
22
23     ;kernel32.dll всегда в памяти, ее не надо предварительно грузить
24     noimport_invoke chr$("ExitProcess"), chr$("kernel32.dll"), 0
25 ret
26 start ENDP
27 end start
```

Сами макросы:

```
UNICODE_STRING STRUCT
    xLength dw ?
    MaximumLength dw ?
    Buffer dd ?
UNICODE_STRING ENDS

PEB_LDR_DATA STRUCT 4
    xLength dd ?
    Initialized db ?
    SsHandle dd ?
    InLoadOrderModuleList LIST_ENTRY <>
    InMemoryOrderModuleList LIST_ENTRY <>
    InInitializationOrderModuleList LIST_ENTRY <>
PEB_LDR_DATA ENDS

PEB STRUCT
    InheritedAddressSpace db ?
    ReadImageFileExecOptions db ?
    BeingDebugged db ?
    Spare db ?
    Mutant dd ?
    ImageBaseAddress dd ?
    LoaderData dd ?
    ProcessParameters dd ?
    SubSystemData dd ?
    ProcessHeap dd ?
    FastPebLock dd ?
    FastPebLockRoutine dd ?
    FastPebUnlockRoutine dd ?
    EnvironmentUpdateCount dd ?
    KernelCallbackTable dd ?
    EventLogSection dd ?
    EventLog dd ?
    FreeList dd ?
    TlsExpansionCounter dd ?
    TlsBitmap dd ?
    TlsBitmapBits dd 2 dup(?)
    ReadOnlySharedMemoryBase dd ?
    ReadOnlySharedMemoryHeap dd ?
    ReadOnlyStaticServerData dd ?
    AnsiCodePageData dd ?
    OemCodePageData dd ?
    UnicodeCaseTableData dd ?
    NumberOfProcessors dd ?
    NtGlobalFlag dd ?
    Spare2 db 4 dup(?)
    CriticalSectionTimeout dq ?
    HeapSegmentCommit dd ?
    HeapDeCommitTotalFreeThreshold dd ?
    HeapDeCommitFreeBlockThreshold dd ?
    NumberOfHeaps dd ?
    MaximumNumberOfHeaps dd ?
    ProcessHeaps dd ?
    GdiSharedHandleTable dd ?
    ProcessStarterHelper dd ?
    GdiDCAttributeList dd ?
    LoaderLock dd ?
    OSMajorVersion dd ?
    OSMinorVersion dd ?
    OSBuildNumber dd ?
    OSPlatformId dd ?
    ImageSubSystem dd ?
    ImageSubSystemMajorVersion dd ?
    ImageSubSystemMinorVersion dd ?
```

```
65      GdiHandleBuffer dd 22h dup(?)
66      PostProcessInitRoutine dd ?
67      TlsExpansionBitmap dd ?
68      TlsExpansionBitmapBits db 80h dup(?)
69      SessionId dd ?
70  PEB ENDS
71
72  LDR_MODULE STRUCT
73      InLoadOrderModuleList LIST_ENTRY <>
74      InMemoryOrderModuleList LIST_ENTRY <>
75      InInitializationOrderModuleList LIST_ENTRY <>
76      BaseAddress dd ?
77      EntryPoint dd ?
78      SizeOfImage dd ?
79      FullDllName UNICODE_STRING <>
80      BaseDllName UNICODE_STRING <>
81      Flags dd ?
82      LoadCount dw ?
83      TlsIndex dw ?
84      HashTableEntry LIST_ENTRY <>
85      TimeDateStamp dd ?
86  LDR_MODULE ENDS
87
88
89  xinvoke MACRO name:REQ, params:VARARG
90      count = 0
91      FOR xparam, <params>
92        count = count + 1
93        @CatStr(var,%count) TEXTEQU @CatStr(&xparam)
94      ENDM
95
96      REPEAT count
97        IF @SizeStr(%@CatStr(var,%count)) GT 4
98          IFIDNI @SubStr(%@CatStr(var,%count), 1, 4),<addr>
99            __temp_text TEXTEQU @SubStr(%@CatStr(var,%count), 5)
100           lea eax, __temp_text
101           push eax
102         ELSE
103           push @CatStr(var,%count)
104         ENDIF
105       ELSE
106         push @CatStr(var,%count)
107       ENDIF
108
109       count = count - 1
110     ENDM
111
112     mov eax, name
113     call eax
114 ENDM
115
116
117 noimport_invoke_load MACRO funcname:REQ, libname:REQ, params:VARARG
118     count = 0
119     _addr_found = 0
120     _eax_found = 0
121     FOR xparam, <params>
122       IFIDNI <&xparam>,<eax>
123         _eax_found = 1
124       ENDIF
125
126       count = count + 1
127       @CatStr(var,%count) TEXTEQU @CatStr(&xparam)
128     ENDM
129
130     REPEAT count
131       IF @SizeStr(%@CatStr(var,%count)) GT 4
132         IFIDNI @SubStr(%@CatStr(var,%count), 1, 4),<addr>
133           __temp_text TEXTEQU @SubStr(%@CatStr(var,%count), 5)
```

