

ENCICLOPEDIA

Del REVERSING

Garzanti Ctrl

A Cura di Ctrl_alt_canc

- La guida segue un ordinato metodo di apprendimento
- Suddiviso in sezioni logiche
- Corredato da almeno un esempio pratico per sezione

Tanti Crackme per esercitarsi!

C.REV.T

Una produzione Ctrl_alt_canc Corporation

INDICE

Il Reversing

▪ Piccola introduzione sulla CPU e come funziona.....	6
▪.....Sistemi di enumerazione (Decimale,binario e esadecimale).....	7
▪ Gli strumenti del reverser.....	9
▪ Concetti base sul reversing	
▪.....PE.....	17
▪.....EP/OEP.....	17
▪.....Packer.....	17
▪.....Dump.....	17
▪.....Subsystem.....	17
▪.....BreackPoint.....	18
▪.....Hendle.....	18
▪.....Trhead.....	18
▪.....CrackME.....	18
▪.....Stack.....	19
▪.....Registri	20
▪.....La memoria.....	21
▪.....RAM & ROM.....	21
▪.....Signature.....	21
▪.....L'importanza del linguaggio.....	23
▪ Concetti di base sulle API e i Breackpoint	
▪.....API Utili.....	26
▪ Istruzioni di linguaggio assembly	
▪.....Cosa sono.....	28
▪.....MOV.....	28
▪.....CMP.....	29
▪.....JMP & Condizionali (Je,Jz...).....	30
▪.....CALL & RET.....	31
▪.....INT	32
▪ Le protezioni	
▪.....Serial.....	33
▪.....Nag	36
▪.....Trial & Scadenze	40
▪.....Packer & Unpaking.....	45

Guida al Reversing • C.REV.T • Ctrl alt canc Corp.

▪.....Normal Unpack.....	46
▪.....Manual Unpack.....	49
▪.....Advance Unpack.....	53
▪.....Accenno ad altri metodi di Packing	56
▪.....Riferimenti esterni a file o registro	63
▪Nozioni e trucchi di reversing	
▪.....Abilitare un bottone	64
▪.....Ricostruire le Import,e cosa sono.....	65
▪.....Effettuare un dump.....	65
▪.....Oltrepassare la subdola tecnica dei caratteri con OllyDbg.....	66
▪Il reversing di Visual basic (5/6).....	67
▪I crack commerciali,reversing serio [tutorial]	
▪.....Pacific assault [Livello ①②③④⑤⑥⑦⑧⑨⑩].....	74
▪.....Axe [Livello ①②③④⑤⑥⑦⑧⑨⑩].....	77
▪.....Dama Delux [Livello ①②③④⑤⑥⑦⑧⑨⑩].....	84
▪I tool utili di Ctrl_alt_canc.....	88
▪Ringraziamenti e saluti.....	89

►INTRODUZIONE



Il reversing.

Parola ricca di mistero,e se utilizzata nella sua variante “volgare”,cracking,diventa un sogno agognato da tutti.

Bisogna però innanzitutto distinguere l'etica dei reverser da quella dei pirati.

I pirati,rubano videogame o software e li distribuiscono craccati per internet.

I reverser sono invece coloro che craccano i suddetti software ma senza fine di lucro.

Nel mondo del reversing infatti mai e poi mai imparerete a “crakkare” un videogioco prendendo da internet la “crack” e piazzandola nella cartella di gioco. Questo **NON** è reversing,è lamering.

Tuttora sono circondato di gente che afferma di essere cracker. Fantastico. Parliamo un po',dimmi che ne pensi delle nuove protezioni che stanno pian piano introducendo le case di software? I keygen sono sempre più inutili,meglio sradicare la protezione se possibile piuttosto che carpirne l'algoritmo. Difficile runnare l'eseguibile protetto da hardware fingerprint su tutti i PC dopo averlo cracciato.

La reazione “tipo” di uno di questi individui è: *“Di che cazzo parli? Cioè,ma hai visto che cavolata craccare san andreas? Cioè potevano anche sforzarsi di più...”*

Eggià,per fortuna che mettono i readme altrimenti non saresti neppure capace di copiare l'eseguibile.☺

Essendo questo il livello medio di un utente che si limita a scaricare eseguibili craccati da altri,ho deciso di stendere questa guida.

Tuttora le autorità non distinguono il pirata,che si limita a scaricare le crack usate da altri,e il SINCERO fine di studio del reverser. (A me non frega un cazzo avere programmi a pagamento craccati,tanto ci sono spessissimo i corrispondenti open source).

Il reversing è una forma d'arte, e se praticata con perseveranza e senza ossessione può diventare un ottimo calmante per la mente.

Combattiamo per questo ideale.

Purtroppo devo avvertirvi che per inoltrarvi in questo mondo, è necessario avere almeno dei rudimentali fondamenti anche su aspetti meno pratici del reversing (quindi mettete via quegli schifosi debugger che state scaldando,sperando che si inizi subito con un bel crack commerciale).

Infatti i fondamenti a cui mi riferisco sono quelli della programmazione e del funzionamento del vostro PC.

No,non limitatevi all'apparenza.

Non un hard disk da cui si leggono i dati,uno schermo con cui leggere i documenti e via dicendo.

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Le nozioni che seguiranno sono abbastanza complicate (ma da me esplicate nel modo più semplice possibile).

Vi premetto che queste informazioni sono necessarie al reverser, sono l'aria che respira, altrimenti, in mancanza di questi fondamenti sarà soltanto un povero user che tenta di seguire malamente un tutorial.

Infatti la guida segue un ordinato metodo di apprendimento, suddiviso in sezioni logiche, corredata da almeno un esempio pratico per sezione.

Non scoraggiatevi e studiate!

NB.: Se sei un lamer che legge questa guida per crakkare un videogame o un programma perchè sei furbo e non hai voglia di pagarlo, fa pure. Tanto non ci capirai un cazzo XD

NBB.: La guida è stata corretta e revisionata con attenzione da Predator (<http://nexenteam.net/>) che con pazienza ha dato il supporto tecnico necessario ad una stesura semplice ed efficace del documento che vi accingete a leggere.

►PICCOLA INTRO ALLA CPU



Il vostro computer ha un centro ove vengono svolte tutte le operazioni.

Questo è la CPU (Central Process Unit).

Ci sono molti tipi di processori. In particolare noi ci occuperemo di quelli Intel.

Questa fabbricuccia di processori divenne incredibilmente famosa negli anni 70 quando incominciò a produrre microprocessori sempre più veloci e sempre compatibili. Cosa vuol dire compatibili? Vuol dire che un programma che girava sul vecchissimo 8086 gira pure su un pentium.

Questo perchè l'interprete dei comandi binari è rimasto lo stesso. Il linguaggio con cui si programmava su quelle macchine (e sto parlando del linguaggio alto cioè il linguaggio base su cui tutto poggia) è rimasto costante. Al contrario varie altre case produttive creavano processori ma che non comunicavano l'uno con l'altro.

Anche se può sembrare il contrario, la CPU fa eseguire solo una operazione alla volta. Però ci sono da dire 2 cose:

- 1 La velocità di funzionamento sembra che le faccia partire in contemporanea
- 2 La CPU ha la possibilità di mettere in Standby un'azione.

Questo crea l'effetto di simultaneità.

In pratica è come un giocatore di ping pong che corre intorno al tavolo per prendere la pallina senza farla cadere, dove il giocatore è la CPU e la pallina sono i processi.

La reale simultaneità c'è quando si usano delle periferiche esterne. Qui il discorso è diverso perchè la CPU di occupa soltanto di impostare la lettura della periferica in questione. Poi tutta la prelevazione di informazioni è fatta dalla DMA. Qui c'è la simultaneità reale perchè una volta impostata la lettura della periferica, la CPU si preoccupa di altro.

► SISTEMI DI ENUMERAZIONE

be 75 b1
7c 55 eb
2b 2e 18
b4 e0 00
90 52 e3

Io parto dal presupposto che voi sappiate già contare in base 10!

Cosa vuol dire contare in base 10? Vuol dire quello che avete sempre fatto ogni volta che contavate. $1+1=2$ $60-40=20$ $4+4=8$

Naturalmente, si può contare in molte altre basi, 2, 3, 4 ecc.

Quelle che ci interessano a noi sono la 2 e la 16. Queste vengono rispettivamente dette binaria, esadecimale.

Non sappiamo che i numeri nella base decimale vanno dallo 0 al 9 per poi ripetersi con un 1 davanti che poi diventa un 2 e così via.

Cioè 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14.... Si va dallo zero al nove per poi ripetersi nel 10 fino al 19. Poi il 20 fino al 29, ecc.

Supponiamo che ora io fermi i numeri non al 9 ma all'8.

Si avrebbe una condizione del genere:

0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, ...

Se quindi ora devo sommare 2+2 ottengo sempre 4. Ma se sommassi 4+5, ottengo non 9 (perchè non c'è!) ma ottengo 10. Perchè se sommo 4+5 devo ottenere il numero dopo l'otto che nel nostro caso è il 10.

Se poi il al posto di levare solo il nove e limino anche altri numeri potrei arrivare ad averne solo più 2: lo 0 e l'1. Se infatti tolgo 9, 8, 7, 6, 5, 4, 3, 2 mi rimangono solo i primi 2.

Cosa vuol dire questo? Che contando non posso superare l'1. Se faccio quindi 1+1, non ottengo 2 ma ottengo 10. Perchè? Perchè la sequenza sarebbe:

0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, ...

Che in decimale vuol dire:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ...

Quindi ora sappiamo che 1+1 è uguale a 10. ☺

Questo metodo di enumerazione è molto utile, se non essenziale per inoltrarsi nel reversing. Ma un altro metodo forse ancora più essenziale è la base esadecimale. Questa base è come dice la parola stessa, in 16. Quindi $1+1=2$, $4+5=9$ ma $12+4=22$!

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Come funziona? Semplice, al posto che togliere numeri (come per le basi precedenti) qui si inseriscono nuovi numeri. Questi sono:

1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,10,11,12,13,14,15,16,17,18,19,1A,1B,1C,1D,1E,1F,20...

Ecco una comoda tabella riassuntiva:

Desimale	Binario	Esadecimale
1	0	1
2	1	2
3	10	3
4	11	4
5	100	5
6	101	6
7	110	7
8	111	8
9	1000	9
10	1001	A
11	1010	B
12	1011	C
13	1100	D
14	1101	E
15	1110	F
16	1111	10
17	10000	11
18	10001	12
19	10011	13
20	10100	14

Questa piccola tabellina vi mostra i valori numeri nelle tre basi che a noi interessano. Poi ci sono infinite altre basi che però non sono utilizzate dal computer.

Ma perchè il computer usa le basi 2 e 16?

Il computer come voi sapete, lavora il linguaggio macchina. Ciò vuol dire che tutto ciò che voi vedete in verità non è così com'è, ma è un'insieme di 1 e di 0.

Per questa ragione, quando si vuole comunicare col computer in modo "terra terra" bisogna livellarci al suo linguaggio. Lui infatti non capisce assolutamente nulla, egli 1 e gli zero

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

corrispondo al flusso di corrente sui suoi transistor (1) e nessun flusso (0). Per quanto riguarda la base 16, questa è perchè all'interno del computer, i file funzionano e vengono trasmessi a piccoli pacchetti di 8 unità ciascuno (8 bit) Se noi moltiplichiamo 8 per 2 otteniamo 16. Perchè non usiamo 8 lo capiremo più avanti.

Per finire:

- h significa un numero esadecimale 16h
- b significa un numero binario 101b
- per i numeri decimali non si usa niente

► GLI STRUMENTI DEL REVERSER



Eccoci. Un buon reverser sicuramente deve essere dotato di un minimo di intelligenza e perspicacia, ma una suite di tool come quelli che seguono può sicuramente fare comodo e abbreviare il lavoro di tutti.

NB: I tool presenti sono solo un punto di partenza suggerito dal sottoscritto. NON è detto che siano in assoluto i migliori, con l'esperienza ognuno di noi troverà il metodo e il programma che più si addice al nostro modo di procedere. Talvolta si dovrà imparare anche a scriversi i programmi perché non sempre sono disponibili nella rete. (Io pure ho creato una piccola serie di utility che alla fine della guida vi presenterò).

Se la vostra domanda è: *“ma cavolo! Farsi un programma? Io non so programmare!”*. Spiacente amico, ma è necessario sapere bene almeno un linguaggio di programmazione. I motivi sono 2:

a) Per creare dei keygen è necessario saper programmare ☺

b) Per capire la logica di un programma da reversare, bisogna entrare nella mente del programmatore per capire cosa avrà pensato e come ci vuole ingannare. Come minimo bisogna saper programmare appunto. Non importa quale linguaggio, tanto come dico io, sono tutti fratelli.

Primo di tutti in assoluto, il debugger.

DEBUGGER

Cosa è un debugger? È un tool molto complesso, che permette di disassemblare il codice dell'eseguibile preso in analisi, fino a risalire al linguaggio assembly che lo compone. (Per la trattazione completa dell'assembly recarsi nell'apposita sezione).

Non solo questo, bensì permette di seguire il “movimento” di un eseguibile passo-passo debuggandolo appunto.

Permette di settare dei punti di stop (breakpoint), di modificare le istruzioni assembly con

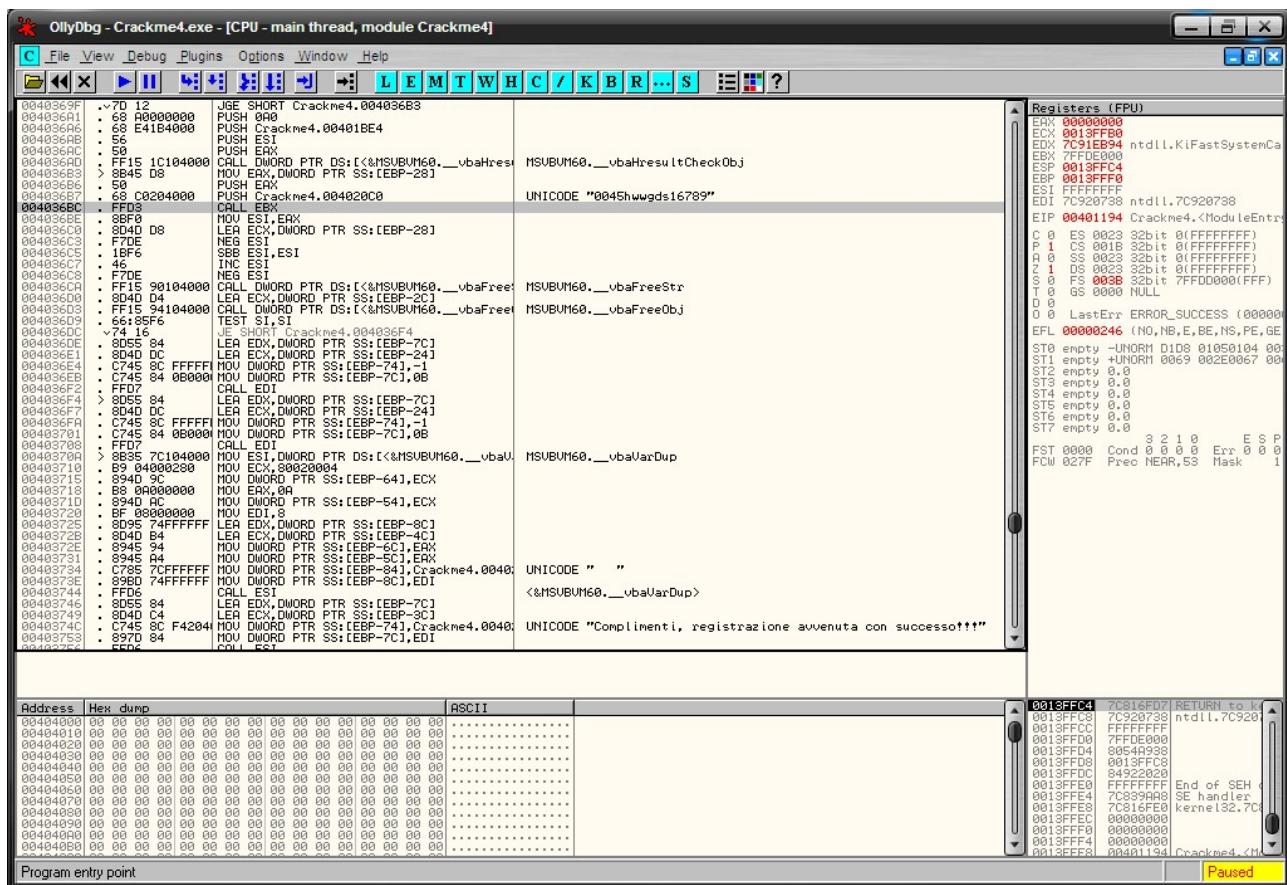
Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

altre, imposte da noi, magari proprio per crakkarlo e far funzionare senza registrazione il programma preso in esame.

Permette di visualizzare la memoria,i registri lo stack e un sacco di altre cose interessanti (Non preoccuparsi se per ora molti dei termini citati sono a voi sconosciuti,avrete modo di impararne il significato proseguendo nella lettura).

Un debugger interessante, funzionale e open source è sicuramente OllyDbg. L'unica pecca è che lavora a ring 3, cioè al livello di permessi di un normale programma, e questo in alcuni casi può dare problemi.

Comunque, ecco ollydbg in tutto il suo splendore:



La parte principale della finestra presenta il disassemblato del programma. Nella barra in alto ci sono i comandi per runnare il programma sotto debug (F9), e steppare (andando passo passo per analizzare il comportamento poco per volta) (F8). La colonna di destra contiene i registri in tempo reale, la sezione bassa e lunga è il dump della memoria, mentre l'ultima cella in basso a destra è lo stack.

Olly è davvero facile da usare. Per settare un breackpoint è sufficiente un doppio click sulla istruzione, la quale diventerà rossa. Per modificare l'istruzione è sufficiente un doppio click sulla parte centrale.

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Essenziale per l'uso di olly è anche la command line, per settare breackpoint al volo senza doverli ricercare all'interno del codice disassemblato.

Packer detector

Davvero importante questo tool, specie quando si ha a che fare con il codice protetto da un packer. Infatti un programma di questo tipo permette in genere di ottenere il nome del packer, il linguaggio con cui il è stato programmato l'eseguibile e varie info generali sul formato PE.

Uno dei più interessanti packer detector è il mitico RDGpacker detector.

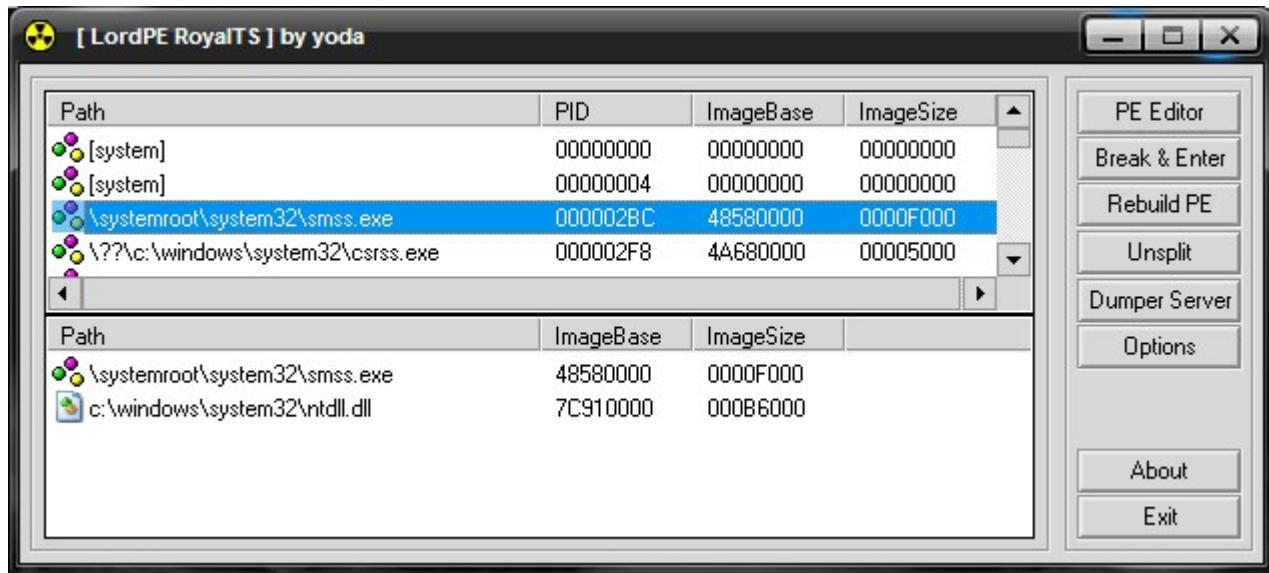


PE Analyzer

Un PE analyzer serve in generale per ottenere tutte le info sul PE dell'eseguibile preso in esame. E' utilissimo per ricostruire un eseguibile funzionante per esempio dopo un dump.

