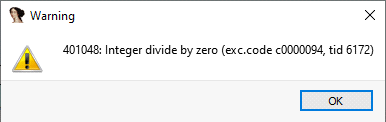
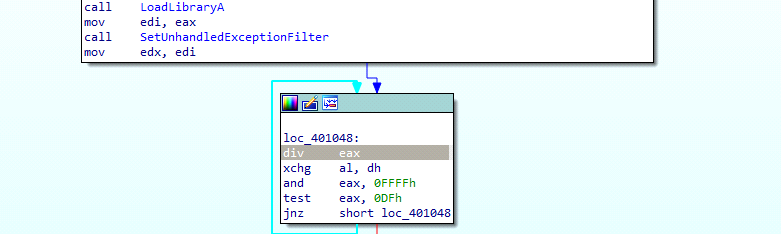
* **SetUnhanledExceptionFilter()**

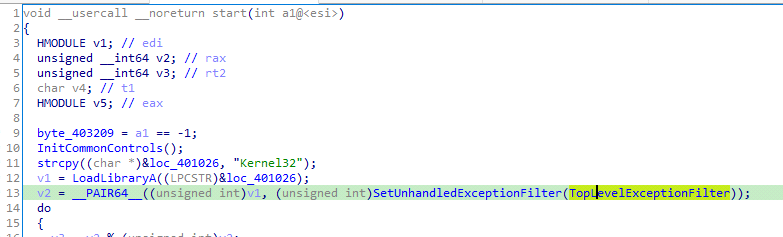
Load file vào Ida 32bit, thực hiện debug thì gặp exception Integer divide by zero.



Exception này gây ra tại “div eax”



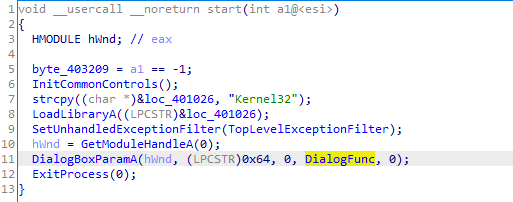
Đây là ý đồ của chương trình vì ở phía trên chương trình có gọi API SetUnhandledExceotionFilter để đăng kí handle exception với handler là TopLevelExceptionFilter.

 và flow của chương trình sẽ được tiếp tục khi exception này được handle (khi chương trình handle exception thì hàm này sẽ được call, và nếu có debuger thì exception này được handle bởi debuger chứ không phải chương trình, do vậy hàm kia không được call và chương trình không tiếp tục).

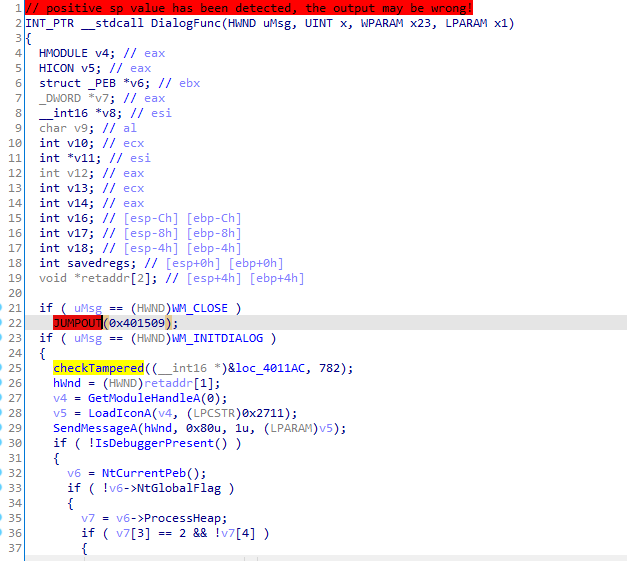
Để bypass trick này, ta patch program jump thẳng tới TopLevelExceptionFilter.

2. Check integrity

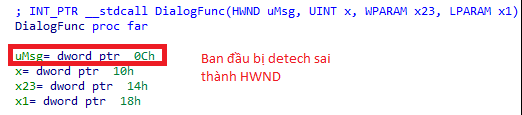
F5 để decompiler update lại chương trình.