```
134           lea eax, __temp_text
135           push eax
136           _addr_found = 1
137         ELSE
138           push @CatStr(var,%count)
139         ENDIF
140       ELSE
141         push @CatStr(var,%count)
142       ENDIF
143
144       count = count - 1
145     ENDM
146
147
148     IF _addr_found EQ 1
149       IF _eax_found EQ 1
150         %ECHO [WARNING: EAX is used with ADDR at Line @CatStr(%@Line)]
151       ENDIF
152     ENDIF
153
154     xinvoke _LoadLibraryAFunc, libname
155     xinvoke _GetProcAddressFunc, eax, funcname
156
157     call eax
158 ENDM
159
160 noimport_invoke MACRO funcname:REQ, libname:REQ, params:VARARG
161     count = 0
162     _addr_found = 0
163     _eax_found = 0
164     FOR xparam, <params>
165       IFIDNI <&xparam>,<eax>
166         _eax_found = 1
167       ENDIF
168
169       count = count + 1
170       @CatStr(var,%count) TEXTEQU @CatStr(&xparam)
171     ENDM
172
173     REPEAT count
174       IF @SizeStr(%@CatStr(var,%count)) GT 4
175         IFIDNI @SubStr(%@CatStr(var,%count), 1, 4),<addr>
176           __temp_text TEXTEQU @SubStr(%@CatStr(var,%count), 5)
177           lea eax, __temp_text
178           push eax
179           _addr_found = 1
180         ELSE
181           push @CatStr(var,%count)
182         ENDIF
183       ELSE
184         push @CatStr(var,%count)
185       ENDIF
186
187       count = count - 1
188     ENDM
189
190
191     IF _addr_found EQ 1
192       IF _eax_found EQ 1
193         %ECHO [WARNING: EAX is used with ADDR at Line @CatStr(%@Line)]
194       ENDIF
195     ENDIF
196
197
198     xinvoke _GetModuleHandleA, libname
199     xinvoke _GetProcAddressFunc, eax, funcname
200
201     call eax
202 ENDM
```

```
203
204
205 noimport_call_prepare MACRO
206     .data?
207         _GetProcAddressFunc dd ?
208         _LoadLibraryAFunc dd ?
209         _GetModuleHandleA dd ?
210         _krnl dd ?
211     .code
212         assume fs:nothing
213
214         mov eax, fs:[30h]
215         mov eax, PEB.LoaderData[eax]
216         mov eax, PEB_LDR_DATA.InLoadOrderModuleList[eax]
217         mov eax, LDR_MODULE.InLoadOrderModuleList[eax]
218         mov eax, LDR_MODULE.InLoadOrderModuleList[eax]
219         mov eax, LDR_MODULE.BaseAddress[eax]
220         mov edx, eax
221         mov _krnl, eax
222         add eax, IMAGE_DOS_HEADER.e_lfanew
223         mov eax, [eax]
224         add eax, edx
225         add eax, IMAGE_NT_HEADERS.OptionalHeader.DataDirectory.VirtualAddress
226         mov eax, [eax]
227         add eax, edx
228         mov edi, eax
229         add eax, IMAGE_EXPORT_DIRECTORY.AddressOfNames
230         mov eax, [eax]
231         add eax, edx
232         xor esi, esi
233
234         __find:
235             push eax
236             mov eax, [eax]
237             add eax, edx
238             mov ebx, chr$("GetProcAddress")
239
240             __compare:
241                 mov cl, [eax]
242                 mov ch, [ebx]
243                 test cl, cl
244                 jz __test_ch
245                 test ch, ch
246                 jz __next
247                 jmp __check
248
249                 __test_ch:
250                     test ch, ch
251                     jz __found
252                     jmp __next
253                 __check:
254                     cmp cl,ch
255                     jne __next
256
257                     inc eax
258                     inc ebx
259                     jmp __compare
260
261         __next:
262             inc esi
263             pop eax
264             add eax,4
265             jmp __find
266         __found:
267
268         pop eax
269
270         mov eax, edi
271         add eax, IMAGE_EXPORT_DIRECTORY.AddressOfNameOrdinals
```