Uno dei migliori è sicuramente LordPE, la cui funzione "Rebuild PE" è in assoluto unica al mondo per funzionalità e capacità.

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.



Hex Editor

Sinceramente lo trovo abbastanza inutile dato che l'hex editor è di norma integrato nei debugger e decompilatori, tuttavia talvolta è comodo avere a disposizione un tool a parte.

Serve per analizzare gli exe a livello binario per cui ben poco possiamo modificare senza aver prima analizzato il programma da craccare dentro un debugger.

Uno molto comodo e Opensource è Frhed:

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

The screenshot shows a hex editor window titled "[C:\DOCUMENTI\utente\Desktop\Download\LPES-RTS\LordPE.EXE] - fr...". The menu bar includes File, Edit, View, Options, Bookmarks, and Help. The main pane displays memory dump data from offset 0x000000 to 0x001200. The left column shows the memory address, and the right column shows the ASCII representation of the bytes. The file starts with the MZ header, followed by various sections of code and data, including strings like "Win32", "only!", "Rich", and "LordPE". The status bar at the bottom indicates the current offset is 0x000000, the file is unsigned, and its size is 192512 bytes.

Disassemblatore

Altro essenziale strumento, disassembla in codice macchina un eseguibile. Il migliore in assoluto, forse un po' difficoltoso da usare per chi è alle prime armi è sicuramente IDA Pro. Che permette di fare una miriade di cose. Tra le più importanti ricordiamo:

Istruzioni logiche sotto forma di schema

Hex editor integrato relazionabile col disassemblato (e viceversa)

Import

Export

Funzioni

Ricerca stringhe di testo

Funziona anche da debugger e permette l'alterazione dei registri "on the fly"

Il programma monitora continuamente il flusso dell'eseguibile (da notare la barra colorata in alto) e permette di esplorare call e funzioni in maniera semplice e veloce.

Eccolo:

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

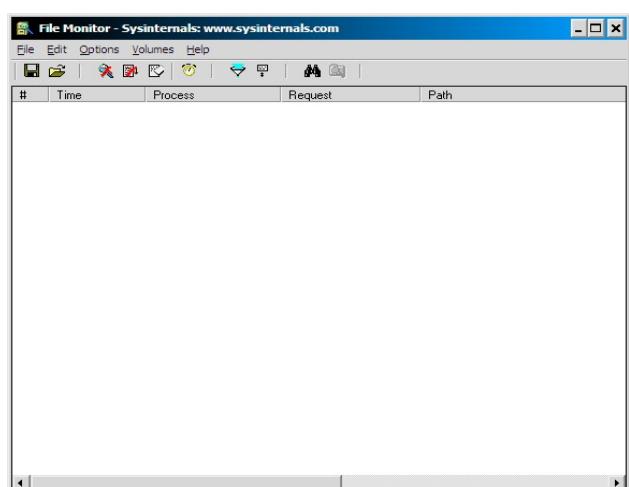
The screenshot shows the IDA Pro interface with the title bar "IDA - C:\Documents and Settings\utente\Desktop\Crackme4.exe - [IDA View-A]". The menu bar includes File, Edit, Jump, Search, View, Debugger, Options, Windows, Help. The toolbar has various icons for file operations, search, and analysis. The status bar at the bottom shows "AU:disabled" and "Disk: 4GB". The main window displays assembly code:

```
• .text:0040372E      mov    [ebp-6Ch], eax
• .text:00403731      mov    [ebp-5Ch], eax
• .text:00403734      mov    dword ptr [ebp-84h], offset asc_402160 ; ""
• .text:0040373E      mov    [ebp-8Ch], edi
• .text:00403744      call   esi ; __vbaVarDup
• .text:00403746      lea    edx, [ebp-7Ch]
• .text:00403749      lea    ecx, [ebp-3Ch]
• .text:0040374C      mov    dword ptr [ebp-74h], offset aComplimentiReg ; "C
• .text:00403753      mov    [ebp-7Ch], edi
• .text:00403756      call   esi ; __vbaVarDup
• .text:00403758      lea    ecx, [ebp-6Ch]
• .text:0040375B      lea    edx, [ebp-5Ch]
• .text:0040375E      push   ecx
• .text:0040375F      lea    eax, [ebp-4Ch]
• .text:00403762      push   edx
• .text:00403763      push   eax
• .text:00403764      lea    ecx, [ebp-3Ch]
• .text:00403767      push   40h
• .text:00403769      push   ecx
• .text:0040376A      call   ds:rtcMsgBox
• .text:00403770      lea    edx, [ebp-6Ch]
• .text:00403773      lea    eax, [ebp-5Ch]
```

The status bar at the bottom shows "00003734 00403734: sub_403708+2C". A message in the status bar says "The initial autoanalysis has been finished. Search failed.".

RegMon & Filemon

Due ottimi programmi volti a monitorare l'accesso ai file e al registro di un determinato programma.

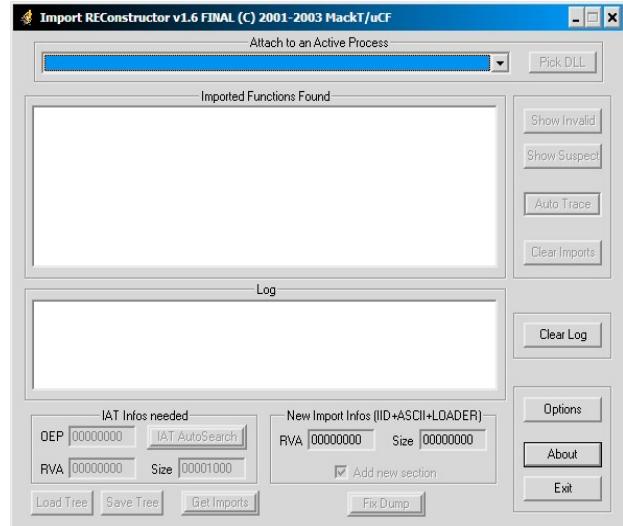


Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

ImpRec

Ricostruisce le Import dopo un dump.

Permette anche il comodo uso di plug in per abbreviare il lavoro di ricostruzione.

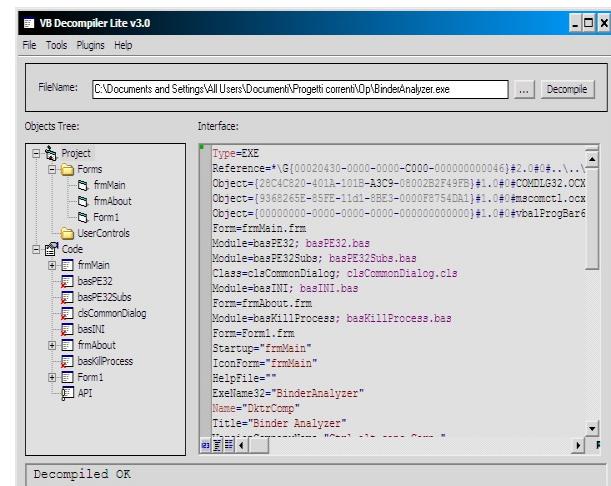


VbDecompiler

Non molto utili i decompilatori, dato che non funzionano realmente.

Tuttavia quello per vb6 è davvero molto utile

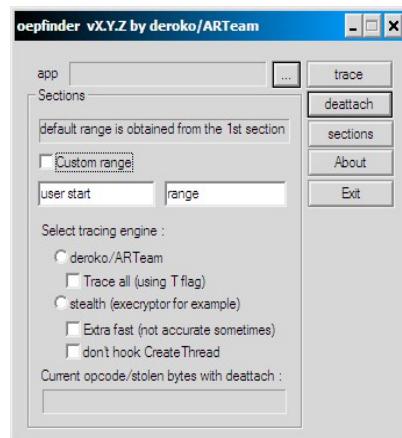
specie per risalire alle API usate oppure al codice diretto di un tasto.



OepFinder

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Un tool veramente indispensabile per risalire all'OEP del programma preso in esame.



Naturalmente questi sono solo consigli,in realtà i tool li cercherete da soli quando ne avrete bisogno.Ciò non esclude che spesso vi troverete a scriverne di personali,un po' per sfizio un po' per reale necessità.

► CONCETTI DI BASE SUL REVERSING ABC

► PE (Portable Executable)

Il formato PE è un formato standard utilizzato dai sistemi windows per “capire” un file di tipo .exe (eseguibile) .dll (libreria) o .sys (driver). La sigla “portable” è soltanto un identificativo per far capire che tutti i PE possono funzionare su ogni piattaforma windows a 32 o 64 bit.

Dentro un PE sono dislocate molteplici informazione per la sua corretta interpretazione, tra cui le Import e Export Tabl, che sono l'elenco delle API utilizzate nel programma e le relative funzioni, le risorse (l'icona, l'interfaccia grafica, ma anche le stringhe di testo) e le TLS Table.

► EP/OEP

L'EP (Entry Point, punto di ingresso) di un programma, non è altro che il punto di inizio da cui partono tutte le routine del programma, da cui windows parte per far funzionare un eseguibile. Spesso un EP può essere camuffato (per esempio con un exe packer) o addirittura virtualizzato (come nel caso di execryptor).

L'OEP invece sta per Original Entry Point, che è un'altra denominazione per chiamare l'EP nascosto... ed è il sogno di tutti gli unpacker...)

► PACKER / PROTECTOR

E' un software capace di comprimere un eseguibile e proteggerlo (si, x modo di dire) dagli okki dei reverser.

Alcuni packer sono molto semplici come UPX o ASP pack i quali comprimono l'eseguibile e al momento dell'unpack lo scomprimono in memoria, e da lì lo eseguono, senza che il reverser (se non con un debug) possa vederne il contenuto.

Altri invece, denominati protector, perché proteggono il codice come TheMida utilizzano un hardware fingerprint, un identificativo univoco del PC a cui viene associata una sola key valida. Armadillo invece virtualizza anche delle parti di codice per rendere ancora più difficile raggiungere l'OEP.

Alcuni anche usano delle avanzate o basilari tecniche antidebug.

Armadillo apre dei thread che rivelano la presenza di un debug, ma esistono anche altre soluzioni... la più utilizzata (forse neppure più però) è l'API IsDebuggerPresent la quale restituisce al prog un valore 1 se il processo è sottodebug e un valore 0 se non lo è... facile capire come viene implementata quest'aprotezione, ma è altrettanto facile superarla e identifierla (Vedi il mio prog e i vari plugin di olly)

► DUMP

Il dump, da non intendere come dump esadecimale, è una operazione che si esegue spesso in ambito di reversing, per esempio per liberare un software da un packer. Consiste nell'estrare una parte del codice binario (generalmente appena scompresso in memoria nel caso dei packer) e renderlo eseguibile direttamente da quel punto, come un exe a sé stante. Spesso i dump non funzionano al primo colpo, poiché o la imagesize dell'exe è corrotta, o le Import sono state "distrutte", o per altri motivi. Per questo i dump vanno sempre ricostruiti, affinché l'OS possa comprenderli.

► Subsystem

Indica il tipo di applicazione con la quale si ha a che fare (tutti i packer detector e PE analyzer ne portano traccia). Per esempio un'applicazione può essere una Library (un Dll), una applicazione GUI (con interfaccia grafica) o Win32 console (in cui non è presente la GUI, la riga di comando per intenderci).

► BreackPoint

E' un punto di interruzione che viene settato dentro un debugger per analizzare una determinata routine o chiamata alle API (vedi sezione API).

In genere per settare breackpoint è sufficiente fare un doppio click nella linea laterale a sinistra del disassemblato, oppure (dentro ollydbg) è sufficiente dotarsi della commandline di OllyDbg e far precedere al nome dell'API la sigla bpx (vedi sezione API).

► Handle

E' un valore univoco che viene attribuito ad ognuna delle finestre che abbiamo aperte nel PC. Spesso viene usato dai programmi anti reverser per individuare un debugger o simili.

► Trhead

Un Trhead è un processo figlio di un processo più grande. Il trehead ha un identificativo univoco, e una creation flag, che corrisponde al tipo di trhead. Per esempio, può esserci un trehead con creation flag "CREATE_SUSPENDED" che crea trhead addormentati, e possono essere chiusi solo da appropriati comandi.

Anche i trhead sono dei diversivi che possono essere utilizzati per proteggere un'eseguibile, per esempio Execryptor, o lo stesso Pirupiru crack me di predator che trovate nella sezione crackME.

► CrackMe

Un crackME è la palestra dei reverser. Si esercitano su questi programmi-prova per evitare i

compiere illegalità su programmi commerciali.

Esistono vari tipi di crack me:

Patch

Consiste nel modificare una parte di assembly utile per rendere utilizzabile un programma senza protezioni di sorta.

Keygen

Prevedono in genere protezioni tramite password, e richiedono la scoperta dell'algoritmo che genera la chiave e la eventuale scrittura di un KeyGen

Unpack

Prevedono l'uso di un packer (noto o scritto dal creatore del crackme) per proteggere l'eseguibile da un reverse. Lo scopo è quello di ottenere un Dump funzionante

Enable

Generalmente hanno alcuni pulsanti o controlli disabilitati da rendere attivi definitivamente.

Una trattazione completa delle varie protezioni la troverete durante la lettura di questa guida.

► Lo Stack

Lo stack è un'area di memoria dove possono essere salvati "momentaneamente" (perchè memoria temporanea) i registri della CPU. Questa operazione di salvataggio può essere fatta o attraverso l'istruzione PUSH o attraverso l'istruzione CALL. Il primo metodo è voluto direttamente, l'altro è indirettamente.

Gli stessi dati vengono ripresi sia direttamente attraverso istruzioni POP che indirettamente (relativamente al registro IP o alla coppia CS:IP) attraverso istruzioni RETN o RETF (RETurn Near o RETurn Far).

Lo Stack può trovarsi sia nello stesso segmento del programma (come nel caso dei files .COM) che in un altro segmento (files .EXE); in ogni caso i dati nello stack vengono puntati dalla coppia di registri SS:SP e i dati vengono inseriti dall'ALTO verso il BASSO, cioè con valori di SP sempre decrescenti via via che si inseriscono nuovi dati nello stack.

Spesso nel reversing lo stack viene utilizzato per prelevare informazioni utili circa seriali, risultati di routine e altro.

Ricordiamo che possiamo esplorare lo stack direttamente da OllyDbg (vedi sezione: Tool del reverser).

► I Registri

Per quanto riguarda la base 16, questa è perchè all'interno del computer, i

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

*file funzionano e vengono trasmessi a piccoli pacchetti di 8 unità ciascuno (8 bit)
Se noi moltiplichiamo 8 per 2 otteniamo 16. Perchè non usiamo 8 lo capiremo più
avanti. (Ricordate? Ecco la spiegazione)*

Questi bit vanno in base 8 e sono quelle cose che contengono concretamente tutto ciò che il computer ha e visualizza.

Questi bit funzionano grazie a dei registri che sono i contenitori dei numeri binari. Questi sono tanti e sono continuamente modificati dall'INTEL, ma sta di fatto che sono compatibili l'un l'altro tra i vari computer.

I registri possono contenere un numero binario che va da 00000000 a 11111111 che, in parole povere vuol dire da 0 a 255.

Abbiamo quindi detto: i bit trasportano le informazioni. Le informazioni sono in binario e questi numeri sono contenuti nei registri dei vari bit. Non devete però pensare che in ogni bit ci siano dei registri! E' il contrario! In ogni registro ci sono 8 bit. Ma dato che non possiamo slegarli l'un l'altro, non preoccupatevi di sapere esattamente la struttura architettonica di un microprocessore.

Ogni registro ha una struttura simile. Ecco uno schema:

Registri a 8 bit	AH AL	BH BL	CH CL	DH DL
Registri a 16 bit	AX	BX	CX	DX
Registri a 32 bit	EAX	EBX	ECX	EDX

Dovete, all'incirca impararlo a memoria!

Cosa sono i registri a 16 bit? Nulla sono due registri presi assieme. Dato che si usa spesso e volentieri prenderli a coppie e considerarli uno solo, si conta anche a 16.

Questi comunque sono i registri base. Poi esistono altri tipi di registri nati successivamente allo 8088 e per questa ragione sono detti speciali:

-Registri speciali a 16 bit:

IP (Instruction Pointer) ovvero puntatore alle istruzioni.

BP (Base Pointer) ovvero puntatore alla base di dati.

SI (Source Index) ovvero Indice Sorgente di dati.

DI (Destination Index) ovvero Indice Destinazione di dati.

SP (Stack Pointer) ovvero puntatore allo stack.

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

-Registri di segmento a 16 bit:

- CS (Code Segment) Punta al segmento dove si trovano le istruzioni
- DS (Data Segment) Punta al segmento dei dati
- ES (Extra Segment) Punta al segmento dati (extra)
- SS (Stack Segment) Punta al segmento di stack

Un metodo pratico per visualizzare il contenuto dei registri, è di aprire il prompt dei comandi e digitare debug, poi premere r e invio

```
Microsoft Windows XP [Versione 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>debug
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1562 ES=1562 SS=1562 CS=1562 IP=0100 NV UP EI PL NZ NA PO NC
1562:0100 0000 ADD [BX+SI],AL DS:0000=CD
-
```

► LA MEMORIA

Ogni singolo file, ogni singola esecuzione di programma, tutto quanto richiede memoria nel vostro computer. Tutto è memoria. Noi sappiamo per adesso che la memoria è formata da bit. Se noi raccogliamo a gruppi di 8 i bit otteniamo un byte. I byte sono dei pacchetti di memoria che non contengono altro che 0 e 1 in sequenza. Anche i byte ovviamente sono dei registri. Questi in particolare sono:

CS:IP	Indirizza la Memoria dove risiedono i programmi
SS:SP	Indirizza la Memoria dove risiede lo stack
DS:[BP BX SI DI]	Indirizza la Memoria dove risiedono i dati
ES:[BP BX SI DI]	Indirizza la Memoria dove risiedono i dati

A partire dal 386 esistono altri due registri di segmento FS e GS che possono essere utilizzati per indirizzare la memoria Dati. In ultima analisi quindi, sia un programma che qualunque tipo di dato è memorizzato sotto forma di singoli bit (stato 0 o 1) e indirizzabili a gruppi di 8 (un Byte per volta) o a gruppi di più Bytes secondo il seguente standard:

1 Byte

1 Word (pari a 2 Bytes)

1 DoubleWord (pari a 4 Bytes)

1 QuadWord (pari a 8 Bytes)

1 TenByte (pari a 10 Bytes)

Queste sono raggruppazioni standard.

► RAM & ROM

Il vostro computer ha della memoria. Questa è suddivisa in 2 parti per distinte, la RAM e la ROM. La RAM (Random Access Memory o meglio memoria di accesso casuale) è quella memoria che viene utilizzata dal computer per ricordare file momentanei o per far partire dei programmi. Questa memoria è temporanea nel senso che se spegnete il computer quella memoria si cancella automaticamente e perdetе tutto.

Poi c'è la memoria ROM (Read Only Memory o Memoria di sola lettura). Questa è quella permanente. Quella ove stanno i vostri file salvati, come i vostri programmi, come tutto quello che avete ogni volta che ravviate il computer.

