2008

[Anti-Ollydbg Techniques & Solutions]

Unregistered !

www.reaonline.net

10/08/2008

..::[REA TEAM]::..
Reverse Engineering Associati...

I. Introduction

Anti Debug – một thuật ngữ có thể đã khá quen với rất nhiều người trong số chúng ta, tuy nhiên tôi chắc chắn rằng không phải bất cứ cracker nào cũng biết về nó một cách rõ ràng, nhất là đối với những ai mới bắt đầu bước vào thế giới Reverse. Qua quá trình tự tìm hiểu cũng như tham khảo các tài liệu có sẵn, tôi đã tổng hợp lại và viết lên tài liệu này nhằm mục đích giới thiệu về những phương pháp Anti Debug hay gặp và đưa ra những phương pháp giải quyết thích hợp (có thể bạn không hoàn toàn hiểu khái niệm về Anti Debug thì xin hãy kiên nhẫn đọc tiếp).
**Lưu ý : Những gì tôi làm dưới đây không gì ngoài mục đích học tập và nghiên cứu do đó tôi sẽ không chịu bất kì trách nhiệm nào nếu các bạn sử dụng nó vào những mục đích không tốt* !! 0k13! L3t's R4p w1th m3… (hehe, câu này là cọp pi của anh Kiên, anh ấy thích nhạc Rock còn em khoái Rap nên thay Rock = Rap ☺)

II. Theoretically

Trước khi đi vào vấn đề chính chúng ta cần nắm một số lý thuyết:

1. Anti Debug là gì ?

Anti Debug có thể hiểu nôm na là những phương pháp làm cho một chương trình tránh khỏi sự debug của các cracker/unpacker.

2. Tại sao tài liệu này chỉ nói tới Anti-Debug đối với OllyDbg ?

Hiện nay Ollydbg là Debugger được các cracker ưa chuộng nhất, chính vì thế tài liệu này chỉ nói tới những phương pháp Anti-Debug đối với OllyDbg.

3. Những dạng "thù hình" của Anti – Ollydbg ?

Dưa vào từng cách thức, mục đích của Anti – Olly mà ta có thể phân chia nó ra thành từng mục nhỏ như sau :

- Anti – Debugging
- Anti – Disassembling
- Anti – Break Point
- Anti – Tracing
- Anti – Analyzing
- Anti – Attaching
- Anti – Dumping
- Debug Breaking

Ở phần tiếp theo chúng ta sẽ đi vào phân tích từng phương pháp Anti – Olly

III. Researching to understand carefully

Trước khi bắt tay vào công việc tìm hiểu, các bạn hãy tự tìm cho mình một bản Olly "nguyên thủy" tức là chưa được chỉnh sửa (tốt nhất nên download từ ollydbg.de) và nên gỡ bỏ hết các pluggin chỉ chừa lại 2 cái đó là AnalyzeThis! và CommandBar.

1. Anti – Debugging

Phần này chúng ta sẽ tìm hiểu những cách mà một phần mềm có thể phát hiện ra debugger của chúng ta và những cách qua mặt những kiểu anti này mà hạn chế tác động đến file gốc nhất có thể.
Sau đây là những cách mà một phần mềm có thể kiểm tra xem nó có đang bị debug bởi một debugger hay không :

a. PEB Checks

PEB (Process Environment Block) là một cấu trúc nằm trong cấu trúc TEB (Thread Environment Block), nó cung cấp cho ta thông tin về những tiến trình đang thực thi trên máy, vì dụ như thông tin về Memory, Size, Executable, Directory, … Cấu trúc này thường thấy trong kĩ thuật coding virus đặc biệt là trong hidding process.
Vậy ta có thể dựa vào cái của PEB để phát hiện debugger ? Một software có thể dựa vào các cờ (Flag) được Set trong PEB để kiểm tra xem liệu nó có đang bị debug hay không ? Cụ thể là các Flag sau:

i. BeingDebugged Flag

Cờ này nằm ở offset thứ 2 của PEB, nếu một ứng dụng đang bị debug thì cờ này sẽ được Set giá trị 1 và sẽ là giá trị 0 nếu ngược lại.
Thực chất API IsDebuggerPresent trong module kernel32.dll cũng sử dụng Flag này để Return value.

```
mov eax,dword ptr fs:[30h] ;PEB
cmp byte ptr ds:[eax+2h], 0 ;check BeingDebugged
jne being_debugged
```