Sau đó chương trình tạo 1 dialog box với procedure là DialogFunc.

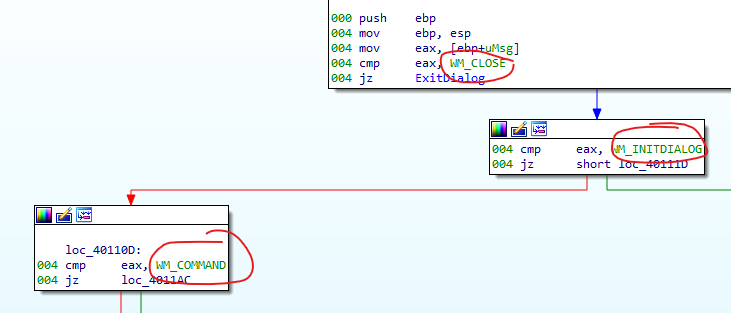


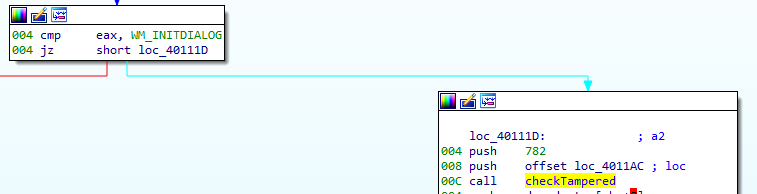
Mặc dù IDA detect sai các tham số của DialogFunc:



tuy nhiên dựa vào calling convention, ta có thể biết được offset +0xC là MessageId của winapi chứ không phải là HWND, tạm thời sửa tên cho dễ nhìn.

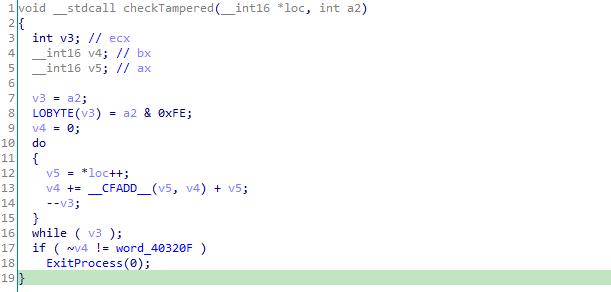
Sau khi sửa xong thì đã dễ nhìn hơn rất nhiều, có thể thấy DialogFunc handle 3 window message là WM\_CLOSE, WM\_INITDIALOG và WM\_COMMAND





Khi handle WM\_INITDIALOG, chương trình gọi hàm checkTampered để load 782bytes từ địa chỉ 4011AC để kiểm tra xem có bị breakpoint hay patch không (nếu đặt breakpoint thì debugger sẽ patch 1 byte thành int3(CC) để raise exception tại đó). Nếu phát hiện vùng nhớ bị thay đổi thì sẽ thoát.





Bypass bằng cách patch line 17, jump thẳng tới return.

Graphical user interface, application

Description automatically generated

3. NtQueryInformationProcess và IsDebuggerPresent

Trong DialogFunc cũng có sử dụng 1 số API để check debugger, patch jump condition để không nhảy vào luồng kết thúc nữa.

4. PEB, và vân vân

Tất cả đều bypass bằng patch jump condition

Graphical user interface, application

Description automatically generated

Sau khi bypass hết, debug lại thì sẽ xuất hiện dialog

Vẫn trong DialogFunc, trrong nhánh xử lý WM\_COMMAND, chương trình có thực hiện 1 switch-case wParam low word (Identifier của các control) để xử lý từng button. Debug thêm ta biết được hành vi của từng control:

* Button clear set text trong edit box thành xâu tại đại chỉ &unk\_403000 (empty string), cập nhật thanh tiến độ progress bar về trống và digit box về 0.
* Button check, nếu có data trong edit box, get text trong input box, check debug một lần nữa bằng cách lấy status của process cộng với từng byte tại loc\_4010A4 (NtQueryInfomation trả về struct PROCESSBASICINFORMATION có byte đầu tiên luôn luôn bằng 0 nếu tiến trình không đang debug và -1 nếu đang bị debug. Nếu byte đó là -1 thì vùng nhớ tại 0x4010A4 sẽ bị thay đổi). Để bypass thì ta chỉ cần nop instruction đó.