```
272         mov eax, [eax]
273         add eax, edx
274         add eax, esi
275         add eax, esi
276         mov bx, [eax]
277         movzx ebx, bx
278         mov eax, edi
279         add eax, IMAGE_EXPORT_DIRECTORY.AddressOfFunctions
280         mov eax, [eax]
281         add eax, edx
282         add ebx, ebx
283         add eax, ebx
284         add eax, ebx
285         mov eax, [eax]
286         add eax, edx
287         mov _GetProcAddressFunc, eax
288
289         xinvoke _GetProcAddressFunc, _krnl, chr$("LoadLibraryA")
290         mov _LoadLibraryAFunc, eax
291
292         xinvoke _GetProcAddressFunc, _krnl, chr$("GetModuleHandleA")
293         mov _GetModuleHandleA, eax
294
295         assume fs:error
296 ENDM
```

Надеюсь, кому-нибудь пригодится.

Исходные коды одним архивом: скачать.

Обновлено: 14.11.11

 dx / Ноябрь 5, 2011 / Assembler, Windows, Сниппеты / masm32, импорт, макросы

Вызов функции без импорта: 21 комментарий

 **flisk**

Ноябрь 6, 2011 в 16:38

Спасибо, весьма интересно! Единственное, что значит базозависимые?

"Минусы: макросы базозависимые."

 **dx**

Ноябрь 6, 2011 в 18:47

Значит, что они используют абсолютные смещения (секция данных используется). Вот если бы все макросы только стек и относительные смещения использовали, тогда базозависимыми они бы не были.

User

Ноябрь 13, 2011 в 14:17

А если один из параметров макроса "xinvoke" будет "addr localvar" адресом локальной переменной, то такой макрос должен не сработать. Как быть?

dx

Ноябрь 13, 2011 в 22:09

Явно использовать инструкцию lea.

User

Ноябрь 13, 2011 в 22:18

Это само собой, вопрос, непонятно как отличить локальную от глобальной, ведь нужно будет во время генераии push сделать lea reg, push reg.

dx

Ноябрь 13, 2011 в 22:20

Что значит "как отличить", если ты сам код пишешь и видишь, где локальные, где глобальные :)
Я предлагаю в макросы не толкать addr, а перед вызовом макроса делать lea, а в макрос уже регистр передавать.
Хотя если немного поколупать макросы, возможно, получится и универсально сделать.

User

Ноябрь 14, 2011 в 00:43

Понял ход твоих мыслей. Это обходной путь, так неинтересно. :) А я думаю как можно сделать полную эмуляцию invoke, чтобы и с адресами локальных переменных не было проблем. Неисключена и такая ситуация, что количество параметров которые должны содержать адреса локальных переменных будет больше, чем рабочих регистров. Можно конечно использовать указатели, но это обходной путь.

 **dx**

Ноябрь 14, 2011 в 00:47

Кстати, если в invoke ты используешь и регистр eax, и addr что_то, он ругнется. А чтобы эти макросы заставить работать как invoke, надо над ними поработать еще)

 **dx**

Ноябрь 14, 2011 в 01:42

Только что сделал полную эмуляцию invoke, даже предупреждение выдает, если попытаться использовать в одном вызове оператор addr и регистр eax. Завтра выложу.

 **dx**

Ноябрь 14, 2011 в 19:40

Обновление добавлено.

 **User**

Ноябрь 15, 2011 в 00:01

Спасибо за обновление, полезный макрос. Предлагаю использовать предупреждение только в том случае, если регистр eax используется левее, чем addr, т.к. только в таком случае регистр eax будет реинициализироваться. Такую проверку можно сделать сохранением индекса параметров, а перед варнингом сделать проверку IF

index_eax LT index_addr.
Правильно ли я понимаю, различия между адресом глобальной и локальной переменной нету? Будут идеи как отличить адрес глобальной от локальной переменной?

dx

Ноябрь 15, 2011 в 14:44

Адрес никак не отличишь. Разве что, существуют какие-то виндовые функции, позволяющие отличить адрес стека от адреса кучи. Локальные переменные всегда в стеке лежат, глобальные - в куче. Может быть, такое и вручную реализуемо, не задумывался. По идее, где-то должны лежать указатели на начальные адреса стека и кучи процесса...

Уведомление: Обходим все популярные антивирусы, или лажовая защита в действии

Викинг

Июнь 26, 2012 в 23:44

DX, ты бы не мог сделать не базозависимую версию (для крипторов/пакеров/протекторов)? Я пишу криптор.

Kaimi

Июнь 26, 2012 в 23:54

Расскажи-ка, что там базозависимого сейчас?

Викинг

Июнь 27, 2012 в 13:13

DX:
"Значит, что они используют абсолютные смещения (секция данных

используется). Вот если бы все макросы только стек и относительные смещения использовали, тогда базозависимыми они бы не были."

dx

Июнь 27, 2012 в 14:20

Ну так возьми и напиши базонезависимые макросы, what's the problem? Человек, пишущий криптор сам, должен в таком разбираться.

glukftp

Июль 23, 2012 в 16:33

Здесь можно взять более удобную реализацию такой идеи!
http://www.wasm.ru/srclist.php?list=9#358

аркад

Август 24, 2014 в 04:04

А можнобрать функции из адрессного пространства библиотеки без подгрузки самой библиотеки

Kaimi

Август 24, 2014 в 11:11

ГРЕШНОВАТО

dx

Август 25, 2014 в 10:34

Верно толкуешь, Каимий! Какие ж адресные пространства, когда модуль не подгружен?

Misc / Сайт работает на WordPress