* Solution: Ta hoàn toàn dễ dàng bypass Flag này luôn set giá trị là 0 bằng cách đi đến địa chỉ của nó rồi úm ba la … 1 → 0 hoặc có thể sử dụng các plugin có sẵn như HideOD, AdvanceOlly.

ii. NtGlobalFlag & Heap Flag

NtGlobalFlag nằm ở offset 0x68 của PEB, tương tự như BeingDebugged nó cũng được Set là 0 nếu như không bị Debug và sẽ là 0x70 trong trường hợp ngược lại.

```
mov eax, dword ptr fs:[30h] ;PEB
cmp byte ptr ds:[eax+68h], 70h ;check NtGlobalFlag
jne being_debugged
```

Nếu như NtGlobalFlag được Set thì đồng thời Heap Flag cũng được Set. Vậy Heap Flag là cái wái gi? Đó là một cờ trong trường ProcessHeap của PEB Structure. Nếu một ứng dụng không bị debug thì cờ này sẽ mang giá trị 0x2 ngược lại sẽ là đang bị Debug.

```
mov eax, dword ptr  fs:[30h] ;PEB
mov eax, dword ptr ds:[eax+18h] ;get process heap base
mov eax, dword ptr ds:[eax+0ch] ;Flags
sub eax, 2
jne being_debugged
```

* Solution: Dễ dàng set cho các Flag này luôn mang giá trị như không bị debug.

b. Timming Checks

Chương trình dùng các lệnh lấy thời gian rồi tính khoảng cách giữa những lần lấy, nếu chúng quá lớn có nghĩa rằng nó đang bị debug. Có 2 lệnh thường được dùng trong Detect Method này đó là dùng API **GetTickCount** hoặc dùng **RDTSC** Instruction trong ASM.

Mothod dùng GetTickCount khá đơn giản để bypass bằng cách Hook thẳng vào API này, còn đối với RDTSC thì sao?

RDTDC (Read Time Date Stamp Counter) đúng như ý nghĩa của nó, lệnh này đọc giá trị của chu kỳ đồng hồ và trả về giá trị tại EAX, bình thường khi chạy một ứng dụng ngoài Debugger thì nó trả về giá trị nhỏ hơn 0x12345678 còn khi nó bị debug thì giá trị trả về sẽ là rất lớn, Coder có thể lợi dụng điều này để check xem chương trình có đang bị debug không bằng cách lấy 2 giá trị RDTSC rồi trừ cho nhau, đối với Olly giá trị này luôn lớn hơn 0x0FFFFFF. Các bạn có thể xem một ví dụ nhỏ về cách dùng lệnh này:

```
RDTSC
PUSH EAX
RDTSC
SUB EAX,DWORD PTR SS:[ESP]
CMP EAX,0FFFFFF
JG being_debugged
```

* Solution : Có thể dùng plugin AdvandeOlly để Fake RTDSC và hook API GetTickCount

c. Parents Process Checks

Method này kiểm tra xem tiến trình cha của tiến trình của chính nó có phải là explorer.exe không, nếu không thì rất có thể nó được gọi từ một Debugger hay một Command – Line Program nào đó. Quá trình Check này được thực hiện nhu sau:

- Lấy Process ID (PID) của chính nó
- Lấy PID của Process cha
- Lấy PID của Explorer.exe
- So sánh 2 giá trị vừa lấy được ở trên, nếu không trùng khớp thì chương trình đang bị Debug

* Solution : Hook các API liên quan đến việc lấy PID của process cha sao cho nó luôn trả về PID của explorer.exe

d. API Checks

Chương trình dùng các API đặc biệt liên quan đến việc Debug ứng dụng rồi tùy theo từng giá trị trả về để xem nó có đang bị Debug hay không. Các API thường gặp là:

- IsDebuggerPresent
- OutputDebugStringA
- CheckRemoteDebuggerPresent
- ZwQuerySystemInformation
- ZwQueryInformationProcess
- ZwQueryObject

…

* Solution : Tùy theo từng kiểu Check mà ta Hook các API tương ứng.

e. Find Debugger

i. Find Window & Find Process

Chương trình có thể tìm process của Debugger hay tìm Debugger Window mà cụ thể ở đây là tìm Process "OLLYDBG.EXE" và dùng API FindWindowA với parameter sCation là "OLLYDBG".