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Nella ROM c'è il BIOS. Il BIOS (Basic Input-Output System) è un “programma” che ha il compito dell'accensione del computer. Verifica i dispositivi presenti, esegui il POST (Power on Self Test) e infine carica il sistema operativo, dando libero sfondo alla MRB. La MRB è una parte della memoria ove sono inseriti i file (quasi inaccessibile) che permettono l'accensione del vostro sistema operativo.

Dopo aver fatto tutto ciò il BIOS lascia tutto il controllo della macchina al vostro sistema operativo. Ma il BIOS continuerà sempre ad essere presente e offre una serie di servizi a cui fanno spesso riferimento gli stessi sistemi operativi. Tra i tanti quelli che ci interessano a noi ci sono gli Interrupt.

Come tutta la memoria, la RAM e la ROM sono nient'altro che insieme di bytes. Questi saranno formati come tutti da 0 e da 1. Qui c'è tutto il funzionamento della macchina.

Ogni singolo Byte della RAM (e della ROM) ha un INDIRIZZO UNICO, e a questo indirizzo si fa riferimento per prelevare o memorizzare i dati o i programmi.

Gli indirizzi della RAM (e della ROM) sono puntati da registri specifici del Microprocessore.

Per quanto riguarda gli indirizzi dei programmi, si usa normalmente la coppia di registri CS:IP. Per quanto riguarda l'uso dei file di dati, si usano i registri di segmento (DS - ES - FS e GS) in accoppiata con i registri (BX - BP - SI - DI) in diverse combinazioni.

Quindi per l'esecuzione o l'apertura di un programma ci sono i CS:IP mentre per gli altri, cioè i file che contengono dati (cioè tutti quelli che non sono programmi), ci sono i registri di segmento (DS - ES - FS e GS) da utilizzarsi con i registri (BX - BP - SI - DI) in diverse combinazioni.

Poi c'è lo STACK (vedi paragrafo *stack*)

Con gli ultimi microprocessori si è cercato di separare e di proteggere accuratamente l'area destinata ai dati e quella destinata ai programmi. E' però possibile modificare attraverso un programma in assembler la destinazione fisica del programma. Questo è ciò che realmente fanno molti virus creati in assembler. Si collocano nella zona dati rendendosi quasi del tutto invisibili agli antivirus.

Per quanto concerne la memoria RAM, questa è gestita dal sistema operativo che la cede ad un programma piuttosto che ad un altro.

► Signature

La diretta traduzione dall'inglese della parola “signature” è “Firma”.

Tutti i software di protezione degli eseguibili, ma anche i virus e qualsiasi programma informatico distribuito su larga scala, contiene al suo interno un “pattern” di byte che lo rende distinguibile.

I packer soprattutto sono famosi per questa caratteristica, e, una volta riconosciuti, è poi facile

addottare le contromisure.

I moderni packer detector (pure il mio) attuano delle scansioni dei byte dell'eseguibile da analizzare confrontando simultaneamente alcuni pattern famosi presenti in un database. Ecco un esempio di database di "firme" dei vari packer. (tratto dal mio packer detector)

```
[1]
PackerName = APatch GUI 1.x
Signature =
5231C0E8FFFFFFFC75E83EE0F05A000000059E901000000C0FC29D18D3C0601C8680F214
000A9??????370F84D60000003D??????771835??????0F8462010000330747BB0020000083
E90574D64FA1A72040008B15B2204000A3B22040001D00305B92040008915B9204000E812
00000F7F18915A7204000E901
[2]
PackerName = UPX 1.93 - > upx.sourceforge.net
Signature =
60BE???04?008DBE????F?FF
[3]
PackerName = ARM Protector 0.1 -> SMoKE
Signature =
E8040000008360EB0C5DEB054555EB04B8EBF900C3E8000000005DEB010081ED5E1F4000
EB0283098DB5EF1F4000EB028309BAA3110000EB01008D8D923140008B09E81400000083E
B01008BFEE800000005883C00750C300EB04584050C38A0646EB0100D0C8E81400000083E
B01002AC2E800000005B83C30753C300
[ecc...]
```

(i punti di domanda sono per sostituire dei byte variabili, che non sono fissi)

Naturalmente le firme possono essere falsificate, ma un reverser esperto riconosce a prima vista un packerdetector dopo aver analizzato qualche riga di codice, dato che spesso hanno un moto molto personale di proteggere i programmi.

► L'importanza del linguaggio

Spesso (ma non sempre) il linguaggio usato è il C++ (per i programmi commerciali).

Le differenze tra i linguaggi, in ambito di reversing sono importantissime. Sapere con quale linguaggio è stato prodotto un eseguibile è necessario per sapere come operare. Per esempio, dobbiamo sapere che il C++, l'asm e il delphi utilizzano le chiamate alle win32 API. (vedi paragrafo "*Concetti di base sulle API e i Breackpoint*")

Queste sono piuttosto comuni, e facili da trovare, sapendo che un MsgBoxA è sempre una messagebox, e via dicendo.

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Caso a parte è il Visual Basic che essendo un linguaggio interpretato e non compilato, non fa direttamente uso delle api se non su esplicita richiesta (Vedi paragrafo "Visual Basic").

Ecco quindi che un breackpoint su un MsgBoxA sarebbe inutile,e ci farebbe credere che non si siano messagebox.

O addirittura in ambito di unpacking,conoscere il linguaggio di programmazione è necessario er ricostruire un EP (vedi sezione “*concetti base sul reversing*”) corrotto

In genere i packer detector risalgono anche al linguaggio di programmazione,ma io ho creato uno strumento apposito che compie eccellentemente il suo dovere (Anche i linguaggi di programmazione hanno delle signature!). Per maggiori info vedere la sezione finale della guida sui miei tools.

E' necessario conoscere almeno a occhio un EP da un codice normale,questo perchè tutti gli EP hanno caratteristiche simili.

Ecco una tabella riassuntiva da *studiare*:

Linguaggio	Entry Point	Descrizione
VB5/6	<pre>JMP DWORD PTR DS:[40103C] JMP DWORD PTR DS:[40102C] JMP DWORD PTR DS:[401034] JMP DWORD PTR DS:[401064] ADD BYTE PTR DS:[EAX],AL PUSH CM6-Upx2.004B1340 CALL CM6-Upx2.004B1144 ADD BYTE PTR DS:[EAX],AL ADD BYTE PTR DS:[EAX],AL ADD BYTE PTR DS:[EAX],AL XOR BYTE PTR DS:[EAX],AL ADD BYTE PTR DS:[EAX],AL INC EAX ADD BYTE PTR DS:[EAX],AL</pre>	<p>MSVBVM60.EVENT_SINK_QueryInt MSVBVM60.EVENT_SINK_AddRef MSVBVM60.EVENT_SINK_Release MSVBVM60.ThunRTMain</p> <p>JMP to MSVBVM60.ThunRTMain</p> <p>L'EP è preceduto da tutte le funzioni del programma (es: MSVBVM60.EVENT_SINK_AddRef) ed è seguito da una call alla ThunRTMain.</p>
C++	<pre>JMP UWURU PTR DS:L<&USER32.DdeHoccessDat JMP DWORD PTR DS:[<&USER32.DdeQueryStri JMP DWORD PTR DS:[<&USER32.DdeCreateDat PUSH EBP MOV EBP,ESP PUSH -1 PUSH firefox.00AB5570 PUSH <JMP.&MSVCRT._except_handler3> MOV EAX,WORD PTR FS:[0] PUSH EAX</pre>	<p>USER32.DdeHoccessDat USER32.DdeQueryStri USER32.DdeCreateDat</p> <p>SE handler ins:</p> <p>Inizia SEMPRE con un Push EBP e prosegue pressochè uguale alla figura qui a sinistra.</p>
Delphi	<pre>004E4F00 A04E4F00 00 00 00 00 00 3CD54F00 55 8BEC 83C4 F0 B8 64D54F00 E8 B099F0FF 33C9 B2 01 A1 98974E00</pre>	<pre>DD TheCompi.004F4E00 DD TheCompi.004F4EA0 DB 00 DB 00 DB 00 DB 00 DB 00 DD TheCompi.004FD53C PUSH EBP MOV EBP,ESP ADD ESP,-10 MOV EAX,TheCompi.004FD564 CALL TheCompi.00407204 XOR ECX,ECX MOV DL,1 MOV EAX,WORD PTR DS:[4E9798]</pre> <p>Leggermente simile al C++,ma non è preceduto da tutti quei jmp,bensi da codice inutile.</p>

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Linguaggio	Entry Point	Descrizione	
ASM	XOR EAX,EAX PUSH EAX CALL <JMP.&KERNEL32.GetModuleHandleA> MOV DWORD PTR DS:[4031E0],EAX PUSH 0 PUSH oepfinde.00401028	pModule => NULL GetModuleHandleA lParam = NULL lParam = oepfinde.00401028	Tipicamente non è preceduto da nulla,e inizia con un Xor.

► CONCETTI DI BASE SULLE API & BREAKPOINT



Ecco a voi un elenco di API utili.

Per prima cosa,cosa è un API e a cosa ci serve saperlo...Dice wikipedia: **API** è l'acronimo di *Application Program(ming) Interface (Interfaccia di Programmazione di un'Applicazione)*, indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per un determinato compito. È un metodo per ottenere un'astrazione, di solito tra l'hardware e il programmatore, o tra software a basso ed alto livello. Le API permettono di evitare ai programmatori di scrivere tutte le funzioni dal nulla. Le API stesse sono un'astrazione: il software che fornisce una certa API è detto *implementazione dell'API*.

Ottimo,ora,tutti i programmi che andremo ad analizzare (tranne quelli in VB6 e .NET ma poi vedremo anche per quelli) utilizzano le API per fare qualcosa,Questo qualcosa per essere utile a noi reverser,viene inteso come "Prelevo il seriale inserito e lo leggo" oppure "scrivo nel registro le informazioni di registrazione".

Per cercare le API qui sotto elencate (Che non sono tutte !! Ma solo quelle più ricorrenti) è sufficiente dotarsi della commandline di OllyDbg e far precedere al nome dell'API la sigla bpx..circa così:

bpx MessageBoxA

Da ricordare che i nomi delle API sono case sensitive!! (maiuscolo e minuscolo contano)
Intercettarle e settare su di loro un breackpoint è utile per trovare parti essenziali del disassemblato senza impazzire a studiarlo tutto.

Messaggi

MessageBoxA

Semplicemente invia una messagebox all'utente (tipicamente indica l'inserimento di un seriale errato,o corretto)

SendMessageA

Questo metodo non fa altro che inviare un messaggio, opportunamente codificato, ad una finestra

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

specifica. (WM_CLOSE et similia)

Finestre

GetWindowTextA & GetDlgItemTextA

Entrambe utilizzate quando si ha a che fare con una dialogbox che richiede serial e name,in due aree di testo.

CreateWindowExA & ShowWindow

Apronno una nuova finestra di dialogo i dialogo

File

ReadFile

Tipicamente utilizzato per leggere da un file,specie se il file contiene info sulla registrazione del programma.Prestare attenzione che ci sono moltissimi casi in cui un prog accede a dei file,e non farsi trarre in inganno.

WriteFile

Come sopra,serve per scrivere su un file

CreateFileA

Serve per generare un file

GetPrivateProfileStringA

Serve per leggere una riga da un file .ini (quelli di configurazione)

Registro

RegQueryValueExA

Cerca un valore nel registro

RegOpenKeyA

Lo carica nel programma per usarlo (nel senso,legge il contenuto e lo trasforma in variabile o stringa)

Orari

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

GetLocalTimer

Essenziale individuarla nei trial a tempo (tipo,30 giorni) serve per restituire al programma l'orario di sistema.

Drive

GetDriveTypeA

Indispensabile quando si ha a che fare con giochi che chiedono il CD originale,serve infatti a determinare quale delle unità sia ad esempio un lettore CD.

Timer

SetTimer

Tipicamente usato per inizializzare l'evento timer,specie per quei demo in limitazione a minuti.

GetTickCount

Serve per prendere appunto un "tick",un passaggio di un minuto o di un secondo di un timer,sempre utile da individuare nei prog con limitazione a minuti.

► ISTRUZIONI DI LINGUAGGIO ASSEMBLY



L'assembler è stato concepito come linguaggio a basso livello per facilitare leggermente il compito ai programmati, dato che il codice macchina non è umanamente comprensibile.

L'assembly è abbastanza complicato come linguaggio, nonostante non abbia numerose parole chiave. La difficoltà del linguaggio risiede tutta nell'uso obbligatorio dei registri dello stack e di tutte le memorie del computer. Naturalmente un uso improprio di istruzioni è rischioso, e può portare al crash di sistema compilando erroneamente un programma.

Tuttavia, per procedere sulla nostra strada del reversing avere qualche piccola conoscenza di istruzioni assembly non può fare che bene, dato che da qui in avanti ci troveremo davanti soltanto a questo.

Concludo questa piccola introduzione facendovi notare che un compilatore asm lo avete sempre avuto nel vostro PC. Si chiama Debug.exe e si può richiamare direttamente dal prompt dei comandi.

MOV

E' un'istruzione le cui funzioni sono fondamentali nel cracking.

Difatti, molte volte vi ritroverete a dover magari muovere un valore giusto in una locazione semplicemente cambiando un CMP in un MOV (poi vi spiego....non temete).

La sua sintassi è:

MOV [destination],[source]

MOV AX,BX → muove il valore di BX in AX

MOV CX,[0400] → muove i due bytes contenuti all'indirizzo specificato (DS:0400) in CX.

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

```
sub_401B60    proc near                ;  
               push    ebp  
               mov     ebp, esp  
               sub     esp, 0Ch  
               push    offset loc_4010B6  
               mov     eax, large fs:0  
               push    eax  
               mov     large fs:0, esp  
               sub     esp, 0ACh  
sub_401B60    endp
```

CMP

Semplicemente, confronta due valori in memoria, siano essi registri oppure bytes, cambiando di conseguenza il flag relativo.

La sintassi, secondo i casi, può essere:

CMP EAX,01 → confronta AX con 01

CMP EAX,EBX → confronta AX con BX

CMP EAX,[0550] → confronta AX con i 2 bytes a DS:0550

Questa funzione funziona come una sottrazione source-destination, cioè ponendo AX=20 e BX=21, il risultato del CMP EAX,EBX sarà 20-21=-1. Il risultato di questa sottrazione attiverà il flag CARRY (C), attivato quando si sottrae un valore più grande da uno più piccolo; se invece la sottrazione non dà risultato negativo, il flag rimane 0 (disattivato). Dal punto di vista del reversing, le istruzioni CMP sono usate SEMPRE per verificare il vostro input, oppure per far sapere al programma se è registrato o meno.

```
push    edi  
mov     ecx, [edi]  
call    dword ptr [ecx+0A0h]  
cmp    eax, esi  
fncllex  
jge    short loc_401BFE  
push    0A0h  
push    offset dword_401798  
push    edi  
push    eax  
call    ds:_vbaHresultCheckObj
```

JMP

Lo troverete SEMPRE dopo il CMP, nelle varie forme. La più semplice, e ovvia, è la forma JMP <indirizzo>, che naturalmente provoca un salto all'indirizzo desiderato, quali che siano i valori dei flags dati dal CMP.

Per quanto riguarda l'indirizzo, questo può essere nello stesso CS (JMP 0AF4) oppure in un'altro (JMP

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

2000:18A0 oppure JMP DWORD PTR ES:[DI],che vi spedirà alla locazione contenuta in ES:DI. Ma sono solo alcuni esempi).

Questa comunque è l'istruzione che userete più spesso per i vostri scopi. Difatti, se la vostra protezione, ad esempio, controlla una locazione e poi fà un JNZ che riporta il programma all'inserimento della password/serialnumber, basta che sotituiate il jump condizionale con un JMP.

Ecco una tabellina riassuntiva (dei principali)

Istruzione	Definizione	Condizione
JNZ 030A	Jump if not zero	AX diverso da BX
JNE 030A	Jump if not equal	"
JZ 030A	Jump if zero	AX=BX
JE 030A	Jump if equal	= Jnz
JL 030A	Jump if less	AX<BX
JNG 030A	Jump if not greater	
JG 030A	Jump if greater	AX>BX Z=0 oppure S=O

CALL

Richiama una subroutine, e dopo l'esecuzione di questa torna all'indirizzo successivo alla CALL stessa (tramite un RET/RETF).Esempio:

Programma...

Programma...

CALL 68AB → esegue la subroutine a CS:68AB

Codice subroutine...

Codice subroutine...

RET → torna all'istruzione successiva a CALL 68AB

Programma...

Programma...

Una call può essere anche nel formato CALL FAR (come anche il JMP),cioè viene eseguita una subroutine ad un'indirizzo in un altro CS.

Nei vostri primi approcci di cracking, se avete la fortuna di trovare la CALL che salta direttamente alla protezione, potete benissimo togliere quella. Più in là sarà meglio che impariate a districarvi con i jump e identificando e modificando quelli relativi al solo controllo della protezione. Difatti, se per caso quella CALL chiamasse una subroutine che contiene la protezione ma anche istruzioni necessarie al buon funzionamento del programma, eliminandola avrete problemi.

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Per esempio IDA Pro. Quando trova una Call permette con un doppio click di giungere all'indirizzo chiamato, mentre scorrendo col mouse sopra la call esplorarne il contenuto:

The screenshot shows the IDA Pro interface with assembly code. A tooltip is displayed over a `call ds:_vbaHresultCheck0bj` instruction at address `:_401BFE`. The tooltip contains the following information:

```
extrn _vbaHresultCheck0bj:dword ; .text:0
extrn _adj_fdio_md2:dword ; DATA
extrn _vbaObjSet:dword ; DATA XP ; .text:0
extrn rtcMsgBox:dword ; DATA XP ; .text:0
extrn _adj_fdio_md61:dword ; DATA
extrn _adj_fdior_md61:dword ; DATA
extrn _CisIn:dword ; DATA XP
extrn _vbachkstk:dword ; DATA XP
extrn EVENT_SINK_Address:dword ; D
```

The assembly code visible includes:

```
call    ds:_vbaHresultCheck0bj
;_401BFE:
; ext:00401BDE
; \utente\Desktop\A\idc\ida.idc'...
; A\idc\onload.idc
```

INT

Chiama un interrupt (tipo una CALL ma non relativa al programma), con una specifica funzione assegnata da un valore, di solito mosso in AX.

E' utile ad esempio monitorare l'interrupt 13 con il debugger (nel caso sivoglia sproteggere un programma che accede al CD), oppure l'interrupt 16 con funzione 10 (AX=10) nel caso il programma attenda la pressione di un tasto.

NOP

Nop, equivalente a 0x90. Nop significa "No Operation", nessuna operazione. La Cpu salta quell'istruzione finchè non ne trova una operativa

Lo useremo spessissimo in ambito di reversing, dato che talvolta alcune istruzioni sono intutili (le routine di controllo per esempio) e non potendo cancellare le parti di assembler, l'unica altra maniera è l'utilizzo della suddetta istruzione.

