**AntiDebug**

Graphical user interface

Description automatically generated

+ Bài này yêu cầu ta phải hiểu các technique antidebug được sử dụng và bypass để có được flag

+ Khi run ta sẽ nhập key sau đó chương trình sẽ gen ra flag để so sánh với key ta đã nhập

+ Sẽ có thanh biểu thị so sánh key mà ta đã nhập, và ô thể hiện số phần trăm chính xác

Graphical user interface, application

Description automatically generated

+ Ta sẽ có flow chương trình về các technique antidebug như sau:

\* SetUnhandledExceptionFilter(lpTopLevelExceptionFilter);

+ Ở đây ta thấy sau khi SetUnhandledExceptionFilter được gọi ta sẽ gặp exception *INT\_DIVIDE\_BY\_ZERO* tại dòng *“div eax”* (eax = 0)

+ Nếu chương trình không bị debug và bắt được một exception sẽ tiến hành gọi *TopLevelExceptionFilter* (#filter function) cho phép ta tạo một dialog box sử dụng hàm *DialogBoxParamA*

+ Để bypass ta sẽ bỏ qua phần bắt exception bằng cách patch cho chương trình nhảy thẳng vào TopLevelExceptionFilter sau khi gọi SetUnhandledExceptionFilter

Trước và sau khi patch, ta đã bỏ qua được phần *“div eax”*

Graphical user interface, application

Description automatically generated

\* DialogBoxParamA: hàm này sẽ tạo modal dialog box từ resource ta truyền vào, ta sẽ phân tích 0x4010F7: *DialogFunc* được truyền vào

Text, application

Description automatically generated

Khi khởi tạo một dialog box mặc định lParam đầu sẽ là message WM\_INITDIALOG => tiến hành phân tích tại địa chỉ 0x40111D

Diagram, text

Description automatically generated

\* Tại 0x40111D (tạm gọi INITDIALOG) Graphical user interface, text, application, email

Description automatically generated

Text, whiteboard

Description automatically generatedTa thấy có hàm 0x401757 là một hàm nhận 2 param (size, lpFunc) và tiến hành kiểm tra tính toàn vẹn

ESI: point to function address

ECX: size check

Load word from ESI to EAX then add to EBX

Compare BX with value at 0x40320F to check integrity

If not equal => ExitProcess(0);

=>Ta có thể bypass bằng cách patch đoạn so sánh với BX từ *JE* thành *JMP* để bỏ qua bước so sánh

A picture containing timeline

Description automatically generated

Trở lại với INITDIALOG ta thấy phía dưới có tiến hành check các flag để antidebug A picture containing text

Description automatically generated

+ [PEB+3] (IsDebugged): dùng trong hàm *IsDebuggerPresent* hàm sẽ trả về 1 nếu chương trình đang bị debug

+ [PEB+0x68] (NtGlobalFlag): giá trị này sẽ bằng 0x70 nếu đang debug, 0 nếu không debug

+ [PEB+18] (Heap struct):

- [\_HEAP + 0xC] (HeapFlags): nếu không phải HEAP\_GROWABLE (0x00000002) => đang debug

- [\_HEAP + 0x10] (ForceFlags): nếu không bằng 0 => đang debug

=> Do mọi giá trị đều khác nên sẽ patch từ *JNE -> JE* để bypass

Text

Description automatically generated with medium confidence

Tiếp ta có 1 vòng loop để modify 10 bytes bắt đầu từ 0x40109A tạm thời chưa biết để làm gì ta sẽ note vị trí này lại

Diagram, text

Description automatically generated

=> KẾT THÚC INITDIALOG

\* Bây giờ ta có thể tiếp tục run -> dialog sẽ hiện ra lúc này khi ta tương tác sẽ sent các message command tiếp hành phân tích 0x4011AC (Tạm gọi: COMMAND)

Text

Description automatically generatedtại đây có switch case kiểm tra wParam truyền vào DialogFunc

+ 0x3EC giá trị wParam khi nút “Clear” được click

+ 0x3EE giá trị wParam khi nút “Check” được click

Đặt break point tại địa chỉ mà check button sẽ nhảy tới

0x4011C4. Nhập key ngẫu nhiên và click “Check”

Text

Description automatically generated with medium confidenceSau khi run over 2 hàm SendDlgItemMessage \*NtQueryInformationProcess

+ Tại đây ta có tham số truyền vào là Debugport(7) và CurrentProcess(-1) để check debugger, nếu debug => giá trị tại 0x403714=1 ngược lại bằng 0