* Solution : Phương pháp này khá đơn giản nên ta có thể change Olly's Name và Olly's Caption.

ii. Initial ESI Register Value

Với Window XP, khi chương trình được load vào Olly, thanh ghi ESI luôn (hoặc hầu hết) mang giá trị -1 (0xFFFFFFFF), ta có thể coi đây như là một Method Detect Debugger (trường hợp này có thể là ngẫu nhiên, không phải lúc nào cũng đúng)

f. TLS Callback.

Kỹ thuật cho phép chạy một đoạn code trước khi tới Entry Point của target, tại Callback Function tác giả có thể thêm những Method Anti Debug khác nữa nhằm phát hiện và tiêu diệt Debugger

* Solution : Với Olly ta có thể Check Option System Break Point để dừng lại trước khi TLS Callback Function được thực thi từ đó bypass chương trình chính.

2. Anti – Disassembling

Để làm cho quá trình Debug và Disassemble khó khăn hơn, chúng ta có thể thay thế các code đơn giản bằng những cái phức tạp hơn, sau đây là một vài ví dụ:

- Đối với lệnh MOV, chắc chắn các bạn đã rất quen thuộc với câu lệnh này rồi. Thay vì dùng lệnh MOV đơn giản như MOV EAX, EBX ta có thể dùng các câu lệnh khác kết hợp lại:

```
PUSH EBX
POP EAX
```

Hay:

```
XCHG EAX,EBX
PUSH EAX
POP EBX
...
```

- Đối với lệnh JMP, thay vì dùng câu lệnh JMP thông thường bạn có thể thay thế nó bằng lệnh PUSH + RET:

VD: JMP 401000
Ta có thể dùng cặp lệnh:

```
PUSH 401000
RET
```

Ngoài việc thay thế các câu lệnh đơn giản bằng các câu lệnh phức tạp hơn ta còn có thể chèn vào chương trình chính những đoạn mã giả, những đoạn mã không hề có một chút chức năng gì hay chén vào những lệnh nhảy chồng chéo… Người ta gọi đây là dạng Obsfuscate.
Để có thể debug được những dạng anti – debug này đòi hỏi reverser phải có tính kiên nhẫn và phải có nhiều kinh nghiệm phân tích code từ đó đưa ra một phương pháp đơn giản hóa Code, người ta gọi việc làm này là DeObsfuscate.

2. Anti – BreakPoint

Như đã biết, Ollydbg cung cấp cho ta 3 dạng Breakpoint để làm điểm ngắt của chương trình trong khi Debug, chúng bao gồm :

- Software Break Point
- Hardware Break Point
- Memory Break Point

Chúng ta sẽ đi tìm hiểu từng phương thức mà Olly làm cho chương trình dừng lại tại điểm mà ta đạt Break Point.

- Đầu tiên là Software Break Point.

Để chương trình dừng lại tại vị trí user đặt software break point (SBP) thì Olly đã thay thế 1 opcode tại đó bằng byte 0xCC, việc làm này tương tự như chương trình gọi API DebugBreak, thay vì dừng lại tại API, Olly đã điều khiển cho chương trình bị debug dừng lại tại nơi đặt SBP. Chính vì đặc điểm này mà một SBP có thể vị phát hiện bằng các so sánh từng byte của chương trình với 0xCC. Ngoài ra có thể phát hiện SBP thông qua CRC Check.

- Hardware Break Point (HBP)

Một HBP sẽ dừng chương trình bằng một bộ mô phỏng điện tử ngoài hoàn toàn không phụ thuộc vào phần cứng. Các mạch điện logic (thông thường được tạo ra trong RAM) sẽ quan sát mỗi chu kỳ Bus và dừng thực thi khi EIP đi tới địa chỉ mà chúng ta đã đã đặt breakpoint trước đó.
Theo lý thuyết thì HBP hoàn toàn không ảnh hưởng gì đến chương trình, tuy nhiên HBP vẫn có thể bị phát hiện. Khi gặp exception do ta đặt HBP, hệ thống gửi đến Exception Handle một thông điệp là cấu trúc CONTEXT chứa thông tin giá trị của các Register của hệ thống, của các Control và của cả Debugger, khi Olly nhận được thông điệp này của hệ thống Olly sẽ phân tích nó và xác định điểm dừng của HBP. Ta có thể phát hiện HBP nhờ chu trình này và qua viện cài đặt SEH.

```
MOV EAX,DWORD PTR SS:[ESP+C] ;ContextRecord
XOR ECX,ECX
XOR ECX,DWORD PTR DS:[EAX+4] ;Dr0
XOR ECX,DWORD PTR DS:[EAX+8] ;Dr1
XOR ECX,DWORD PTR DS:[EAX+C] ;Dr2
XOR ECX,DWORD PTR DS:[EAX+10] ;Dr3
```