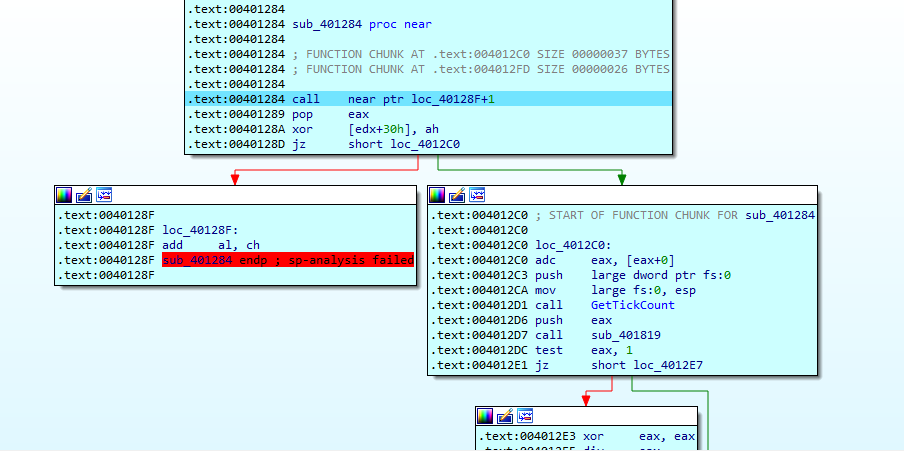
Graphical user interface

Description automatically generated

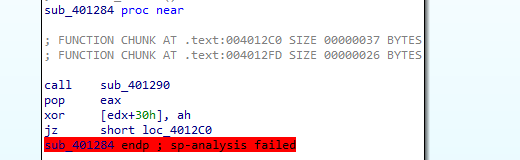
Timeline

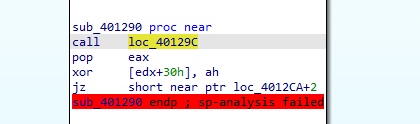
Description automatically generated with low confidence

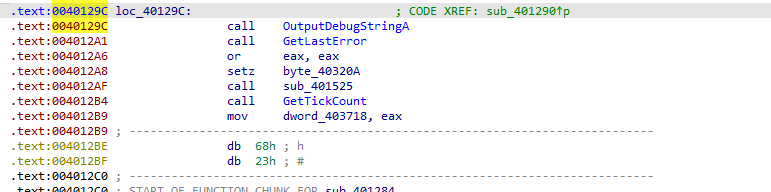
Sau đó, chương trình gọi hàm sub\_401284: hàm này anti disassemble (call loc\_40128F+1). Để xem được assembly phải redefine 0x401290, tại 0x401290 call loc\_40129C, redefine lại thì xem đc assembly



Define lại sẽ thành:



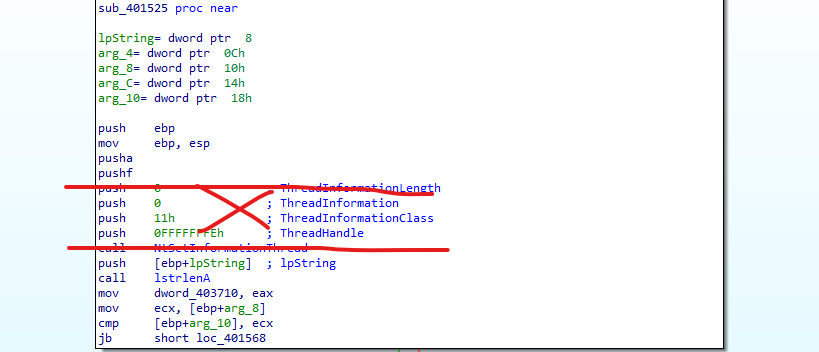




1 kỹ thuật anti-debug sử dụng OutputDebugStringA và GetLastError

nếu chạy trên win vista trở xuống thì sẽ bị check debugger với 2 hàm OutputDebugStringA và GetLastError vì vậy có thể patch hoặc chạy trên windows phiên bản cao để bypass.