... IC74	mov [ebp-5Ch], eax
IC77	call ds:_vbaVarCopy
IC7D	lea eax, [ebp-7Ch]
IC80	lea ecx, [ebp-6Ch]
IC83	nop
IC84	lea edx, [ebp-5Ch]
IC87	push ecx
IC88	push edx
IC89	lea eax, [ebp-4Ch]
IC8C	push esi
IC8D	push eax
IC8E	call ds:rtcMsgBox
IC94	lea ecx, [ebp-7Ch]

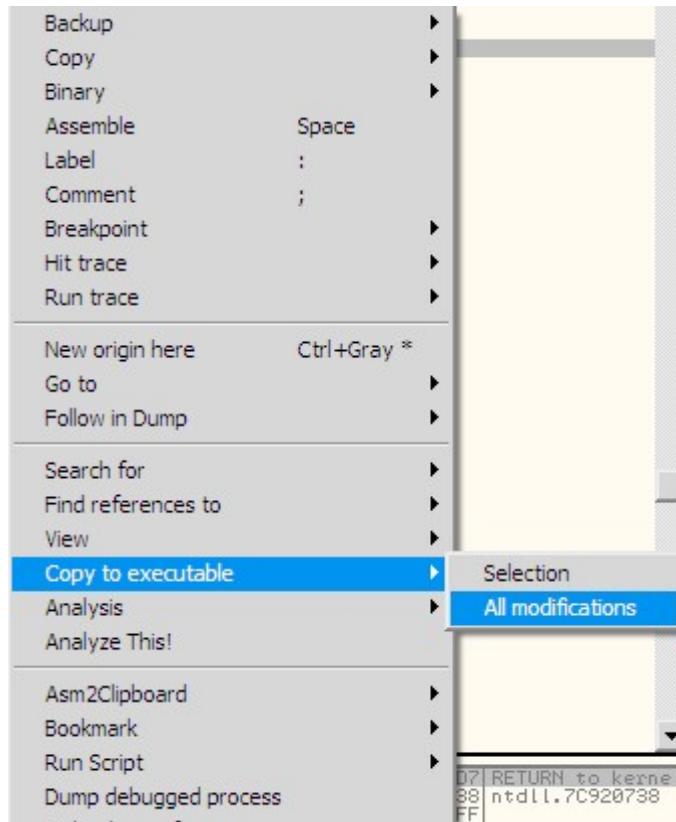
► LE PROTEZIONI



Sempre più spesso le software house hanno deciso di implementare nei loro programmi dei meccanismi di protezione sempre più evoluti, al fine di scoraggiare eventuali reverser. Questo è un elenco con relativi esempi pratici e spiegazione dei principali meccanismi che ci troveremo ad affrontare.

NB: Essendo questa la sezione pratica della guida è bene che sappiate almeno come si salva un programma patchato dentro Olly ,cosa per nulla intuitiva.

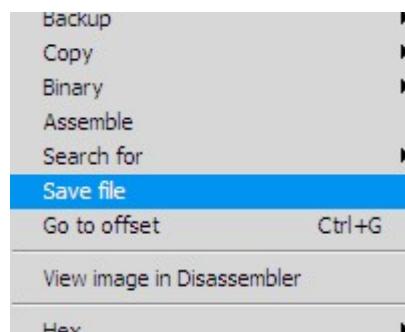
Una volta fatte tutte le modifiche necessarie, premere col tasto destro su una riga di disassemblato, e premere:



Dopodichè premere Copy All



Nella nuova finestra che appare, premere di nuovo tasto destro



e selezionare save file. Dalla finestra di salvataggio dare un nome e ricordarsi di immettere manualmente l'estensione .exe (se no salva in formato testo)

► I serial

Una delle più comuni protezioni di programmi, specie se a tempo limitato, comodi perché vengono calcolati (in genere, ma non sempre) dal programma stesso, sulla base di un nome utente. Infatti i serial vengono generati da un algoritmo interno (da studiare e comprendere per reversarlo) che solo la software house conosce, e dopo una registrazione con pagamento, rilascia un serial valido per l'username fornito.

Esempio pratico n. 1:

CrackMe utilizzato: Un crack me creato da me, lo trovate nello zip allegato a questa guida. [CM1-Serial]

Tools necessari: OllyDbg con il plug in della command bar.
Un blocco note.

Apriamo il crackme e analizziamolo. Solo 2 textbox di input, una richiede un ID numerico, l'altra il seriale corrispondente.

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Se sbagliamo seriale, un messagebox di errore ci ammonisce.

Apriamo ollydbg e carichiamo il programma da crakkare.

Subito settiamo dei breackpoint sulle messagebox, per arrivare a del codice utile

Apriamo la command line (plug in -> command line) e digitiamo (PRECISAMENTE)

```
bpx rtcMsgBox
```

Premiamo Invio.

Ecco che appariranno tutti i breackpoint evidenziati in rosso. Nello specifico 2: la messagebox di errore e quella di successo:

Address	Disassembly	Destination
004011C1	CALL <JMP.&MSVBU60.#100>	MSVBU60.ThunRTMain
00401FD3	CALL DWORD PTR DS:[<&MSVBU60.__vbaObjS	MSVBU60.__vbaObjSet
00401FFA	CALL DWORD PTR DS:[<&MSVBU60.__vbaHres	MSVBU60.__vbaHresUltCheckObj
0040201C	CALL EDI	MSVBU60.__vbaVarMove
00402021	CALL DWORD PTR DS:[<&MSVBU60.__vbaFree	MSVBU60.__vbaFreeObj
00402035	CALL DWORD PTR DS:[<&MSVBU60.__vbaObjS	MSVBU60.__vbaObjSet
00402066	CALL DWORD PTR DS:[<&MSVBU60.__vbaHres	MSVBU60.__vbaHresUltCheckObj
00402087	CALL DWORD PTR DS:[<&MSVBU60.__vbaFree	MSVBU60.__vbaFreeObj
004020B0	CALL DWORD PTR DS:[<&MSVBU60.__vbaVarC	MSVBU60.__vbaVarCat
004020F5	CALL DWORD PTR DS:[<&MSVBU60.__vbaVarD	MSVBU60.__vbaVarDiv
00402100	CALL DWORD PTR DS:[<&MSVBU60.__vbaVarS	MSVBU60.__vbaVarSub
00402115	CALL DWORD PTR DS:[<&MSVBU60.__vbaVarM	MSVBU60.__vbaVarMul
00402130	CALL DWORD PTR DS:[<&MSVBU60.__vbaObjS	MSVBU60.__vbaObjSet
00402157	CALL DWORD PTR DS:[<&MSVBU60.__vbaHres	MSVBU60.__vbaHresUltCheckObj
00402175	CALL DWORD PTR DS:[<&MSVBU60.__vbaVarT	MSVBU60.__vbaVarTstEq
00402180	CALL DWORD PTR DS:[<&MSVBU60.__vbaFree	MSVBU60.__vbaFreeObj
00402189	CALL DWORD PTR DS:[<&MSVBU60.__vbaFree	MSVBU60.__vbaFreeVar
004021D9	CALL DWORD PTR DS:[<&MSVBU60.__vbaVarD	MSVBU60.__vbaVarDup
004021F0	CALL DWORD PTR DS:[<&MSVBU60.#595>]	MSVBU60.rtcMsgBox
00402231	CALL DWORD PTR DS:[<&MSVBU60.__vbaVarD	MSVBU60.__vbaVarDup
00402246	CALL DWORD PTR DS:[<&MSVBU60.#595>]	MSVBU60.rtcMsgBox
0040226C	CALL DWORD PTR DS:[<&MSVBU60.__vbaFree	MSVBU60.__vbaFreeVarList
00402282	CALL DWORD PTR DS:[<&MSVBU60.__vbaFree	MSVBU60.__vbaFreeStr
0040228B	CALL DWORD PTR DS:[<&MSVBU60.__vbaFree	MSVBU60.__vbaFreeObj
004022A9	CALL DWORD PTR DS:[<&MSVBU60.__vbaFree	MSVBU60.__vbaFreeVarList
004022BC	CALL ESI	MSVBU60.__vbaFreeVar

Se clicchiamo sul primo e risaliamo leggermente il codice, ecco che ci troviamo ad un punto interessante:

0040209B	. 52	PUSH EDX
0040209C	. C785 5CFFFFFF	MOV DWORD PTR SS:[EBP-A4],CM1-Seri.0040 UNICODE "000"
004020A6	. C785 54FFFFFF	MOV DWORD PTR SS:[EBP-AC],8
004020B0	. FF15 70104000	CALL DWORD PTR DS:[<&MSVBU60.__vbaVarC MSVBU60.__vbaVarCat
004020B6	. 8BD0	MOV EDX,EAX
004020B8	. 8D40 BC	LEA ECX,DWORD PTR SS:[EBP-44]

Questo codice aggiunge al primo argomento (l'ID, ricordiamoci?) tre zeri.

Premiamo F2 e diventerà una riga rossa. Premiamo F9 e avremo il programma in run. Portatelo in primo piano (è ridotto a icona) e inserite come ID: 1234 e come serial 1234.

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Premete il tasto Controlla, ed ecco che Olly si ferma proprio sulla riga con i tre zeri. Premendo F8 steppiamo un passo per volta...Arrivati all'istruzione __vbaVarCat (2 righe sotto) se osserviamo lo stack (ricordiamo con olly dove si trova?In basso a destra) in quel momento,ecco che appaiono 3 argomenti (che sono 3 variabili) L'ID inserito,il seriale,e la stringa che la routine sta costruendo:

```
0013F3F4 0013F49C Arg1 = 0013F49C
0013F3F8 0013F45C Arg2 = 0013F45C
0013F3FC 0013F4E4 Arg3 = 0013F4E4
0013F400 0013F514
0013F404 0013F5E4
0013F408 00000001
0013F40C 00000009
0013F410 00000001
0013F414 77E475A8 RETURN to GDI32.77E475A8 from ntdll.RtIL
0013F418 77E83020 GDI32.77E83020
0013F41C 00000000
0013F420 00000000
0013F424 77E46B1D GDI32.GetTextExtentPointW
0013F428 003B1144
```

Ottimo.Subito sotto a quel codice troviamo:

```
004020E1 . C785 5CFFFFFF MOV DWORD PTR SS:[EBP-A4],5
004020E5 . C785 4CFFFFFF MOV DWORD PTR SS:[EBP-B4],3
004020F5 . FF15 68104000 CALL DWORD PTR DS:[&MSUBUM60.__vbaVarD] MSUBUM60.__vbaVarDiv
```

Indovinate a cosa serve? Esatto Div sta per “Dividere”. (notiamo che nello stack ci sono ancora i 3 argomenti). Per che numero?Eheh, 5. Sta scritto nella prima riga. *NB: il 5 è in esadecimale,ma ricordiamo che 5 in hex e in dec è identico. Se fosse stato per esempio 22 avremmo trovato la dicitura 16. (ripassate la sezione dei sistemi di numerazione).*

```
00402100 . FF15 00104000 CALL DWORD PTR DS:[&MSUBUM60.__vbaVarS] MSUBUM60.__vbaVarSub
00402106 . 50 PUSH EAX
00402107 . 8D95 44FFFFFF LEA EDX,DWORD PTR SS:[EBP-BC]
0040210D . 8D85 74FFFFFF LEA EAX,DWORD PTR SS:[EBP-8C]
00402113 . 52 PUSH EDX
00402114 . 50 PUSH EAX
00402115 . FF15 58104000 CALL DWORD PTR DS:[&MSUBUM60.__vbaVarMu] MSUBUM60.__vbaVarMul
0040211B 8B00
```

Ehe,questi invece servono per sottrarre e moltiplicare. Rispettivamente per sottrarre alla divisione avuta in precedenza e moltiplicare per 3.

Bene...con queste info possiamo creare un keygen!

Procediamo per passi.Intanto verifichiamo che funzioni. Inseriamo come ID: 1234 e come seriale:

$$(1234000 - (1234000 / 5)) * 3 = 2961600$$

↑

Sono gli zeri aggiunti dalla routine,ricordate?

Se proviamo,avremo un messaggio di conferma ☺

Ora proviamo a portare questa piccola espressione in un linguaggio di scripting facile e funzionante:

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

visual basic script (vbs)

Aprite il blocco note e metteteci questo:

```
'chiede il nome utente per generare il seriale
a = inputbox("Numero utente")
'aggiunge i tre "0" alla fine dell>ID
c = a & "000"
'crea l'espressione,e la mette nella variabile d
'dove c è la variabile vista prima (a & "000")
d = (c - (c / 5)) * 3
'mostra il risultato (ossia la variabile d riempita col numero seriale)
'uso una imputbox perchè il risultato è più facile da copiare che in una messagebox
inputbox"seriale","Risultato",d
```

Salvate il file con estensione .vbs e fate un doppio click sopra. Ecco che genererà seriali per ogni ID inserito!

Le protezioni dei programmi commerciali sono spesso in questa ottica, anche se magari le routine sono leggermente più complicate.

► I Nag

Cosa sono i nag? Semplice, sono delle finestre che danno fastidio. Per esempio quelle che limitano alcune funzioni del programma (*vedi Cracking programmi commerciali, paragrafo Dama delux*) o che avvertono che il programma è da registrare e hanno un conto alla rovescia di un minuto, e via dicendo.

Esempio pratico n. 2:

CrackMe utilizzato: Un crack me creato da me, lo trovate nello zip allegato a questa guida. [CM2-Nag]

Questo crackme riunisce in sé le peggiori caratteristiche dei nag.

- Appaiono a sproposito
- Non si possono chiudere se non dopo un certo tempo.
- Sono ben 2 nag: un messagebox e una finestra.

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Noi avremo 2 obbiettivi: Eliminare il nag a messagebox, e abilitare il bottone del conto alla rovescia fermandolo.

(questo perchè a volte la finestra in cui appare il nag contiene anche impostazioni o cose utili che non sempre va bene eliminare.

Tools: Ollydbg con commandline.

Per prima cosa analizziamo il programma. Aprendolo appare un messagebox,e dopo pochi secondi si apre una finestra di configurazione non chiudibile con un conto alla rovescia,al termine del quale si possono salvare le impostazioni (Tipica limitazione).

Come procediamo?

- Noppiamo il messagebox
- Fermiamo il conto alla rovescia del tasto
- Abilitiamo il tasto a essere clikkato

Apriamo il programma dentro ollydbg,e come nell'esempio precedente,settiamo un breackpoint sulle messagebox.

Ne apparirà solo una,doppio click e vediamo cosa appare:

The screenshot shows the OllyDbg debugger interface. The assembly pane displays the following code snippet:

```
4024C5: E745 A4 481F41 MOV DWORD PTR SS:[EBP-5C],CM2-Nag.004011 UNICODE "Ti ricordo che devi registrare il programma!"  
4024CC: C745 9C 080000 MOV DWORD PTR SS:[EBP-64],8  
4024D3: FF15 90104000 CALL DWORD PTR DS:[<&MSUBUM60._vbaVarD]  
4024D9: 8045 AC LEA EAX,DWORD PTR SS:[EBP-54]  
4024DC: 8040 BC LEA ECX,DWORD PTR SS:[EBP-44]  
4024DF: 50 PUSH EAX  
4024E0: 8055 CC LEA EDX,DWORD PTR SS:[EBP-34]  
4024E3: 51 PUSH ECX  
4024E4: 52 PUSH EDX  
4024E5: 8045 DC LEA EAX,DWORD PTR SS:[EBP-24]  
4024E8: 56 PUSH ESI  
4024E9: 50 PUSH EAX  
4024FA: FF15 24104000 CALL DWORD PTR DS:[<&MSUBUM60.#595>]  
4024FB: 8040 BC LEA ECX,DWORD PTR SS:[EBP-54] MSUBUM60.rtcMsgBox
```

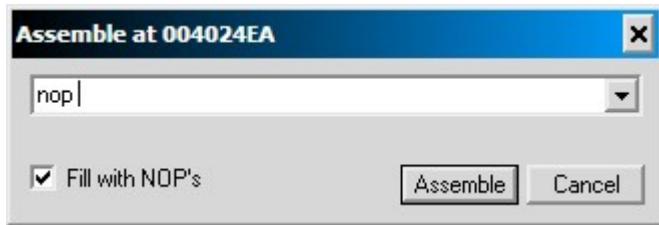
A tooltip is visible over the assembly code at address 4024FA, displaying the function name `MSUBUM60.rtcMsgBox`.

Ottimo. Ricordate quel discorso iniziale sulle istruzioni della CPU ecc? Ecco che arriva la nostra prima istruzione pratica: Nop, equivalente a 0x90. Nop significa “No Operation”,nessuna operazione. La Cpu salta quell'istruzione finchè non ne trova una operativa.

Ecco come procedere dunque. Se sostituiamo alla messagebox un NOP la CPU crede che non ci sia nulla,ed ecco che abbiamo eliminato il nag ☺

Doppio click sulla istruzione e si apre una finestra di modifica. Da lì,cancellare tutta l'istruzione (CALL DWORD PTR DS:[401024]) e sostituirla con “NOP”:

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.



Quindi premere su assemble.

Premiamo F9,e il programma (prima brekka sul vecchio breackpoint msgbox,saltiamolo premendo ancora F9) partì senza mostrare nag ☺

Ora la parte più “difficile”. Procediamo fermando il timer di quel pulsante.(quello del form che si apre con le impostazioni).

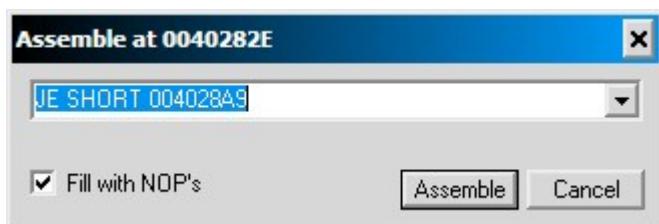
Tuttavia,per non disperderci con inutili tentativi,vediamo cosa succede quando il conto alla rovescia scade...Nulla, il tasto diventa abilitato e assume come testo “salva”. Ottimo punto di partenza.

Andiamo in olly e premiamo tasto destro → Search for → All reference text strings

Fatto ciò,cerchiamo la dicitura “salva” e facciamo doppio click. Ecco dove ci troveremo:

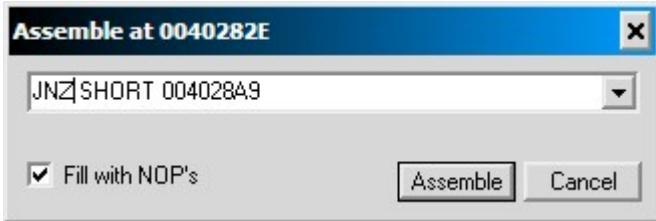
```
00402822  . 0040 00 FF15 AC104000 LEM ECX,DWORD PTR DS:[EIP+24] MSVBVM60.__vbaFree
00402828  . 8B0E CALL DWORD PTR DS:[&MSVBVM60.__vbaFree]
0040282C  . 56 MOV ECX,DWORD PTR DS:[ESI]
0040282D  . 56 PUSH ESI
0040282E  . 66:85DB TEST BX,BX
0040282F  v74 79 JE SHORT CM2-Nag.004028A9
00402830  . FF91 FC020000 CALL DWORD PTR DS:[ECX+2FC]
00402836  . 8055 DC LEA EDX,DWORD PTR SS:[EBP-24]
00402839  . 50 PUSH EAX
0040283A  . 52 PUSH EDX
0040283B  . FFD7 CALL EDI
0040283D  . 8B08 MOV EBX,EAX
0040283E  . 68 00214000 PUSH CM2-Nag.00402108 UNICODE "Salva"
00402844  . 53 PUSH EBX
00402845  . 8B03 MOV EAX,DWORD PTR DS:[EBX]
00402847  . EED9 E4 CALL DWORD PTR DS:[EBP+14]
```

Particolarmente interessante la riga scritta in grigio chiaro JE Short...Ricordate? (*sezione istruzioni assembly utili*) E' un salto condizionale. Probabilmente quello che determina che se il timer arriva a 0 allora il tasto si può premere. Cambiamolo nel suo opposto come abbiamo fatto prima,ossia doppio click sulla istruzione e sostituiamo



Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Con



Premiamo assemble,ed ecco che il programma funziona sembra più nag!

► Trial e scadenze

Ecco una delle protezioni più usate in assoluto. La scadenza (tipicamente di 30 giorni) o la limitazione a minuti.

Esempio pratico n. 3:

CrackMe utilizzato: Un crack me creato da me,lo trovate nello zip allegato a questa guida. [CM3-Trial]

Il crack prevede solo una limitazione di 2 minuti (invece dei 30 giorni),scaduti i quali appare un nag che ci avverte della chiusura del programma.

Tralasciamo direttamente la superata protezione di confrontare la data con una preimpostata,perchè troppo semplice da eludere e totalmente inattuale.

Tool: OllyDbg con plug in della command line

Bene,per prima cosa apriamo il nostro eseguibile dentro OllyDbg,e cerchiamo subito un controllo Timer settando un bel breackpoint.

Nulla di più facile,command line, e digitiamo bp SetTimer.