Nếu có HBP thì Dr0, Dr1, Dr2, Dr3 sẽ mang thông tin về HBP, từ đó ta có thể Detect được Debugger.

- Memory Break Point (MBP)

Olly đã lợi dụng Guard Page để tạo Exception rồi lại lợi dụng Exception để tạm ngưng chương trình với MBP. Olly sẽ Set Guard Page hay còn gọi là Access Mode tới một vùng trong Memory tùy theo kiểu MBP của user, sau đó Olly sẽ tạo SEH để lọc Exception, khi gặp đúng Exception đã dự tính sẵn từ trước nó sẽ Set lại Access Mode rồi Stop ở chỗ đã đặt MBP.
Vậy một MBP có thể bị phát hiện hay không? Câu trả lời là hoàn toàn dễ bị phát hiện khi coder đã nắm vững và ghi nhớ Page Guard của từng Section trong Memory, và họ có thể kiểm soát được Access Mode tới những Section này để hạn chế MBP.

3. Anti – Tracing

Phần này sẽ giới thiệu về các phương pháp hạn chế Reverser/Cracker thực hiện Tracing hay còn gọi là Single Step trong quá trình Debug.
Để gây khó khăn trong việc RE, coder có thể thêm vào chương trình chính những đoạn code gây rối, vì vậy Anti – Disassembling (Obfuscation) cũng là một hình thức Anti – Tracing. Ngoài ra còn có thể áp dụng một phương pháp nữa đó chính là dựa vào bẫy lỗi. SEH (Structured Exception Handler) được sử dụng như một công cụ kiểm soát lỗi của chương trình, khi một chươn trình gặp Exception thông thường nó sẽ Crash, nhưng khi bắt đầu chạy, coder cài đặt một SEH tức là tạo một Structured Exception Handle để khi chương trình gặp Exception sẽ quay lại thực thi code tại SEH Chain

Vậy tại sao ta lại có thể dùng SEH vào việc Anti – Tracing?
Chương trình sẽ cài đặt một SEH rồi sau đó sẽ tự tạo một Exception để buộc chương trình phải dừng tại Exception sau đó execute tiếp tục tại SEH Chain đã Set.
Chính vì vậy nếu như ta Trace từng câu lệnh trong Olly thì gặp Exception là không thể tránh khỏi, khi gặp Excetion ta buộc phải nhấn Shift + F7/F8/F9 để bypass Excetion và tiếp tục thực thi chươn trình, vì thế quá trình Trace sẽ bị gián đoạn.

* Solution : Để tránh điều này, Olly cung cấp cho ta một chức năng đó là xem trước các SEH Chain, khi gặp Exception các bạn hãy xem address của SEH tiếp theo ở đâu bằng cách vào Menu View → SEH Chain → Đặt BP tại đó → Shift + F9.

4. Anti – Analyzing

Những phương pháp phổ biến nhất đó là Encryption và Compression, khi gặp phải những dạng này Reverser không thể phân tích code ngay khi load vào Debugger mà còn phải để cho chương trình tự Decrypt và Decompress code.

Encryption: Mã hóa code, trong những Packer hiện nay thì quá trình Decrypt trong file packed khác rất nhiều so với quá trình Encrypt chính vì thế Reverse không thể từ thuật toán Decrypt suy ra thuật Encrypt và ngược lại được, tuy nhiên encrypt lại có một nhược điểm: nếu không kết hợp khéo léo giữa Obfucation và Encrypt thì việc tìm ra được thời điểm code đã được Decrypt hoàn toàn đơn giản và thường là kết thúc của mỗi vòng lặp.

Compression: Nén code mã hóa và Data gốc của chương trình, giống như Encrytion, Compression cũng dễ dàng bị Decompress bởi vòng lặp quá lộ liễu nếu không kết hợp Anti – Disassembling.