Sau đó tiếp tục call sub\_401525, trong này sử dụng NtSetInformationThread(), THREAD\_INFORMATION\_CLASS::ThreadHideFromDebugger (0x11) để hide thread khỏi debugger. Nop đi là xong ☺



Đoạn sau có gọi hàm để check xem từ entry point trở đi 2086 bytes có breakpoint(CC) không, patch nop hết ☺

Hàm này cũng là hàm gen ra flag đầu tiên, debug tiếp thu được đoạn flag đầu tiên là NtQu3ry1nf0rm@t10nPr0(355

Kết thúc hàm trên patch đoạn push offset loc\_401323 thành jmp offset loc\_401323 để đến được phần thực thi của hàm sub\_40134F. Hàm sub\_40134F gọi tới api CheckRemoteDebuggerPresent. Nếu debug\_port bằng 1 thì jump đến đoạn ExitProcess, bypass bằng cách nop đoạn jump đến ExitProcess. debug qua đoạn anti-debug sẽ thu được đoạn flag thứ 2 là NtQu3ry1nf0rm@t10nPr0(355R@!s33xc3pt!on

debug qua đoạn anti-debug sẽ thu được đoạn flag thứ 2 là NtQu3ry1nf0rm@t10nPr0(355R@!s33xc3pt!on.

Đoạn code tiếp theo sử dụng kỹ thuật CreateToolhelp32Snapshot và Process32Next để travel trên snapshot, kiểm tra xem có debugger không, nếu có thì thoát chương trình. Ta bypass bằng cách thay đổi giá trị so sánh của eax sau khi gọi hàm anti-debug thành 0FFFFFFFFh, debug tiếp thu được đoạn flag thứ 3 là NtQu3ry1nf0rm@t10nPr0(355R@!s33xc3pt!onD3bugPr1v1l3g3.

Tại offset 0x0040108C, patch để instuction jbe làm cho ZF == CF để đến hàm sub\_4010C9, debug ở đây để thu được đoạn còn lại của flag gồm 100 ký tự, từ đây đoán theo nghĩa flag thứ 4, 5, 6, 7, 8, 9 là: NtQu3ry1nf0rm@t10nPr0(355R@!s33xc3pt!onD3bugPr1v1l3g3St@ckT1m3CCS3lf-P3BF1ndW1nd0wH1d1ng@nt1-R3v3rs3

Kết quả:

|  |  |
| --- | --- |
| STT | Flag |
| 1 | NtQu3ry1nf0rm@t10nPr0(355 |
| 2 | NtQu3ry1nf0rm@t10nPr0(355R@!s33xc3pt!on |
| 3 | NtQu3ry1nf0rm@t10nPr0(355R@!s33xc3pt!onD3bugPr1v1l3g3 |
| 4 | NtQu3ry1nf0rm@t10nPr0(355R@!s33xc3pt!onD3bugPr1v1l3g3St@ckT1m3 |
| 5 | NtQu3ry1nf0rm@t10nPr0(355R@!s33xc3pt!onD3bugPr1v1l3g3St@ckT1m3CC |
| 6 | NtQu3ry1nf0rm@t10nPr0(355R@!s33xc3pt!onD3bugPr1v1l3g3St@ckT1m3CCS3lf-P3B |
| 7 | NtQu3ry1nf0rm@t10nPr0(355R@!s33xc3pt!onD3bugPr1v1l3g3St@ckT1m3CCS3lf-P3BF1ndW1nd0w |
| 8 | NtQu3ry1nf0rm@t10nPr0(355R@!s33xc3pt!onD3bugPr1v1l3g3St@ckT1m3CCS3lf-P3BF1ndW1nd0wH1d1ng |
| 9 | NtQu3ry1nf0rm@t10nPr0(355R@!s33xc3pt!onD3bugPr1v1l3g3St@ckT1m3CCS3lf-P3BF1ndW1nd0wH1d1ng@nt1-R3v3rs3 |

NtQueryInformationProcess

Raisexception

DebugPrivilege

StackTime

CCSelf

PEB

FindWindowHiding

Anti-Reverse

Waila!