Fatto ciò runniamo il programma, e guardiamo lo stack quando olly brekka:

```
4 660BBE29 CALL to SetTimer from MSVBVM60.660BBE23
8 001205A2 hWnd = 001205A2 (class='ThunderRT6Timer',parent=000D005C4)
C 001205A2 TimerID = 1205A2 (1181090.)
8 00001770 Timeout = 6000, ms
4 00000000 Timerproc = NULL
0 00000001
```

Ottimo. Ora pensiamo,se annulliamo l'effetto del timer,probabilmente il programma non effettuerà nessun controllo sulla registrazione!

Andiamo alla riga corrispondente,e sostituiamola con un NOP (ricordate?).

Ecco che nessun nag verrà poi mostrato!

(per un metodo diverso,ma interessante consultare il capitolo *I crack commerciali,reversing serio,Pacific Assault*)

Sempre in questo ambito,per renderci conto di come può essere un crack di questo tipo (dato che quello appena preso in esame è davvero facile),ricorrerò anche a un crack commerciale,con un tutorial realizzato da Predator (vedi ringraziamenti).

3D Matrix ScreenSaver

Link: <http://www.nexenteam.net/predator/download/3dmatrix.rar>

Dettagli: Uno dei tanti screen saver basati sul tema Matrix. Molto carino da un punto di vista estetico, si sviluppa in un mondo 3D calcolato in tempo reale... ma gli screen saver una volta non miravano ad usare il minimo di CPU possibile? Perchè non mettiamo su direttamente Doom3 come screen saver? :-D

Tool Usati: OllyDbg110, pistacchi

Obiettivi: Richiede Nome e Seriale altrimenti mostra un fastidioso NAG screen durante l'esecuzione.

Carissimi, eccoci al primo tutorial basato su un programma commerciale, un bel screen saver stile Matrix, ma bando alle ciancie e inziamo subito subito... per prima cosa prendiamo dei pistacchi e mangiamoli! Potrei mangiare pistacchi all'infinito, adoro i pistacchi :-D

Prima di reversarlo, diamo un'occhiata come funziona sceen saver, andiamo su proprietà dello schermo, tab screen saver, scegliamo 3D Matrix ScreenSaver, dall elenco dei screen saver; per vederne le impostazioni premiamo su Imposta mentre per vederlo in azione premiamo Prova... come vedete sulle impostazioni ci dice che è una versione UNREGISTERED e l'opzione Volume è disabilitata, mentre su prova appare un contatore e allo scadere appare un fastidioso nag screen in sovraimpressione.

All'opera! (non dove cantano hehe)

Carichiamo lo screen saver in Olly (si trova in "WINDOWS\system32\3D Matrix ScreenSaver.scr") , premiamo play, per prima cosa ci appare un messaggio.

Premiamo Ok,e osserviamo l'interfaccia di configurazione. Premiamo register e inseriamo nome e seriale a caso. Io metto Predator 111222333.

Premiamo ok ed ecco il malefico messaggio:

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.



Registration name or registration code is incorrect... MA DAAIII! Che cattivi hehehe

bene soffermiamoci a pensare e facciamo le nostre considerazioni: abbiamo tranquillamente almeno 3 punti di partenza ben evidenti:

- il messaggio iniziale
- la maschera di richiesta nick e seriale
- il messaggio di errore registrazione

sono tre punti di facile appiglio non concordate? :-)

E' giusto dire che per arrivare ad un risultato ci sono varie vie.. tutte le strade portano a Roma no?
:-)

chissà quanta gente c'è a Roma :D

Se volete seguitemi, anche se sicuramente siete in grado di trovarne altre.

Iniziamo a fare sul serio:

ricarichiamo il programma premendo il tasto reload in Olly [<<]

apriamo la Command Line e piazziamoci un breack point con

bpx MessageBoxA

premiamo Play, Olly senza esitare brekka nella user32:

77D3E824 > CMP DWORD PTR DS:[77D6D3D0], 0

proseguiamo, premiamo per otto volte F8 e ci appare il MessageBox

Premiamo OK, e siamo qui

77D3E848 RETN 10

premiamo ancora una volta F8 per uscire dalla USER32 e tornare all'exe (che poi è un .scr).
Osserviamo cosa abbiamo davanti

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

```

00418698 E8 733C0000 CALL 3D_Matri.0041C310
0041869D 83C4 08 ADD ESP, 8
004186A0 33C9 XOR ECX, ECX
004186A2 8A0D 207B4A00 MOU CL, BYTE PTR DS:[4A7B20]
004186A8 85C9 TEST ECX, ECX
004186A9 75 29 JNZ SHORT 3D_Matri.004186D5
004186AC 68 24B34200 PUSH 3D_Matri.0042B324
004186B1 8D55 98 LEA EDX, [LOCAL.26]
004186B4 52 PUSH EDX
004186B5 E8 563C0000 CALL 3D_Matri.0041C310
004186B8 83C4 08 ADD ESP, 8
004186BD 6A 00 PUSH 0
004186BF 68 4CB34200 PUSH 3D_Matri.0042B34C
004186C4 68 E8B24200 PUSH 3D_Matri.0042B2E8
004186C9 8B45 08 MOU EAX, [ARG.1]
004186CC 50 PUSH EAX
004186CD FF15 50824200 CALL DWORD PTR DS:[<&USER32.MessageBoxA>]
004186D3 EB 22 JMP SHORT 3D_Matri.004186F7
004186D5 > 68 08B94200 PUSH 3D_Matri.0042B908
004186DA 8D4D 98 LEA ECX, [LOCAL.26]
004186DD 51 PUSH ECX
004186DE E8 2D3C0000 CALL 3D_Matri.0041C310
004186E3 83C4 08 ADD ESP, 8
004186E6 68 247B4A00 PUSH 3D_Matri.004A7B24
004186EB 8D55 98 LEA EDX, [LOCAL.26]
004186EE 52 PUSH EDX
004186EF E8 2C3C0000 CALL 3D_Matri.0041C320
004186F4 83C4 08 ADD ESP, 8

```

noi siamo su 004186D3, ma guardiamoci attorno... quante cosucce interessanti che ci sono! Vediamo appena sopra la preparazione del MessageBox che dice UNREGISTERED, nel quale entriamo dopo aver passato il controllo appena sopra:

004186A8 TEST ECX, ECX ; ntdll.77F49037

004186AA JNZ SHORT 3D_Matri.004186D5

in pratica sembra che il JNZ decida se mandarci su UNREGISTERED oppure saltare a Registered to:

004186D5 PUSH 3D_Matri.0042B908 ; ASCII "Registered to:"

è troppo forte la tentazione di fare una prova su quel salto, piazziamo un break point su **004186AA** ricarichiamo lo screen saver ed eseguiamolo, Olly brekka come previsto su

004186AA JNZ SHORT 3D_Matri.004186D5

sotto leggiamo Jump is NOT taken, andiamo a destra nella finestra dei Registers, facciamo doppio click su Z 1, il valore diventa Z 0, leggiamo nella finestra Jump is taken. e appaiono le ferccine rosse :-)

```

004186B1 85C9 TEST ECX, ECX
004186A9 75 29 JNZ SHORT 3D_Matri.004186D5
004186AC 68 24B34200 PUSH 3D_Matri.0042B324
004186B1 8D55 98 LEA EDX, [LOCAL.26]
004186B4 52 PUSH EDX
004186B5 E8 563C0000 CALL 3D_Matri.0041C310
004186B8 83C4 08 ADD ESP, 8
004186BD 6A 00 PUSH 0
004186BF 68 4CB34200 PUSH 3D_Matri.0042B34C
004186C4 68 E8B24200 PUSH 3D_Matri.0042B2E8
004186C9 8B45 08 MOU EAX, [ARG.1]
004186CC 50 PUSH EAX
004186CD FF15 50824200 CALL DWORD PTR DS:[<&USER32.Mes-
004186D3 > 68 08B94200 PUSH SHORT 3D_Matri.004186F7
004186D5 8D4D 98 LEA ECX, [LOCAL.26]

```

premiamo Play per lasciare proseguire il programma e... il primo message box che dice UNREGISTERED VERSION non appare più, sotto c'è scritto Registration: Registered to: e poi il vuoto...nessun nome (ovviamente), inoltre il volume è ancora disabilitato mmm non va bene, sembrava troppo semplice.

per curiosità premiamo [Registered] spariamo dati a caso oppure premiamo direttamente Cancel e

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

vediamo che ritorna la dicitura UNREGISTERED VERSION. Decisamente non basta :-P
 Passiamo al contrattacco, senza ricaricare il programma, nella Command Line digitiamo bpx MessageBoxA, e premiamo nuovamente [Register], inseriamo Predator e 111222333, premiamo OK e ZAAAKKK Olly brekka,
 proseguiamo con F8 fino all'apparizione del messaggio, premiamo OK
 premiamo ancora una volta F8 per uscire da USER32 e analizziamo la situazione :)

```

00417A72 77 72   JMP SHORT 3D_Matri.00417AE6
00417A74 . 8B85 68FFFFF MOU EAX, [LOCAL_38]
00417A7A > FF2485 717B410 JMP DWORD PTR DS:[EAX*4+417B71]
00417A81 > 6A 00 PUSH 0
00417A83 . 68 3CB34200 PUSH 3D_Matri.0042B33C
00417A88 . 68 98B34200 PUSH 3D_Matri.0042B298
00417A8D . 8B4D 00 MOU ECX, [ARG.1]
00417A90 . 51 PUSH ECX
00417A91 . FF15 50824200 CALL DWORD PTR DS:[<&USER32.MessageBoxA>]
00417A97 . C605 207B4A00 MOU BYTE PTR DS:[4A7B20], 1
00417A9E > EB 46 JMP SHORT 3D_Matri.00417AE6
00417AA0 > 6A 00 PUSH 0
00417AA2 . 68 3CB34200 PUSH 3D_Matri.0042B33C
00417AA7 . 68 B0B34200 PUSH 3D_Matri.0042B3B0
00417AAC . 8B55 08 MOU EDX, [ARG.1]
00417AAF . 52 PUSH EDX
00417AB0 . FF15 50824200 CALL DWORD PTR DS:[<&USER32.MessageBoxA>]
00417AB6 > EB 2E JMP SHORT 3D_Matri.00417AE6
00417AB8 > 6A 00 PUSH 0
00417ABA . 68 3CB34200 PUSH 3D_Matri.0042B33C
00417ABF . 68 B0B24200 PUSH 3D_Matri.0042B2B0
00417AC4 . 8B45 08 MOU EAX, [ARG.1]
00417AC7 . 50 PUSH EAX
00417AC8 . FF15 50824200 CALL DWORD PTR DS:[<&USER32.MessageBoxA>]
00417ACE > EB 16 JMP SHORT 3D_Matri.00417AE6
00417AD0 > 6A 00 PUSH 0
00417AD2 . 68 3CB34200 PUSH 3D_Matri.0042B33C
00417AD7 . 68 B0B24200 PUSH 3D_Matri.0042B2B0
00417ADC . 8B4D 00 MOU ECX, [ARG.1]
00417ADF . 51 PUSH ECX
00417AE0 > FF15 50824200 CALL DWORD PTR DS:[<&USER32.MessageBoxA>]
00417AE6 > 33D2 XOR EDX, EDX
00417AE8 . 8A15 207B4A00 MOU DL, BYTE PTR DS:[4A7B20]
00417AEE . 89FA 01 CMP EDX, 1
00417AF1 > 75 13 JNZ SHORT 3D_Matri.00417B00
00417AF3 . 6A 01 PUSH 1

```

3D_Matri.00417AA0
 Style = MB_OK|MB_APPLMODAL
 Title = "Registration"
 Text = "Thank you for purchase?"
 hOwner = 77F49037
 MessageBoxA

Style = MB_OK|MB_APPLMODAL
 Title = "Registration"
 Text = "Registration name or registration code is invalid."
 hOwner = 00140608
 MessageBoxA

Style = MB_OK|MB_APPLMODAL
 Title = "Registration"
 Text = "Registration name or registration code is invalid."
 hOwner = 00000001
 MessageBoxA

Style = MB_OK|MB_APPLMODAL
 Title = "Registration"
 Text = "Registration name or registration code is invalid."
 hOwner = 77F49037
 MessageBoxA

Result = 1

A quanto pare ci sono 3 possibilità di errore, e una che dice "Thank you for purchase" ghghgh
 guardiamo appena sopra quel codice e osserviamo bene questa riga

00417A7A JMP DWORD PTR DS:[EAX*4+417B71] ; 3D_Matri.00417AA0

in pratica cosa fa? Esegue un salto ad un indirizzo dato da un calcolo, e se noi cambiassimo questo salto in un salto fisso? il codice entrerebbe sempre su "Thank you for purchase" giusto? :-)
 proviamo...

Piazziamo un breack point su 00417A7A, andiamo nella lista dei breack point e cancelliamo quello piazzato prima sull'USER32. Premiamo il tasto [C] per tornare al codice in Olly, premiamo Play per far proseguire l'esecuzione, ripremiamo ancora OK e cosa fa?

Ovvio brekka su 00417A7A. Bene ora vediamo che il salto è attivo, noi lo bipassiamo, come?

Trattandosi di un salto NON CONDIZIONATO non basta cambiare il Flag Z da 0 a 1, dobbiamo Nopparlo oppure con il tasto destro del mouse nella riga 00417A81 scegliere New Origin Here... io scelgo di NOPPARLO e dunque piazzo un bel NOP

Premiamo ancora Play per far proseguire... questa volta il programma cade dentro la nostra trappola:

possiamo notare la correttezza dell'attivazione: Registered To : Predator

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

il volume è attivo, insomma tutto sembra aver avuto successo... ma non è così
Ricarichiamo il programma premendo [<<] se vi appare un messaggio di errore
non preoccupatevi, vi sta dicendo che avete piazzato dei break point che sono corrotti... ma è
dovuto ai cambiamenti che abbiamo fatto prima, non c'è nulla che non va, premiamo ok e non
pensiamoci più :)

Premiamo Play... aimè lo screen saver risulta essere ancora UNREGISTERED, è evidente che il
programma fa ulteriori controlli... sicuramente va a leggere delle chiavi nel registro o qualcosa di
simile :)

Per verificare che quanto detto è vero proviamo a:

- eludere il messaggio di unregistered
- forzare la registrazione
- eseguirlo non in modalità configuraione ma in modalità screen saver attivo

vi dimostrerò per ben 2 volte che anche se facciamo questo lo screen saver risulterà con le
limitazioni e il nag screen.

richiarichiamolo, andiamo nella lista dei break point e cancelliamoli tutti,
piazziamo bpx MessageBoxA

premiamo play

Olly brekka dentro la USER32, premiamo F8 fino a quando non appare il message box,
premiamo OK nel message box che dice UNREGISTERED e torniamo ad Olly,
premiamo F8 fino a quando usciamo dalla USER32 (credo basti premerlo una sola volta)

siamo qui

The screenshot shows the OllyDbg debugger. On the left is the assembly dump window showing code from address 00418698 to 004186F4. The code involves calling functions like 3D_Matrix.0041C910 and 3D_Matrix.004186D5, pushing arguments onto the stack, and performing various memory operations. A message box is displayed on the right, titled "3D Matrix ScreenSaver: \\"Inside the Matrix\\\"", with the message "Welcome to \\"3D Matrix Screensaver\\\" UNREGISTERED Version!". The box has style MB_OK|MB_APPLMODAL, title text, and owner set to 00000001. The message box window handle is highlighted in red. The assembly code includes several jumps (JMP) and calls to external functions.

facciamo un po di back tracing fino a 004186AA cioè qui
JNZ SHORT 3D_Matrix.004186D5

cambiamo questo comando in modo che salti sempre su "Registered", cambiamolo così

JMP 004186D5

Proviamo a fare PLAY, che bello niente NAG screen ma purtroppo li volume è ancora disabilitato.
Per verificare che proprio non basta lanciamo lo screen saver anche in modalità start invece che
configurazione, vi spiego come.

Su Olly dal menu Debug->Arguments

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

come argomento scriviamo /s

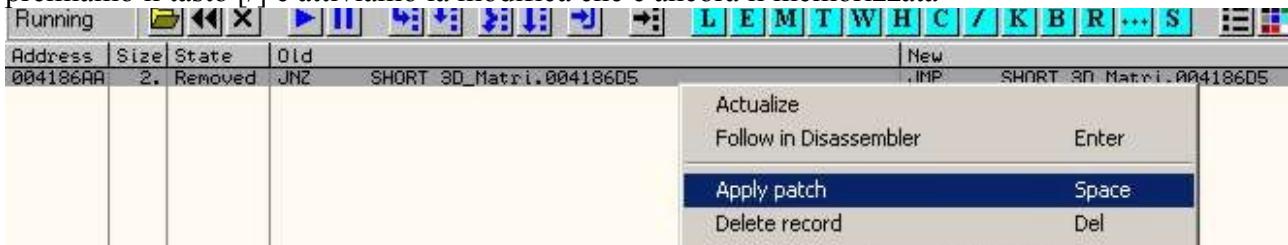
(in pratica è come avviassimo il programma scrivendo "nome programma /s", nel nostro caso è come fosse "3D Matrix ScreenSaver.scr /s"

/s negli screen saver sta per start, mentre /c sta per configurazione.)

premiamo ok e Olly ci dice che questa modifica sarà disponibile solo ricaricando...

premiamo [<<] per ricaricare lo screen saver con i nuovi argomenti di avvio

premiamo il tasto [/] e attiviamo la modifica che è ancora memorizzata



una volta attivata premiamo PLAY per vedere cosa succede

lo screen saver parte... ma abbiamo il contatore, il nag screen allo scadere del contatore e l'assenza di musica.

bleaaaa cambiare quel comando è solo una lamerata!