Ngoài ra còn có một phương pháp anti – analyzing khá thú vị đó là Garbage Code.

Đó là thêm các code vô nghĩa vào chương trình, nghe thì có vẻ giống Obfucsation nhưng thực ra lại khác hoàn toàn, Obfuscation là làm rối quá trình thực thi của chương trình gốc tức là Obfuscated Code vẫn được thựa thi nhưng

còn Garbage Code thì khác, những code này không được thựa thi. Ví dụ sau là áp dụng Garbage Code với MASM:
Một App nhỏ bằng ASM với Section Code như sau:

```
MOV EAX, 1000h
ADD EAX, 200h
MOV ECX, 100h
MUL EAX
```

Khi được load vào Olly code hiển thị rất rõ ràng:

```
00401000    $  B8 00100000   MOV EAX,1000
00401005    .  05 00020000   ADD EAX,200
0040100A    .  B9 00010000   MOV ECX,100
0040100F    .  F7E0          MUL EAX
```

Còn đây là code sau khi đã thêm Garbage:

```
      XOR EBX, EBX
      OR EBX, EBX
      JNE Junk1
      JE NoJunk1
Junk1:
      Db 0Fh
NoJunk1:
      MOV EAX, 1000h
      OR EBX, EBX
      JNE Junk2
      JE NoJunk2
Junk2:
      Db 0Ch
NoJunk2:
      ADD EAX, 200h
      OR EBX, EBX
      JNE Junk3
      JE NoJunk3
Junk3:
      Db 0Ah
NoJunk3:
      MOV ECX, 100h
      OR EBX, EBX
      JNE Junk4
      JE NoJunk4
Junk4:
      Db 0Dh
NoJunk4:
      MUL EAX
```

Giờ hãy load thử APP này vào Olly:

```
00401000    $  33DB          XOR EBX,EBX
00401002    .  0BDB          OR EBX,EBX
00401004    .  75 02         JNZ SHORT 1.00401008
00401006    .  74 01         JE SHORT 1.00401009
00401008    >  0FB8          ???
0040100A    ?  0010          ADD BYTE PTR DS:[EAX],DL
```

```
0040100C    ?  0000          ADD BYTE PTR DS:[EAX],AL
0040100E    .  0BDB          OR EBX,EBX
00401010    .  75 02         JNZ SHORT 1.00401014
00401012    .  74 01         JE SHORT 1.00401015
00401014    .  0C 05         OR AL,5
00401016    .  0002          ADD BYTE PTR DS:[EDX],AL
00401018    .  0000          ADD BYTE PTR DS:[EAX],AL
0040101A    .  0BDB          OR EBX,EBX
0040101C    .  75 02         JNZ SHORT 1.00401020
0040101E    .  74 01         JE SHORT 1.00401021
00401020    .  0AB9 00010000 OR BH,BYTE PTR DS:[ECX+100]
00401026    .  0BDB          OR EBX,EBX
00401028    .  75 02         JNZ SHORT 1.0040102C
0040102A    .  74 01         JE SHORT 1.0040102D
0040102C    .  0D F7E00000   OR EAX,0E0F7
```

He he, chỉ với vài Garbage Code mà ta đã làm cho chương trình trở lên phức tạp rất nhiều mặc dù đã có sự hỗ trợ của plugin AnalyzeThis!.

5. Anti – Attaching

Olly cung cấp cho ta một chức năng khá tốt đó là Attach một tiến trình đang chạy, với tính năng này ta có thể thực hiện Unpack, Crack một ứng dụng mà đôi khi việc load thẳng vào lại gặp rất nhiều khó khăn. Chính vì thế các Packer thuộc dạng hàng khủng trên thị trường cũng đã trang bị cho sản phẩm của họ Option Anti – Attach mà tiêu biểu là TTProtect với hình thức Anti-Debug khá tốt. Chúng ta sẽ đi vào phân tích cách thức anti – attach là như thế nào.
Đầu tiên ta sẽ phải nghiên cứu cách mà Olly sử dụng để Attach một tiến trình đang chạy. Olly sử dụng API DebugActiveProcess để bắt đầu thực hiện Debug một Process đang chạy, bản thân API này sẽ gọi tới nhiều API khác nữa:
DebugActiveProcess → DbgUiConnectToDbg → CrsGetProcessId → NtOpenProcess → DbgUiDebugActiveProcess → …