=> patch giá trị current process => 0

Lúc này hàm sẽ trả về INVALID\_HANDLE

=> tiếp tục patch *jne 0x401256 (ExitProcess(0))* => *je 0x401256* để bỏ qua ExitProcess()

+ Ta sẽ lại có thêm một vòng loop tương tự ở trên, ta sẽ note thêm vị trí này

Text

Description automatically generated with medium confidence

Tiếp đến tại dòng comment “1st” ta sẽ jump call liên tục 3 lần nhưng disassembler ở đây không hiện thị

Kết quả cuối cùng của ta tại dòng comment “OutputDebugStringA” sẽ gọi hàm đó

A picture containing text

Description automatically generated

\*OutputDebugString

Thường thì OutputDebugString sẽ dùng để compare return code với LastError để check xem có debug hay không nhưng trong bài này thì phần này không ảnh hưởng nên có thể bỏ qua

Text

Description automatically generated

Text

Description automatically generatedTiếp khi gọi hàm 0x401525 ta thấy có sử dụng hàm NtSetInformationProcess

Text

Description automatically generatedỞ đây ta cần patch giá trị 0xFFFFFFFE(current process) => 0 để có thể bypass do tham số truyền THREAD\_INFORMATION\_CLASS=0x11 (ThreadHideFromDebugger)

Lúc này sẽ có 2 vòng lặp để gen 19 kí tự flag ra cho chúng ta *“NtQu3ry1nf0rm@t10nPr0(355”*

Text

Description automatically generatedSau khi gen flag hàm 0x401789 nhận 2 tham số là size và địa chỉ sẽ tiến hành check xem có bất kì byte “int 3” (breakpoint) nào hay không, kết quả return lưu tại 0x40320A

+ Ta sẽ patch value từ bên ngoài nếu thấy cần thiết

A picture containing application

Description automatically generated

Sau 2 lần GetTickCount (Lần thứ nhất lưu giá trị vào 0x403718) lần thứ 2 dùng vào hàm 0x401819

Ta sẽ có rất nhiều switch case gây ra exception khi continue tiếp ta sẽ nhảy về hàm check informationthread: 0x401525

Text

Description automatically generated Chạy đến khi return ta được flag kế tiếp “R@!s33xc3pt!on”

Khi return chương trình sẽ nhảy thẳng đến địa chỉ 0x40135A Text

Description automatically generated

Tại đây ta sẽ patch lệnh JNE -> JE để bypass phần compare value mà CheckRemoteDebuggerPresent trả về.

+ Tiếp có hàm 0x401391 nó sẽ gọi tiếp nữa đến 1 hàm find window dùng CreateToolhelp32Snapshot để list process và strstr để tìm process theo tên

+ Sau khi tìm được sẽ thử mở process theo pid => yes: admin privilege -> ExitProcess

Text

Description automatically generated

=> bypass bằng cách patch JE 0x4013B7 -> JMP 0x4013B7

Tiếp đến sẽ có 1 hàm tương tự để check debugger

Hàm này sẽ duyệt và tìm tên của Debugger => Nếu thấy => Terminate

A picture containing text

Description automatically generated

Ta sẽ cần patch từ JE 0x401756 -> JMP 0x401756 để đảm bảo TerminateProcess sẽ không được gọi

Text

Description automatically generated

Sau khi thoát ra, ta sẽ thấy chương trình tiếp tục gọi 0x40107B để check thời gian bằng GetTickCount đã lấy ở bên trên

Text

Description automatically generated

Khoảng cách tối thiểu giữa 2 lần GetTickCount như trong chương trình là 0x12C ta sẽ patch JBE -> JMP để bypass

\*Cuối cùng sau khi đã patch hết tất cả đặt breakpoint cuối tại 0x40143A và run over hàm GenFlag tại địa chỉ 0x4010C9

A picture containing text

Description automatically generated

Flag: “NtQu3ry1nf0rm@t10nPr0(355R@!s33xc3pt!onD3bugPr1v1l3g3St@ckT1m3CCS3lf-P3BF1ndW1nd0wH1d1ng@nt1-R3v3rs3”

**Tổng Kết**

Các technique đã được sử dụng:

+ NtQueryInformationProcess: Debug port value = 7

+ NtSetInformationProcess: Thread Class: 0x11

+ OutputDebugString

+ SetUnhandledExceptionFilter -> TopLevelExceptionFilter function

+ Debug privilege: try to open csrss.exe

+ Check breakpoint in memory (int 3 - “CD 03”)

+ PEB flag: IsDebugged, Heap Flags, Heap Force Flag, NtGlobalFlag

+ GetTickCount

Graphical user interface

Description automatically generated with medium confidence