Ok dai che siamo fregati di portare a segno il nostro obiettivo!

partiamo ad zero, cancelliamo tutti i break point (tasto [B]) e tutte le modifiche (tasto [/]) su argument canelliamo /s e lasciamo vuoto premiamo ok e ricarichiamo il programma [<<]

Premiamo Play

premiamo [Register]

inseriamo Predator e 111222333

su Olly impostiamo bpx MessageBox

torniamo sul programma e premiamo OK, olly brekka, come prima premiamo F8 fino a quando non appare il message box, premiamo OK nel message box e torniamo su Olly

Premiamo F8 una volta per uscire da USER32 e tornare nel sorgente dello screen saver osserviamo bene la nostra situazione:

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

```

00417A72 .> 77 72 00417A74 .> 8B85 68FFFFFF JA SHORT 3D_Matri.00417AE6
00417A74 .> 8B85 68FFFFFF MOV EAX, [LOCAL.38]
00417A74 .> FF2485 717B410 JMP DWORD PTR DS:[EAX*4+417B71]
00417A81 > 6A 00 PUSH 0
00417A83 .> 68 3CB34200 PUSH 3D_Matri.0042B39C
00417A88 .> 68 98824200 PUSH 3D_Matri.0042B298
00417A8D .> 8B4D 08 MOV ECX, [ARG.1]
00417A90 .> 51 PUSH ECX
00417A91 .> FF15 50824200 CALL DWORD PTR DS:[<&USER32.MessageBoxA>]
00417A97 .> C605 207B4A00 MOV BYTE PTR DS:[4A7B20], 1
00417A9E .> EB 46 SHORT 3D_Matri.00417AE6
00417AA0 > 6A 00 PUSH 0
00417AA2 .> 68 3CB34200 PUSH 3D_Matri.0042B39C
00417AA7 .> 68 B0B34200 PUSH 3D_Matri.0042B3B0
00417AAC .> 8B55 08 MOV EDX, [ARG.1]
00417AAF .> 52 PUSH EDX
00417AB0 .> FF15 50824200 CALL DWORD PTR DS:[<&USER32.MessageBoxA>]
00417AB6 .> EB 2E SHORT 3D_Matri.00417AE6
00417ABA > 6A 00 PUSH 0
00417ABA .> 68 3CB34200 PUSH 3D_Matri.0042B39C
00417ABF .> 68 B0B24200 PUSH 3D_Matri.0042B2B0
00417AC4 .> 8B45 08 MOV EAX, [ARG.1]
00417AC7 .> 50 PUSH EAX
00417AC8 .> FF15 50824200 CALL DWORD PTR DS:[<&USER32.MessageBoxA>]
00417ACE .> EB 15 JMP SHORT 3D_Matri.00417B06
00417AD0 > 6A 00 PUSH 0
00417AD2 .> 68 3CB34200 PUSH 3D_Matri.0042B39C
00417AD7 .> 68 B0B24200 PUSH 3D_Matri.0042B2B0
00417ADC .> 8B4D 08 MOV ECX, [ARG.1]
00417ADF .> 51 PUSH ECX
00417AE0 .> FF15 50824200 CALL DWORD PTR DS:[<&USER32.MessageBoxA>]
00417AE6 > 33D2 XOR EDX, EDX
00417AE8 .> 8A15 207B4A00 MOV DL, BYTE PTR DS:[4A7B20]
00417AEE .> 83FA 01 CMP EDX, 1
00417AF1 .> 75 13 JNZ SHORT 3D_Matri.00417B06
00417AF3 .> 6A 01 PUSH 1

```

Style = MB_OK|MB_APPLMODAL
Title = "Registration"
Text = "Thank you for purchase!"
hOwner = 77F49037
MessageBoxA

Style = MB_OK|MB_APPLMODAL
Title = "Registration"
Text = "Registration name or registration code is invalid"
hOwner = 00140608
MessageBoxA

Style = MB_OK|MB_APPLMODAL
Title = "Registration"
Text = "Registration name or registration code is invalid"
hOwner = 00000001
MessageBoxA

Style = MB_OK|MB_APPLMODAL
Title = "Registration"
Text = "Registration name or registration code is invalid"
hOwner = 77F49037
MessageBoxA

Result = 1

vedete cosa c'è un po più sotto di "Thank you for purchase!" ? c'è questo

00417A97 MOV BYTE PTR DS:[4A7B20], 1

è interessantissimo... è come impostasse un valore a 1 che indica che lo screen saver è registrato, mentre se è a 0 è UNregistered. Facciamo una panoramica su questo comando.

spostiamo su e facciamo un click sulla prima riga del codice praticamente su 00401000 premiamo CTRL+F per eseguire la ricerca (in questo modo partiamo dalla prima riga) al suo interno inseriamo **MOV BYTE PTR DS:[4A7B20], 1**

premiamo [FIND]

notiamo che c'è questo codice

```

00416F5C PUSH ECX ; /mette nel buffer BufSize = 0012FFB0
00416F5D 247B4A00 PUSH 3D_Matri.004A7B24 ; mette nel Buffer = 3D_Matri.004A7B24
00416F62 LEA EDX, [LOCAL.7] ; |carica su EDX il contenuto di LOCAL.7
00416F65 PUSH EDX ; |pValueType = 7FFE0304 definisce il tipo
00416F66 PUSH 0 ; |Reserved = NULL
00416F68 PUSH 3D_Matri.0042B7B0 ; |ValueName = "UserName"
00416F6D MOV EAX, [LOCAL.6] ; mette in EAX il contenuto di LOCAL.6
00416F70 PUSH EAX ; |hKey = 0
00416F71 CALL DWORD PTR DS:[<&ADVAPI32.RegQueryValueExA>]; \RegQueryValueExA
mi soffermo a indicarvi la riga qui sopra, in pratica le prime 8 righe preparano la lettura del registro di sistema e la riga a 00416F71 legge (RegQueryValueExA) il contenuto del registro di sistema, in questo caso legge ciò che leggiamo su 00416F68 ovvero lo User Name.
00416F77 CALL 3D_Matri.00416566
00416F7C TEST EAX, EAX testa EAX
00416F7E JNZ SHORT se EAX è diverso da 0 esegui il salto
00416F80 MOV BYTE PTR DS:[4A7B20], 0
00416F87 MOV ECX, [LOCAL.6]
00416F8A PUSH ECX ; |hKey = 0012FFB0
00416F8B CALL DWORD PTR DS:[<&ADVAPI32.RegCloseKey>]; \RegCloseKey
00416F91 CALL 3D_Matri.00416566

```

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

00416F96 TEST EAX, EAX testa EAX
00416F98 JE SHORT se EAX = 0 esegue il salto
00416F9A MOV BYTE PTR DS:[4A7B20], 1 ECCO LA NOSTRA RICERCA DI PRIMA!
00416FA1 MOV ESP, EBP
00416FA3 POP EBP ; kernel32.77E6141A
00416FA4 RETN

Da quello che vediamo si capisce che il programma legge il registro poi fa dei confronti, se il seriale è esatto allora

MOV BYTE PTR DS:[4A7B20], 1
invece se il seriale è sbagliato
MOV BYTE PTR DS:[4A7B20], 0

bene facciamo la nostra mossa!

ricarichiamo il programma e andiamo all'indirizzo 00416F98 (premete CTRL+G e inserite 00416F98)

siamo qui

00416F98 JE SHORT

ora modifichiamo il JE in JNZ così da invertire la sua azione.

Facciamo Play, eheheheh

Il programma non mostra nessun nag screen, è registrato e con il volume attivo! ce l'abbiamo fatta!
NOOO non è vero! :"D

ce l'abbiamo fatta per metà! il programma è crackato per metà tutte le opzioni sono attive ma se lo carichiamo con il l'argomento /s, poi attiviamo la modifica (premere il tasto [/] e attivare la patch) facendo play c'è ancora il contatore e il nag screen

ma tranquilli basta ricercare le restanti stringhe di comando e modificarle come prima :)

ricarichiamo il programma, portiamoci in cima al sorgente

CTRL+F e inseriamo MOV BYTE PTR DS:[4A7B20], 1

premiamo FIND

modifichiamo

00416F98 JE SHORT 00416FA1

in

00416F98 JNZ SHORT 00416FA1

ora continuiamo la ricerca, premiamo CTRL+L che sta per cerca successivo

e siamo su

00417A97 MOV BYTE PTR DS:[4A7B20], 1

qui non ci interessa intervenire perchè è quello di prima che viene eseguito solo se inseriamo il seriale giusto a mano, possiamo premere ancora CTRL+L

00417CAF TEST EAX, EAX

00417CB1 JE SHORT 00417CBA

00417CB3 MOV BYTE PTR DS:[4A7B20], 1

cambiamo

00417CB1 JE SHORT 00417CBA

in

00417CB1 JNZ SHORT 00417CBA

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

ok ragazzi è fatta è tutto crakkato potete provare lo screen saver completo in tutte le sue parti :)

riassumo le patch

Patches

Address	Size	State	Old	New	Comment
00416F98	2.	Active	JE	SHORT 3D_Matri.00416FA1	JNZ SHORT 3D_Matri.00416FA1
00417CB1	2.	Active	JE	SHORT 3D_Matri.00417CBA	JNZ SHORT 3D_Matri.00417CBA

Considerazioni:

per trovare quei punti era possibile anche eseguire dei break point sulla lettura del registro di sistema cioè su RegQueryValueExA, ma avremo altre occasioni di provare anche questo tipo di approccio :-)

Ah! non dimenticate di chiudere bene i pistacchi rimasti :D

► Packing & Unpacking

Eccoci alla protezione più importante che spesso è anche associata a quelle appena analizzate. Si tratta dei packer di eseguibili (vedi paragrafo *Concetti base sul reversing, ExePacker*).

Ci sono tantissimi packer al mondo, tutti con un modo particolare di procedere.

Tra i più intelligenti (e costosi) ricordiamo Execryptor e Armadillo per le cui protezioni bisogna lavorare parecchio (ed essendo reversing avanzato, vi rimando alle ottime guide di Evolution).

In questo capitolo analizzeremo in particolare un unpacking classico di UPX nel metodo automatico e manuale fino all'unpacking di un crackme apposito realizzato da predator (con un sistema intelligente di protezione).

► Normal unpacking

Esempio pratico n. 5:

CrackMe utilizzato: Un crack me creato da me, lo trovate nello zip allegato a questa guida. [CM5-UPX1]

Questo è un semplice eseguibile che è solo packato con UPX.

Tool: UpxShell (o upx stesso), RDGPacker Detector (o il mio :D) e OllyDbg

Bene, prendiamo RDG e apriamo l'eseguibile da analizzare, ecco quello che appare:

[Http://crevt.altervista.org](http://crevt.altervista.org) <http://ctrlaltcanccorp.altervista.org>

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.



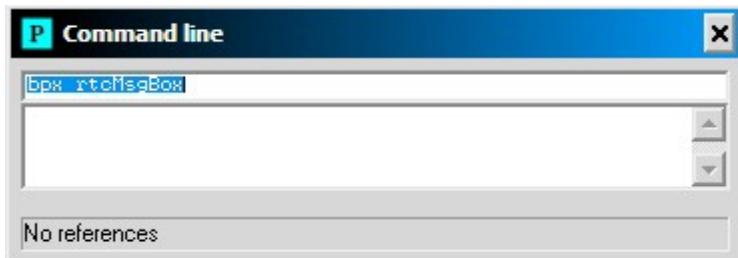
Apriamolo dentro Olly e vediamo cosa succede. Intanto l'EP (*capitolo Basi del reversing,EP*) sembra strano...

```
0040577C 00 00 00 00  
0040577D 00 00 00 00  
0040577E 00 00 00 00  
0040577F 00 00 00 00  
00405780 $ 60 PUSHAD  
00405781 . BE 00504000 MOV ESI,CM4-Upx1.00405000  
00405786 . 80BE 00C0FFFF LEA EDI,DWORD PTR DS:[ESI+FFFFC000]  
0040578C . 57 PUSH EDI  
0040578D . 83CD FF OR EBP,FFFFFFF  
00405790 .~EB 10 JMP SHORT CM4-Upx1.004057A2  
00405792 . 90 NOP  
00405793 . 90 NOP  
00405794 > 90 NOP  
00405795 . 90 NOP  
00405796 . 90 NOP  
00405797 . 90 NOP  
00405798 > 8A06 MOV AL,BYTE PTR DS:[ESI]  
00405799 . 46-- INC ESI
```

Ma va bene. Ora tentiamo di intercettare la messagebox che appare premendo il tasto al centro del crackme.

Commandline, bpx rtcMsgBox

ecco il risultato



No reference. Nessun riferimento. E così è infatti, nessun breakpoint appare. Ecco che un

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

eseguibile packato presenta questo grave problema. Per risolvere,cerchiamo di scomprimerlo.

Prendiamo UpxShell (o UPX stesso) e vediamo come procedere:

Bè è semplicissimo,lo apriamo dentro UPX e premiamo “Decompress” :D

Semplice vero? Era solo per farvi entrare nell'ottica,ma dovete sapere che tutti gli altri programmi di compressione non permettono anche l'unpack.

Infatti spesso troverete molti unpacker già pronti ma non sempre funzioneranno. A quel punto bisogna arrangiarsi a mano. Lo stesso olly mette a disposizione un comodo ssistema di script (previa installazione di OllyScript) che spesso sono voltati all'individuazione dell'EP

Il prossimo crackme infatti ha una difficoltà in più.

►Manual unpacking

Esempio pratico n. 6:

CrackMe utilizzato: Un crack me creato da me,lo trovate nello zip allegato a questa guida. [CM6-UPX2]

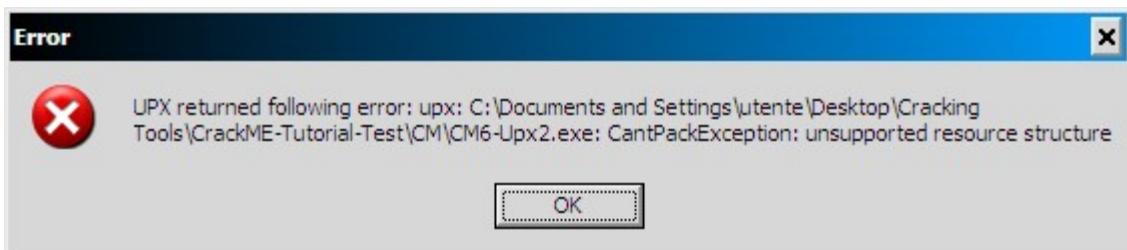
Questo crackme è stato sia compresso con UPX,ma anche trattato con un programmino di mia invenzione,Upx Antiunpack:

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.



Tool: Upx & OllyDbg

Ecco che se proviamo ad unpakkare il nostro crackme con UPX accade questo:



Come procediamo allora? Apriamolo dentro Olly e vediamo.

Come si fa ad unpakkare manualmente? Facile,bisogna risalire all'OEP. (vedi sezione *Concetti base sul reversing,OEP*). Come ottenerlo? Bè,un ottimo reverser ha realizzato un programmino che la maggio parte delle volte funziona (sezione Tool del reverser) ,ma è poco ortodosso.

Facciamo invece manualmente. Rechiamoci alla fine del codice assembly utile , e settiamo un breackpoint:

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

```
004058B8 .^74 07 JE SHORT CM6-Upx2.004058C1
004058BA . 8903 MOV DWORD PTR DS:[EBX],EAX
004058BC . 83C3 04 ADD EBX,4
004058BF .^EB D8 JMP SHORT CM6-Upx2.00405899
004058C1 > FF96 44590000 CALL DWORD PTR DS:[ESI+5944]
004058C7 > 8BAE 40590000 MOV EBP,DWORD PTR DS:[ESI+5940]
004058CD . 8D8E 00F0FFFF LEA EDI,DWORD PTR DS:[ESI-1000]
004058D3 . BB 00100000 MOV EBX,1000
004058D8 . 50 PUSH EAX
004058D9 . 54 PUSH ESP
004058DA . 6A 04 PUSH 4
004058DC . 53 PUSH EBX
004058DD . 57 PUSH EDI
004058DE . FF05 CALL EBP
004058E0 . 8D87 D7010000 LEA EAX,DWORD PTR DS:[EDI+1D7]
004058E6 . 8020 7F AND BYTE PTR DS:[EAX],?F
004058E9 . 8060 28 7F AND BYTE PTR DS:[EAX+28],?F
004058ED . 58 POP EAX
004058EE . 50 PUSH EAX
004058EF . 54 PUSH ESP
004058F0 . 50 PUSH EAX
004058F1 . 53 PUSH EBX
004058F2 . 57 PUSH EDI
004058F3 . FF05 CALL EBP
004058F5 . 58 POP EAX
004058F6 . 61 POPAD
004058F7 . 8D4424 80 LEA EAX,DWORD PTR SS:[ESP-80]
004058FB > 6A 00 PUSH 0
004058FD . 39C4 CMP ESP,EAX
004058FF .^75 FA JNZ SHORT CM6-Upx2.004058FB
00405901 . 83EC 80 SUB ESP,-80
00405904 .-E9 43B8FFFF JMP CM6-Upx2.0040114C
00405909 00 DB 00
0040590A 00 DB 00
0040590B 00 DB 00
0040590C 00 DB 00
0040590D 00 DB 00
0040590E 00 DB 00
0040590F 00 DB 00
00405910 00 DB 00
00405911 00 DB 00
00405912 00 DB 00
00405913 00 DB 00
00405914 00 DB 00
00405915 00 DB 00
00405916 00 DB 00
00405917 00 DB 00
```

Premiamo F9 e mandiamo in Run. Subito Olly bloccherà proprio a quell'indirizzo (fin lì infatti è solo codice del packer, che sta scompattando il crackme in memoria).

Al breack, premiamo F7. Eccoci, quello è L'OEP.

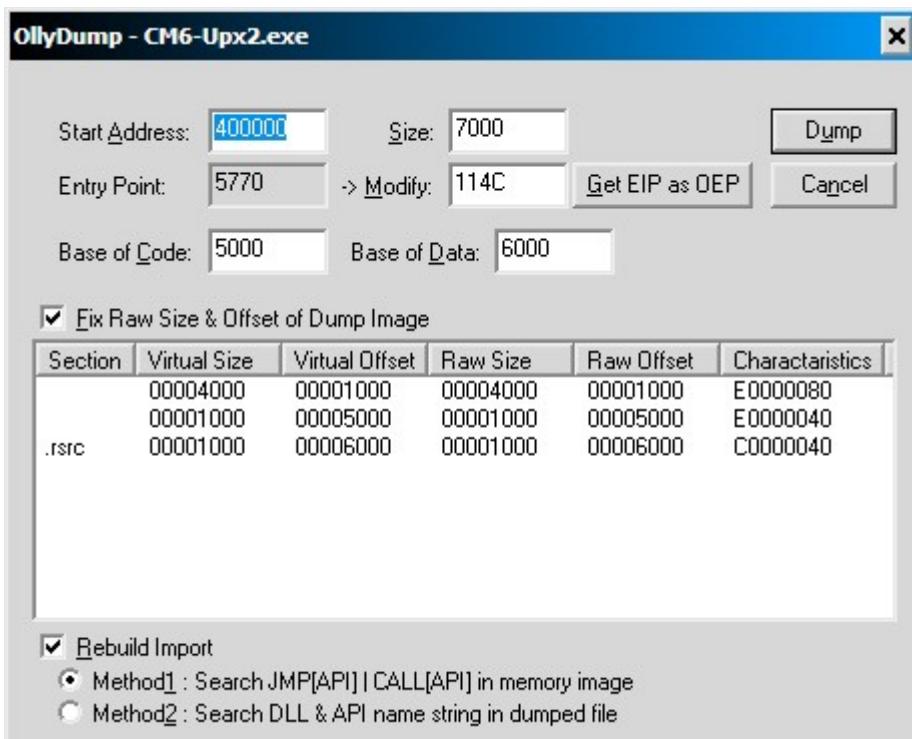
Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

004010F0	-FF25 6C104000	JMP DWORD PTR DS:[40106C]	MSUBUM60._CIatan
004010F6	-FF25 00104000	JMP DWORD PTR DS:[401000]	MSUBUM60._Cicos
004010FC	-FF25 78104000	JMP DWORD PTR DS:[401078]	MSUBUM60._Clexp
00401102	-FF25 50104000	JMP DWORD PTR DS:[401050]	MSUBUM60._CIlog
00401108	-FF25 24104000	JMP DWORD PTR DS:[401024]	MSUBUM60._CIsin
0040110E	-FF25 38104000	JMP DWORD PTR DS:[401038]	MSUBUM60._CIsqrt
00401114	-FF25 74104000	JMP DWORD PTR DS:[401074]	MSUBUM60._CItan
0040111A	-FF25 70104000	JMP DWORD PTR DS:[401070]	MSUBUM60._allmul
00401120	-FF25 08104000	JMP DWORD PTR DS:[401008]	MSUBUM60._vbaFreeVarList
00401126	-FF25 68104000	JMP DWORD PTR DS:[401068]	MSUBUM60._vbaVarDup
0040112C	-FF25 18104000	JMP DWORD PTR DS:[401018]	MSUBUM60._rtcMsgBox
00401132	-FF25 3C104000	JMP DWORD PTR DS:[40103C]	MSUBUM60.EVENT_SINK_QueryInterf
00401138	-FF25 2C104000	JMP DWORD PTR DS:[40102C]	MSUBUM60.EVENT_SINK_AddRef
0040113E	-FF25 34104000	JMP DWORD PTR DS:[401034]	MSUBUM60.EVENT_SINK_Release
00401144	-FF25 64104000	JMP DWORD PTR DS:[401064]	MSUBUM60.ThunRTMain
0040114A	0000	ADD BYTE PTR DS:[EAX], AL	
0040114C	68 40134000	PUSH CM6-Upx2.00401144	JMP to MSUBUM60.ThunRTMain
00401151	E8 EFFFFFFF	CALL CM6-Upx2.00401144	
00401156	0000	ADD BYTE PTR DS:[EAX], AL	
00401158	0000	ADD BYTE PTR DS:[EAX], AL	
0040115A	0000	ADD BYTE PTR DS:[EAX], AL	
0040115C	3000	XOR BYTE PTR DS:[EAX], AL	
0040115E	0000	ADD BYTE PTR DS:[EAX], AL	
00401160	40	INC EAX	
00401161	0000	ADD BYTE PTR DS:[EAX], AL	
00401163	0000	ADD BYTE PTR DS:[EAX], AL	
00401165	0000	ADD BYTE PTR DS:[EAX], AL	
00401167	0017	ADD BYTE PTR DS:[EDI], DL	
00401169	804CEA A5 93	OR BYTE PTR DS:[EDX+EBP*8-5B], 93	
0040116E	D24A B0	ROR BYTE PTR DS:[EDX-501], CL	
00401171	2241 89	AND AL, BYTF PTR DS:[ECX-771]	

Lo riconosciamo per 2 motivi. Essendo questo un crackme in visual basic (ricordiamo l'importanza del linguaggio? Capitolo Fondamenti di reversing) , sappiamo che subito prima dell'EP ci sono i riferimenti alle varie API e subito dopo una chiamata alla ThunRTmain.