Mà bản thân các API được gọi sẽ gọi nhiểu API khác nữa, chính vì thế để anti – attach ta có thể Hook một trong số các API được gọi, các API hay bị Hook để Anti – Attach mà ta thường gặp là:

NtContinue
DbgUiRemoteBreakin
DbgUiConnectToDbg
…

* Solution : Để có thể Attach một process an toàn vào Olly ta có thể khôi phục lại các API đã bị Hook bằng cách ReWrite lại code của API đó từ Process khác bằng API WriteProcessMemory.

6. Anti – Dumping

a. Thay đổi SizeOfImage

Cách đơn giản nhất của anti – dumping là thay đổi giá trị SizeOfImage trong PEB (Process Environment Block), việc làm này can thiệp tới sự truy nhập của Dumper thông thường tới Process để Dump Memory ra Disk đồng thời cũng tác động đến việc Attach Process đó vào Olly, nó cũng hạn chế được LordPE, một PE Editor và Dumper được xếp vào loại Good. Tuy nhiên LordPE lại có quá nhiều độc chiêu, chỉ vài Option là có thể tự động Correct ImageSize của Process :(

Code ASM thay đổi SizeOfImage như sau:

```
mov eax, dword ptr fs:[30h] ;PEB
mov eax, dword ptr ds:[eax+0ch] ;LdrData
mov eax, dword ptr ds: [eax+0ch] ;get InLoadOrderModuleList
add dword ptr ds:[eax+20h], 1000h ;adjust SizeOfImage
```

b. Xóa PE Header

Sau khi được load lên, chương trình có thể tự phá hủy PE Header của nó trong Memory để hạn chế Dumpping, tuy nhiên LordPE lại tiếp tục có nhiều tuyệt chiêu khác : Fix PE Header :(

c. Nanomites

Cái này hoàn toàn quen thuộc với những ai hay mần Armadillo ☺, Nanotmite có thể hiểu nôm na như sau: Chương trình thay thế một số code của chương trình chính bằng cách lệnh INT3 tương ứng với DebugBreak. Chương trình áp dụng cơ chế Anti – Dumping bằng Nanomite đòi hỏi nó phải tự debug chính nó, giống như Debug Blocker của Arm vậy, Process cha có nhiệm vụ làm Debugger của Process con, process con ở đây đã mang Nanomite, khi gặp Exception INT3, process cha với tư cách là một Debugger sẽ tìm trong bảng có sẵn xem đây có phải là Nanomite của chương trình không hay là của protected app, nếu là Nanomite nó sẽ tính toán rồi dẫn hướng chương trình chính đi tới phần code ban đầu của nó.
Chính vì điều này, khi ta dump memory ra disk thì thu được một Process con mang đầy INT3 Exception mà không khôi phục được code gốc

d. Code Splicing

Đây cũng là một trick khá good của Arm, đó chính là nó sẽ write code chính vào một section nào đó nằm ngoài file trong memory sau đó thay thế những code chính này bằng các lệnh nhảy tới section ảo này. Khi dump full ta nhận được file dump với các lệnh nhảy tới các Section không tồn tại. Stolen Byte cũng là một dạng nhỏ trong Code Splicing.

e. Virtual Machine

Đây là một Method xếp vào hạng pro nhất trong các dạng Anti – Dumping, bởi vì không có thời điểm nào mà Code gốc lại được thực thi một cách trực tiếp, giống như cái tên của nó, chương trình thực thi code hệt như một máy ảo, tất cả được copy vào Memory, được giải mã trong Memory, được thực thi trong Memory…
Chính vì thế để dump một Process từ Memory ta gặp phải rất nhiều khó khăn, không chỉ bởi nó không liền kề thành một khối mà ngoài code gốc còn lẫn rất nhiều Junk code.

7. Debug Breaking

Phần này nói về một số cách thức bẻ gãy quá trình debug khi ứng dụng của bạn bị Debug. Cách thức tốt nhất là áp dụng nhiều phương pháp kết hợp lại với nhau. Đặc biết là không thể thiếu được Anti – Disassembling và Anti – Analyzing. Bởi vì mặc dù bạn có dùng nhiều hình thức Detect Debugger đến đâu thì nếu như loạt Code đó không được phức tạp hóa mà cứ hiện chình ình ra trước mắt thì tất cả công sức coi như xuống sông xuống bể hết.