Che fare ora? Esportiamo un Dump funzionante! Fate tasto destro -> Dump debugged process (dovete avere il pluging “OllyDump”)

Ecco cosa appare (non toccate nulla,togliete però lo spunto su rebuild import!)



Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

E premendo su “dump” verrà esportato l'exe unpakkato! Ma non è completo,divete ricostruire la IAT. Per sapere come fare,recatevi nella sezione trucchi reversing che segue qualche capitolo più avanti!

Provare per credere.



► Advance unpacking

Esempio pratico n. 7:

CrackMe utilizzato: Un crack me creato da Predator,lo trovate nello zip allegato a questa guida.
[CM7-Unpack]

Questo unpack me è leggermente più complicato degli altri,perchè usa un metodo tutto suo di proteggere.

Tools: OllyDbg,PETools,RDG Packer Detector,VBDecompiler,PEAnalyzer (by Ctrl_alt_canc :P)

Per prima cosa,apriamo il nostro exe dentro RDGPacker Detector.
Il nostro packer detector non segnala nulla...ma l'exe è inequivocabilmente pakkato,è sufficiente

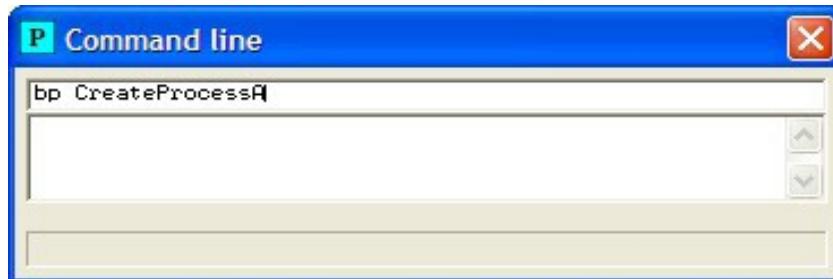
Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

tentare di cambiare una caption,o apportare qualche altra modifica...
Apriamolo allora dentro il VBDecompiler,e rechiamoci alla sezione API...

```
'VA: 4026A0
Private Declare Function CloseHandle Lib "kernel32" Alias "CloseHandle" (ByVal hObject As Long) As I
'VA: 4026C
Private Declare Function VirtualProtectEx Lib "kernel32" Alias "VirtualProtectEx" (ByVal hProcess As
'VA: 402610
Private Declare Sub VirtualAllocEx Lib "kernel32"()
'VA: 4025C8
Private Declare Function ReadProcessMemory Lib "kernel32" Alias "ReadProcessMemory" (ByVal hProcess
'VA: 40257C
Private Declare Function WriteProcessMemory Lib "kernel32" Alias "WriteProcessMemory" (ByVal hProces
'VA: 402530
Private Declare Sub ZwUnmapViewOfSection Lib "ntdll.dll"()
'VA: 4024D0
Private Declare Function CreateProcess Lib "kernel32" Alias "CreateProcessA" (ByVal lpApplicationName
'VA: 402488
Private Declare Function ResumeThread Lib "kernel32" Alias "ResumeThread" (ByVal hThread As Long) As
'VA: 402440
Private Declare Function SuspendThread Lib "kernel32" Alias "SuspendThread" (ByVal hThread As Long)
'VA: 4023F8
Private Declare Function SetThreadContext Lib "kernel32" Alias "SetThreadContext" (ByVal hThread As
'VA: 4023AC
Private Declare Function GetThreadContext Lib "kernel32" Alias "GetThreadContext" (ByVal hThread As
```

Alcune API sono davvero interessanti,tra cui CreateProcessA,ResumeThread e WriteProcessMemory....diremo noi,cosa se ne fa?Semplice,se guardiamo nella documentazione delle API,scopriamo che serve proprio ad aprire un nuovo processo in memoria!

Per prima cosa,carichiamo in olly il programma,e settiamo un bel breakpoint sulla suddetta API:



Ottimo,avviamo il processo da olly,e quando brekka guardiamo lo stack:

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

0013F628	00403C00	CALL to CreateProcessA from TestLock.00403C08
0013F62C	00000000	ModuleFileName = NULL
0013F630	00160894	CommandLine = "TestLockato.exe"
0013F634	00000000	pProcessSecurity = NULL
0013F638	00000000	pThreadSecurity = NULL
0013F63C	00000000	InheritHandles = FALSE
0013F640	00000004	CreationFlags = CREATE_SUSPENDED
0013F644	00000000	pEnvironment = NULL
0013F648	00000000	CurrentDir = NULL
0013F64C	0013F698	pStartupInfo = 0013F698
0013F650	0013F7A4	pProcessInfo = 0013F7A4
0013F654	00406024	TestLock.00406024
0013F658	00150398	

Ecco la chiamata all'API. Analizziamo di cosa è composta... interessante la Creationflag CREATE_SUSPENDED, la quale infatti prevede la chiusura solo dopo un invio di un comando ResumeThread ma guarda guarda! Nelle API del programma troneggia pure quella chiamata! Ma passiamo oltre... abbiamo ora un piccolo problema, il processo in realtà sembra che sia riaperto solo quello principale, ma con il comando (sempre presente tra le API trovate) WriteProcessMemory, si può scrivere un programma direttamente nella memoria del processo... ora, proviamo a brekkare proprio lì: (sempre nello stack)

0013F63C	00403D8C	CALL to WriteProcessMemory from TestLock.00403D
0013F640	00000084	hProcess = 00000084 (window)
0013F644	00400000	Address = 400000
0013F648	0017D5B8	Buffer = 0017D5B8
0013F64C	00001000	BytesToWrite = 1000 (4096.)
0013F650	0013F9C8	pBytesWritten = 0013F9C8
0013F654	00406024	TestLock.00406024
0013F658	00150398	
0013F65C	00000120	
0013F660	0015EA70	
0013F664	00000008	
0013F668	00000000	
0013F66C	00000000	

Cosa notiamo? Buffer 0017D5B8 (cambia da PC a PC, quindi adattatevi al vostro)... proviamo a vedere cosa va a scrivere?



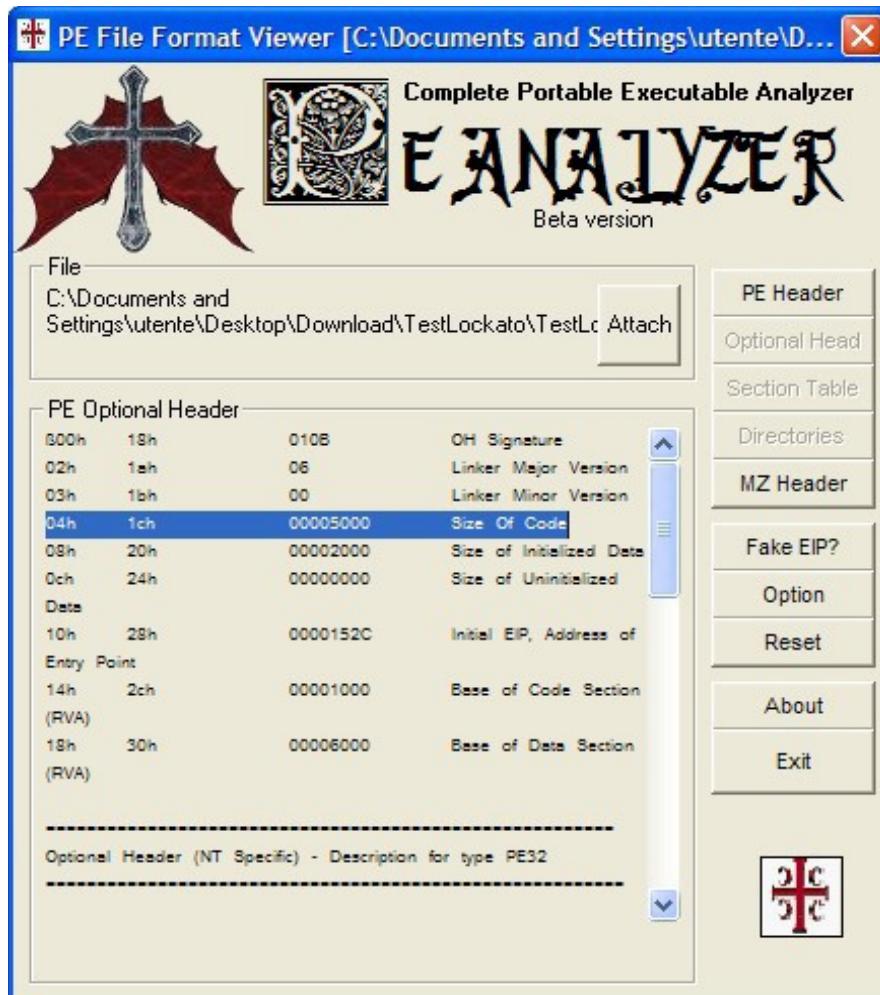
E rechiamoci nella sezione HEX:

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Address	Hex dump	ASCII
0017D5B8	40 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZÈ.♦...◆...
0017D5C8	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00	@.....@.....
0017D5D8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0017D5E8	00 00 00 00 00 00 00 00 00 00 00 00 B8 00 00 00@...
0017D5F8	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	AV A.1.=@0L=†Th
0017D608	69 73 20 70 72 6F 67 72 61 60 20 63 61 6E 6E 6F	is program canno
0017D618	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
0017D628	60 6F 64 65 2E 00 00 00 0A 24 00 00 00 00 00 00	mode...\$.
0017D638	8B 23 C4 DB CF 42 AA 88 CF 42 AA 88 CF 42 AA 88	I#—■DB—ëDB—ëDB—ë
0017D648	4C 5E A4 88 CE 42 AA 88 80 60 A3 88 CD 42 AA 88	L^kejB—ëC'üë=B—ë
0017D658	F9 64 A7 88 CE 42 AA 88 52 69 63 68 CF 42 AA 88	—dœëjB—ëRichDB—ë
0017D668	00 00 00 00 00 00 50 45 00 00 4C 01 03 00PE..L0*.
0017D678	03 01 F5 45 00 00 00 00 00 00 E0 00 0F 01 ♦0SE.....0..*0	
0017D688	0B 01 06 00 00 20 00 00 00 30 00 00 00 00 00 00	00♦.....0.....
0017D698	A0 11 00 00 10 00 00 00 30 00 00 00 40 00	à.....►.....0.....@.
0017D6A8	00 10 00 00 00 10 00 00 04 00 00 00 01 00 00 00	►.....►.....♦.....0....
0017D6B8	24 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

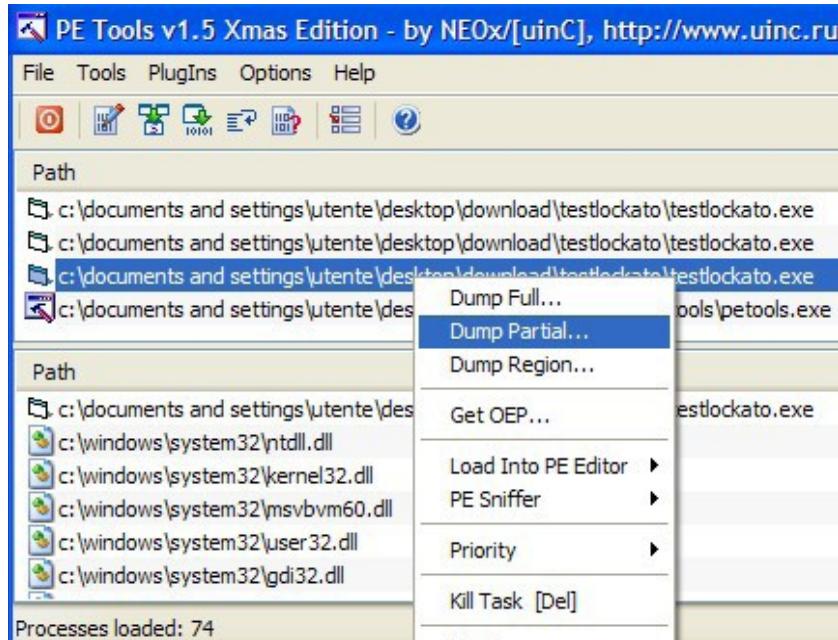
notiamo nulla di strano?? MZ all'inizio...This program cannot be run in DOS mode...ma..ma..è un eseguibile! Se continuiamo a steppare con F9 per ancora 3 o 4 volte,l'exe viene interamente scritto in memoria.

Ora,apriamo il mitico programma PEanalyzer (eheh,il mio) e usiamo la funziona Attach per analizzare il processo del CrackME e rechiamoci negli optional Header:

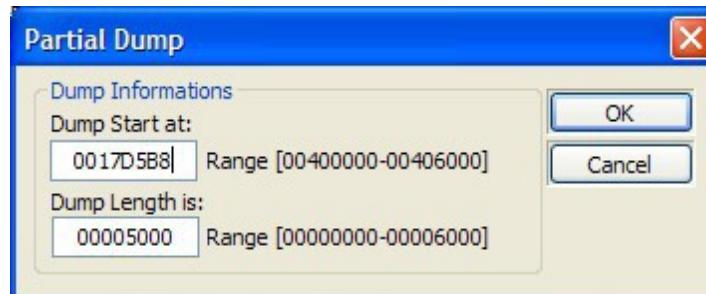


Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Annotiamo il Size Of Code,e proseguiamo! Ora,dobbiamo dumpare,da quell'indirizzo prima individuato (il primo breakpoint su WriteProcessMemory) che nel mio caso è 0017D5B8.
Apriamo quindi PETools,e



Ora,inseriamo i dati appena scoperti,ossia l'indirizzo da cui partire a dumpare,e la dimensione dell'exe (in modo tale che capisca quando fermare il dump)



Ecco fatto! Adesso ricostruiamo un exe dal dump,come meglio credete,usando quello che volete (LordPe...) e avete l'exe unpackato.

Questa è una protezione già più avanzata di un semplice UPX o ASP pack.

► Accenno ad altri metodi di packing

Naturalmente,i metodi di packing con relative protezioni sono molteplici.

Tra i più diffusi e importanti ricordiamo tuttavia:

Execryptor il quale ha come punto di forza la virtualizzazione dell'entry point e di alcune parti di codice,sostituite con garbage code. Per cui alcuni pezzi del programma sono veramente confusi da studiare e da ricostruire.

Distrugge una buona parte delle import,e utilizza una interessante tecnica di debug basata su alcuni thread aperti che monitorano costantemente la presenza di un debugger.

Armadillo Utilizza il metodo dell'hardware fingerprint,ossia un codice univoco per ogni PC cui è associato un seriale,al primo avvio del programma viene richiesto. Se inserito correttamente i dati vengono messi nel registro,e il programma funzionerà per sempre. Utilizza anche delle false Import,per confondere il reverser, e ne sostituisce alcune con altre,create in proprio dai suoi programmatori.

Ottimi tutorial su questi due packer li trovate sul sito di Evolution (vedi ringraziamenti)

► ACCESSO AI FILE

Talvolta invece i programmi basano la loro avvenuta registrazione tramite chiavi di registro,o dei file esterni al programma (spesso codificati o ben nascosti). Ecco un comodo tutorial per prenderci la mano.

Esempio pratico n. 8:

CrackMe utilizzato: Un crack me creato da me,lo trovate nello zip allegato a questa guida. [CM8-File]

Tool: OllyDbg, Filemon.

Ricarichiamo il file dentro olly,e diamo una occhiata. Dopo una prima analisi notiamo che non è possibile fare molto per crakkarlo (o forse sì,ma a noi interessa un altro metodo).

Prendiamo il nostro fedele FileMon,e impostiamo come filtro il nome del nostro crakme.(“CM8-File”)

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Premiamo ok,e avviamo il nostro programma.

Tra le moltissime righe notiamo una cosa molto interessante :

001425	test.exe:5768	QUERY INFORMATION	C:\Documents and Settings\utente\Desktop\Cracking Tools\CrackM...	NOT FOUND
000950	test.exe:5768	QUERY INFORMATION	C:\WINDOWS\system32\SXS.DLL	SUCCESS
003017	test.exe:5768	OPEN	C:\WINDOWS\system32\SXS.DLL	SUCCESS
000922	test.exe:5768	CLOSE	C:\WINDOWS\system32\SXS.DLL	SUCCESS
002235	test.exe:5768	QUERY INFORMATION	C:\WINDOWS\system32\MSVBVM60.DLL	SUCCESS
017209	test.exe:5768	OPEN	C:\tmp.tmp	SUCCESS
001062	test.exe:5768	READ	C:\tmp.tmp	SUCCESS
000810	test.exe:5768	READ	C:\tmp.tmp	END OF FILE
001062	test.exe:5768	CLOSE	C:\tmp.tmp	SUCCESS
001341	explorer.exe:3604	QUERY INFORMATION	C:\Documents and Settings\utente\Desktop\Cracking Tools\CrackM...	SUCCESS
000000		QUERY INFORMATION	C:\Documents and Settings\utente\Desktop\Cracking Tools\CrackM...	SUCCESS

Eheh,quel <C:\tmp.tmp> ci fa intuire la presenza di un file esterno di riferimento. Doppio click,e apriamolo.

Il contenuto è abbastanza banale:

UNREGISTERVERSION-

Sostituiamo l'intera stringa con il nostro nick (Ctrl_alt_canc) e riavviamo il programma...



Primo ostacolo superato! (infatti anche se premiamo check,non mostrerà alcuna messagebox)

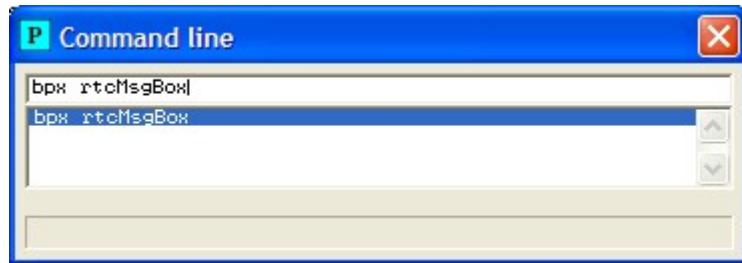
Ora manca il secondo obiettivo,fare in modo che se si preme il tasto register,non appaia il form,ma una messagebox (non meglio specificata)

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.



PASSO 2

Bene bene, ora andiamo nel menù plug in e selezioniamo il plug della command line, e impostiamo un breakpoint sulle messagebox



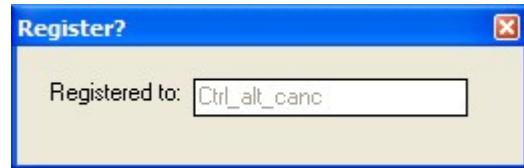
Ottimo, abbiamo 3 riferimenti! Uno è quello che si apre quando premiamo il "?", l'altra quando premiamo il check e non siamo registrati, e l'ultima deve essere la msgbox da fare apparire!