Sau đây là một số kiểu Anti – Ollydbg nữa mà tôi muốn giới thiệu. Tại sao tôi không xếp mục này chung với các phần trên? Bởi vì đây chỉ là những mẹo nhỏ làm cho việc RE trở nên khó khăn hơn, thực chất đây là những trick mà coder có thể add trực tiếp vào chương trình mà không cần nhờ tới sự trợ giúp nào của các Packer/Protector nào khác.

- Đầu tiên là việc sử dụng API BlockInput, API này sẽ tạm thời khóa các thiết bị ngoại vi như Mouse và Key Board, sau khi hoàn thành cách quá trình Detect khác thì ta lại gọi API BlockInput để Unlock các thiết bị. Tuy nhiên API này khá lộ liễu nên dễ dàng bị qua mặt.
- Với Ollydbg, các plugin Hide Debugger hiện nay rất phổ biến, dựa vào nguyên lý hoạt động của Olly cùng các Plugin của nó mà ta có thể phát hiện ra Olly. Ví dụ như kết hợp với việc tìm kiếm Olly Window bằng API FindWindow ta có thể tìm kiếm các Process xem có cái nào mang tên "Ollydbg.exe" hay không? Không tránh khỏi trường hợp nhầm lẫn nhưng ta cứ kill hết, giết nhầm hơn bỏ sót mà. Ngoài ra ta thấy Olly thường Load một DLL có tên là DBGHELP.DLL, ta có thể tìm kiếm các module của process sau đó kiếm xem có cái nào trùng với "DBGHELP.DLL" không rồi từ đó kill.
- AdvanceOlly là một plugin khá tốt, nó có thể bypass hầu hết các cơ chế anti debug. Tuy nhiên sau quá trình sử dụng và xem xét tôi đã tự rút ra cho minhd một phương pháp anti – debug khá hay đó là: Sử dụng kết hợp hàm RDTSC và API CreateFileA. Nguyên lí của phương pháp này như sau:
  Nếu ta check option Anti – RDTSC của AdvanceOlly, plugin này sẽ tự động tạo một file có tên là "fakerdtsc.sys" để fake giá trị

trả về của lệnh RDTSC, ta có thể dùng CreateFileA để check xem file này có tồn tại hay không, nếu có tức là app của bạn đang bị Debug. Nếu không thì ta có thể dùng RDTSC để Detect Olly.

Xin nhắc lại, tất cả phương pháp Anti Debug ở trên bạn nên kết hợp với Anti Analyzing và Anti Disassembling để đạt được hiệu quả cao nhất !

IV. Conclusion

Vậy là những gì tôi muốn trình bày trong bài viết này cũng đã hoàn thành, vì đây là bài viết dựa trên ý kiến chủ quan của tôi nên nếu có gì sai sót mong các bạn góp ý để tôi kịp thời sửa chữa.

Tài liệu chỉ mang tính học tập nghiên cứu, tôi không chịu trách nhiệm khi các bạn sử dụng kiến thức ở trên vào việc của các bạn.

Những tài liệu đã tham khảo:

- ***Anti Unpacker Trick by Peter Ferrie, Senior Anti-Virus Researcher, Microsoft Corporation***
- ***The Art Of Unpacking by Mark Vincent Yason, Malcode Analyst, X-Force Research & Development, IBM Internet Security Systems***
- ***Crackproof Your Software (2002) by Pavol Cerven, http://www.anticracking.sk***
- ***Reversing with Lena151 – TUT 37 - Indept Unpacking & Anti-Anti-Debugging A Combination Packer/Protector***

Xin cảm ơn bạn vì đã bỏ thời gian quý báu đọc bài viết này !

Great Thank: ***Computer_Angel, kienmanowar, Zombie, Moonbaby, Hacnho, Benina, Merc, RCA, Trickyboy, WhyNotBar, hacnho, LittleBoy, moth, NhatPhuongLe, Xianua, P.E Onimusha, ... all member of REA ...lena151 ... and you!***

Unregistered !

Bắc Ninh, 8 tháng 10 năm 2008

Mọi thắc mắc cần giải đáp vui lòng liên hệ email:
Unregistered.08@gmail.com