Andando per esclusione, arriviamo a capire che questa è quella a cui mi riferisco:

00403489	. 8945 CC	MOU DWORD PTR SS:[EBP-34],EAX		
0040348C	. 894D C4	MOU DWORD PTR SS:[EBP-3C],ECX		
0040348F	. C745 9C D02941	MOV DWORD PTR SS:[EBP-64],test.00402900	UNICODE "Ok!"	
00403496	. C745 94 000000	MOV DWORD PTR SS:[EBP-6C],8		
0040349D	. 8055 94	LEA EDX,DWORD PTR SS:[EBP-6C]		
004034A0	. 804D D4	LEA ECX,DWORD PTR SS:[EBP-2C]		
004034A3	. FF15 AC104000	CALL DWORD PTR DS:[&MSUBUM60.__vbaVarD]	MSUBUM60.__vbaVarDup	
004034A9	. 8045 A4	LEA EAX,DWORD PTR SS:[EBP-5C]		
004034AC	. 50	PUSH EAX		
004034AD	. 804D B4	LEA ECX,DWORD PTR SS:[EBP-4C]		
004034B0	. 51	PUSH ECX		
004034B1	. 8055 C4	LEA EDX,DWORD PTR SS:[EBP-3C]		
004034B4	. 52	PUSH EDX		
004034B5	. 6A 00	PUSH 0		
004034B7	. 8045 D4	LEA EAX,DWORD PTR SS:[EBP-2C]		
004034B9	. 50	PUSH EAX		
004034BB	. FF15 2C104000	CALL DWORD PTR DS:[&MSUBUM60.#595]	MSUBUM60.rtcMsgBox	
004034C1	. 804D A4	LEA ECX,DWORD PTR SS:[EBP-5C]		

Benissimo, ora dobbiamo intercettare la chiamata al form che generalmente appare alla pressione del tasto register ()

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.



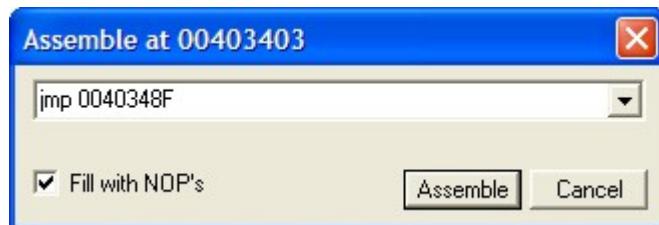
e fare jumpare il programma sull'indirizzo della msgbox...

```
0040348F . C745 9C D02940>MOV DWORD PTR SS:[EBP-64],test.004029D0 ; UNICODE "Ok!"
```

per fare ciò è necessaria sempre la command line,sta volta con la stringa:

```
bpx __vbaNew2
```

Raggiunto il form voluto (all'indirizzo 00403403),modifichiamolo col jmp:



Fine

►NOZIONI & TRUCCHI DI REVERSING



Ecco una raccolta di quei trucchi e consigli utili che si affinano e si scoprono dopo qualche esperienza in ambito di reversing.

►ABILITARE UN BOTTON

Bè, un metodo poco ortodosso è dotarsi di software apposito, come TNT (creato da me ☺) :



Ma noi, siamo professionisti (e anche perchè il programma fa solo modifiche temporanee, fino a che il prog rimane aperto, e basta)

Infatti, dobbiamo sapere che i command button del visual basic, possono essere facilmente abilitati anche dall'hex editor. Usiamo la funzione cerca per recarci alla riga del command button (cerchiamo il valore caption, che in questo caso è :login)

Ora, vicino a tutte le vari cifre, compare un 08, e subito dopo un 00

E:\Tutorial-Test\Copia di CrackMeCtrl.exe															
3 07 af 05 77 01 08 00 11 02 0															
) b0 04 5f 0a 1d 01 12 01 00 f															
) 01 5f 0a 1d 01 12 00 00 ff 0															
) 73 77 64 00 05 f0 00 c0 03 5															
) 6c 33 00 01 01 04 00 55 73 6															
) 00 06 06 00 4c 61 62 65 6c 0															
) 00 07 01 00 00 00 00 00 00 0															

Modifichiamolo in 01, e salviamo l'exe. Aprendolo avremo il command button abilitato!

Naturalmente tutto il procedimento può essere compiuto anche dentro lo stesso debugger, dato che in genere fornisce sempre anche una funzione di hex editor.

► OLTREPASSARE LA SUBDOLA TECNICA DEI CARATTERI CON OLLY

Alcuni crackme (più che i programmi) hanno nel nome del file alcuni caratteri strani.
In particolare mi riferisco ad esempio a :

Programma%\$%\$DaCraccare.exe

Se non lo sapete, %\$ è una stringa molto utilizzata in C e C++ per pescare il valore di una variabile... probabilmente Olly credendo sia una variabile va a leggere nel nome file, (dato che deve impostarlo come nome della finestra corrente e chissà cos'altro) e genera un errore critico.

Avrete capito che per sistemare il problema è sufficiente rinominare il file.

► RICOSTRUIRE LE IMPORT

Le import table. Di cosa si tratta? Praticamente sono uno dei componenti che appartengono al formato PE, e contengono tutte le chiamate alle API utilizzate dal programma.

La tecnica della ricostruzione delle import è utilizzata specialmente (se non unicamente) dopo aver ottenuto un dump da un programma packato con un packer importante.

Ecco un esempio.

Aprire ImpREC (sezione tool del reverser) e selezionare il processo che si sta debuggando, impostare quindi l' OEP a quello desiderato (abbiamo fatto un dump no? Se non ricordate cosa è, cercate nel capitolo *Fondamenti del reversing*, se non sapete farne uno, consultare il paragrafi successivi) e premere il pulsante “IAT Auto Search”, dovrebbe comparire una messagebox contenente la IT che regolarmente è stata trovata.

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.



Premere quindi getImports ed appariranno (se è in vb6 come in questo caso,solo uno) tutti i thunk relativi alle imports.

Talvolta il programma riesce a ricostruirle pienamente,e il dump è a posto. Nel caso che segue però,ci sono stati dei problemi:

```
rva:00001008 ptr:003BA660  
...0000100C mod:msvbm60
```

Se andiamo all'indirizzo indicato (in questo caso 003BA660) con Olly apparirà un codice simile a questo:

003BA660	55	PUSH EBP	
003BA661	8BEC	MOV EBP,ESP	
003BA663	6A FF	PUSH -1	
003BA665	68 00773D00	PUSH 3D7700	
003BA66A	68 30623D00	PUSH 3D6230	
003BA66F	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	JMP to msvert._except_handler3
003BA675	50	PUSH EAX	
003BA676	64:8925 00000000	MOV DWORD PTR FS:[0],ESP	
003BA67D	83EC 10	SUB ESP,10	
003BA680	53	PUSH EBX	
003BA681	56	PUSH ESI	
003BA682	57	PUSH EDI	
003BA683	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
003BA686	33F6	XOR ESI,ESI	
003BA688	68 F4CC3D00	PUSH 3DCCF4	ASCII "MSVBVM60.DLL"
003BA68D	FF15 C0703D00	CALL DWORD PTR DS:[3D70C0]	kernel32.GetModuleHandleA
003BA693	85C0	TEST EAX,EAX	
003BA695	.74 0E	JE SHORT 003BA6A5	
003BA697	68 E8CC3D00	PUSH 3DCCE8	ASCII "__vbaEnd"
003BA69C	50	PUSH EAX	kernel32.GetProcAddress
003BA69D	FF15 C4723D00	CALL DWORD PTR DS:[3D72C4]	
003BA6A3	8BF0	MOV ESI,ESP	
003BA6A5	6A 02	PUSH 2	
003BA6A7	FF15 4C713D00	CALL DWORD PTR DS:[3D714C]	kernel32.SetErrorMode

Fare doppio click quindi sulla funzione non risolta da impRec e dall'elenco che compare selezionare “__vbaEnd” (in questo caso) e premere “OK”.

Ora le import sono a posto,e se premiamo su FixDump avremo il dump funzionante.

► REVERSING DI VB5/6

[Parte di guida creata da Predator, vedi ringraziamenti]

Visual Basic 6, è un linguaggio interpretato e, anche per eseguire semplici operazioni, genera molta confusione nel codice.

Molti ritengono noioso o difficile riversare programmi fatti in visual basic proprio per la confusione generata dal codice VB. In questo documento illustrerò alcune tecniche che sicuramente vi aiuteranno ad avere vita più facile nel reversare Visual Basic.

► Scopo abilitare un tasto disabilitato

Il sorgente in questione prevede che ci sia una funzione che esegue
Command1.Enabled=False

nel Load Form

che si occupa di disabilitare il tasto, pertanto di suo sarebbe abilitato.

Procedura:

caricate l'exe 1EnableMe.exe in Olly

ci sono vari modi per arrivare al punto esatto dove il codice disabilita il tasto, io prendo il più immediato (ottimizzo hehe), cmq vediamo tutto con calma e nei dettagli.

Appena caricato l'exe in Olly premiamo il tasto destro

Search for-> All intermodular calls

* - [Found intermodular calls]	
R	File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S	
Address	Disassembly
0040114C 1EnableM.<Module> PUSH 1EnableM.004012F4	(Initial CPU selection)
00401151 CALL <JMP.&MSUBUM60.#100>	MSUBUM60.ThunRTMain
00401B13 CALL NEAR DWORD PTR DS:[<&MSUBUM60._vbaObjSet>]	MSUBUM60._vbaObjSet
00401B37 CALL NEAR DWORD PTR DS:[<&MSUBUM60._vbaHresultCheckObj>]	MSUBUM60._vbaHresuLtCheckObj
00401B40 CALL NEAR DWORD PTR DS:[<&MSUBUM60._vbaFreeObj>]	MSUBUM60._vbaFreeObj
00401B53 CALL NEAR DWORD PTR DS:[<&MSUBUM60._vbaFreeObj>]	MSUBUM60._vbaFreeObj

La funzione che si occupa di abilitare/disabilitare i comandi è _vbaObjSet

Facciamo doppio click su quella call per andare direttamente al codice che ci interessa:

00401B13 . FF15 18104000 CALL NEAR DWORD PTR DS:[<&MSUBUM60._vbaObjSet>]	MSUBUM60._vbaObjSet
00401B19 . 8BF0 MOV ESI, EAX	ntdll.7C920738
00401B1B . 57 PUSH EDI	
00401B1C . 56 PUSH ESI	
00401B1D . 8B0E MOV ECX, DWORD PTR DS:[ESI]	
00401B1F . FF91 8C000000 CALL NEAR DWORD PTR DS:[ECX+8C]	
00401B25 . 3BC7 CMP EAX, EDI	ntdll.7C920738
00401B27 . DBE2 FCLEX	

La cosa interessante la notiamo all'offset (indirizzo) 00401B1F:

CALL NEAR DWORD PTR DS:[ECX+8C]

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Dobbiamo portare la nostra attenzione sulla costante [ECX+8], in pratica è quella che dice alla CALL di impostare Enabled = False.

Pertanto per raggiungere questo punto cruciale possiamo anche premere CTRL+F (oppure tasto destro Search for-> command) e come testo da ricercare inseriamo

CALL NEAR DWORD PTR DS:[ECX+8C]

Vi troverete esattamente all'indirizzo 00401B1F.

La soluzione per far sì che il tasto non venga disabilitato puo' essere quella di noppare la CALL :-)

Provate, al posto di **CALL NEAR DWORD PTR DS:[ECX+8C]** mettete un **NOP**, avviate il programma (F9) e il tasto sarà abilitato.

Riassumendo:

_vbaObjSet imposta lo stato enabled dei controlli

CALL NEAR DWORD PTR DS:[ECX+8C] esegue l'operazione della **_vbaObjSet** all'indirizzo specificato :-)

Esempio2:

in questo secondo esempio ci occupiamo di analizzare sempre l'abilitazione di un tasto ma questa volta nel caso che non ci sia nessun codice di attivazione/disattivazione, ovvero un tasto disabilitato dalle proprietà del compilatore.

Caricate in Olly il secondo esempio: 2EnableMe.exe

Come detto prima qui siamo in una condizione diversa, il tasto è disabilitato nelle proprietà del compilatore e non c'è nessun codice che ne modifichi lo stato, pertanto dovremo intervenire nella struttura stessa del controllo.

Procedura:

Tasto destro Search for-> All referenced text strings

Vedete la riga evidenziata in grigetto? Il nostro Command1, fate doppioclick.

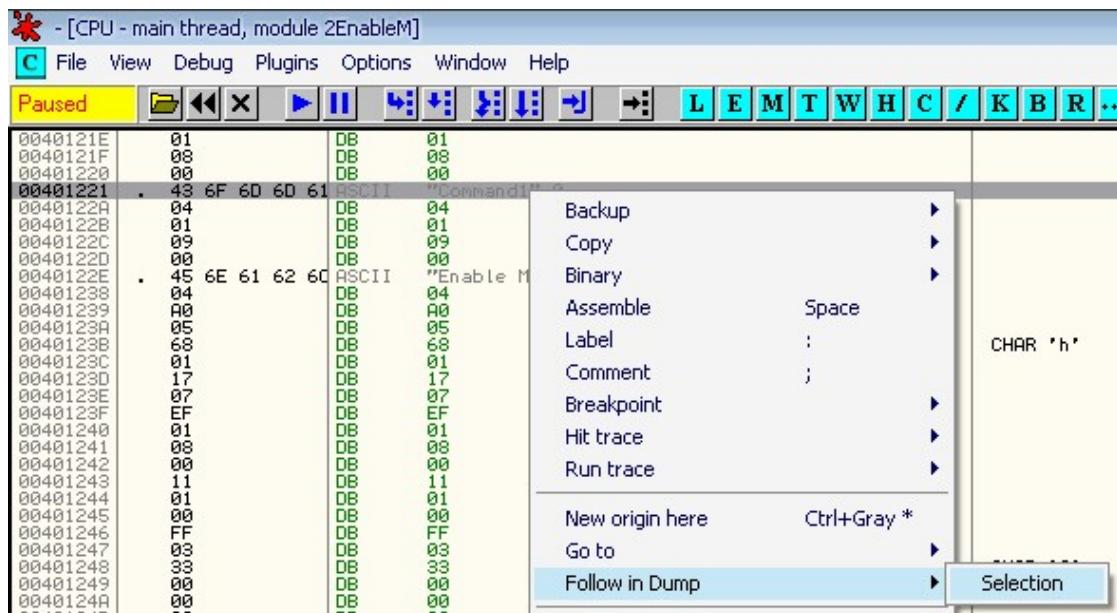
- [Text strings referenced in 2EnableM:.text]			
R	File	View	Debug Plugins Options Window Help
Address	Disassembly	Text string	
00401014 <&MSVBU60._adj_	00 MSVBU60._adj_fdiv_m16i	ASCII "P=0?"	
00401018 <&MSVBU60._adj_	00 MSVBU60._adj_fdivr_m16i	ASCII "P=0?"	
00401020 <&MSVBU60._vba	00 MSVBU60._vbaChkstk	ASCII "QWP="	
0040104C <&MSVBU60._adj_	00 MSVBU60._adj_fdiv_m32i	ASCII "P=0?"	
00401050 <&MSVBU60._adj_	00 MSVBU60._adj_fdivr_m32i	ASCII "P=0?"	
00401128 2EnableM.<Module	PUSH 2EnableM.00401200	(Initial CPU selection)	
00401164	ASCII "Progetto1",0		
004011DC	ASCII "Form1",0		
004011E6	ASCII "Form1",0		
004011FB	ASCII "Form1",0		
00401205	ASCII "5",0		
00401221	ASCII "Command1",0		
0040122E	ASCII "Enable Me",0		
0040124F	ASCII "Label1",0		
0040125A	ASCII "www.nexenteam.ne"		
0040126A	ASCII "t",0		
00401291	ASCII " 0F",0		
004012D0	ASCII "VB5!6&VB6IT.DLL",0		
00401348	ASCII "2EnableMe",0		

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

```
0040121E 01 DB 01
0040121F 08 DB 08
00401220 00 DB 00
00401221 . 43 6F 60 60 61 ASCII "Command1",0
0040122A 04 DB 04
0040122B 01 DB 01
0040122C 09 DB 09
0040122D 00 DB 00
0040122E . 45 6E 61 62 60 ASCII "Enable Me",0
00401238 04 DB 04
00401239 A0 DB A0
0040123A 05 DB 05
0040123B 68 DB 68
0040123C 01 DB 01
0040123D 17 DB 17
0040123E 07 DB 07
0040123F EF DB EF
00401240 01 DB 01
00401241 08 DB 08
00401242 00 DB 00
00401243 11 DB 11
00401244 01 DB 01
00401245 00 DB 00
00401246 FF DB FF
00401247 03 DB 03
00401248 33 DB 33
00401249 00 DB 00
0040124A 00 DB 00
0040124B 00 DB 00
0040124C 02 DB 02
0040124D 06 DB 06
0040124E 00 DB 00
0040124F . 4C 61 62 65 60 ASCII "Label1",0
00401256 01 DB 01
00401257 01 DB 01
```

Vedete la riga evidenziata in grigetto? Il nostro Command1, fate doppioclick.

Come vedete nell'immagine vi trovate nella struttura del tasto, vediamo di spiegarla:
non necessario ma utile per rendere maggiormente comprensibile le informazioni facciamo
tasto destro->follow in dump->selection come da immagine



Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Nella finestra sotto veniamo posizionati esattamente nel dump del tasto:

Address	Hex dump	ASCII
00401221	43 6F 6D 6D 61 6E 64 31 00 04 01 09 00 45 6E 61	Command1.....Ena
00401231	62 6C 65 20 4D 65 00 04 A0 05 68 01 17 07 EF 01	ble Me....h....
00401241	08 00 11 01 00 FF 03 33 00 00 00 02 06 00 4C 613.....La
00401251	62 65 6C 31 00 01 01 11 00 77 77 77 2E 6E 65 78	beli.....www.nex
00401261	65 6E 74 65 61 60 2E 6E 65 74 00 05 78 00 28 05	enteam.net..x..(.

In pratica leggiamo le stesse informazioni che abbiamo sopra disposte in verticale :-D ma qui sono piu' facili da comprendere.

E' chiaro e facile capire quanto segue:

il nome del comando è "Command1"

00401221 43 6F 6D 6D 61 6E 64 31 00 Command1.

Poi...

Address	Hex dump	ASCII
00401221	43 6F 6D 6D 61 6E 64 31 00 04 01 09 00 45 6E 61	Command1.....Ena
00401231	62 6C 65 20 4D 65 00 04 A0 05 68 01 17 07 EF 01	ble Me....h....
00401241	08 00 11 01 00 FF 03 33 00 00 00 02 06 00 4C 613.....La
00401251	62 65 6C 31 00 01 01 11 00 77 77 77 2E 6E 65 78	beli.....www.nex

0040122145 6E 61 Ena

00401231 62 6C 65 20 4D 65 ble Me

Quello che segue fino a prima di 00 04 è la **caption del controllo** "Enable Me"

Poi...

Questa è la **proprietà Left**, solo che (come sapete) nello stack le informazioni si leggono con il criterio LIFO (se non avete chiaro lo stack leggete l'opportuna documentazione che ho scritto), pertanto lo rovesciamo

A005 diventa 05A0 che tradotto in decimale è 1440, infatti se andate a vedere il sorgente vedrete che Command1.Left = 1440

Poi...

Address	Hex dump	ASCII
00401221	43 6F 6D 6D 61 6E 64 31 00 04 01 09 00 45 6E 61	Command1.....Ena
00401231	62 6C 65 20 4D 65 00 04 A0 05 68 01 17 07 EF 01	ble Me....h....
00401241	08 00 11 01 00 FF 03 33 00 00 00 02 06 00 4C 613.....La
00401251	62 65 6C 31 00 01 01 11 00 77 77 77 2E 6E 65 78	beli.....www.nex

Stesso criterio di prima questa è la **proprietà Top**

6801->0168->360 pertanto Command1.Top = 360 heheh è proprio vero eh? :P

Poi...

Address	Hex dump	ASCII
00401221	43 6F 6D 6D 61 6E 64 31 00 04 01 09 00 45 6E 61	Command1.....Ena
00401231	62 6C 65 20 4D 65 00 04 A0 05 68 01 17 07 EF 01	ble Me....h....
00401241	08 00 11 01 00 FF 03 33 00 00 00 02 06 00 4C 613.....La
00401251	62 AF AF 31 00 01 01 11 00 77 77 77 2F AF AF 78	hell.....www.nex

qui incontriamo la **proprietà Width** (la larghezza del controllo)

1707->0717->1815

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Poi...

Address	Hex dump	ASCII
00401221	43 6F 6D 60 61 6E 64 31 00 04 01 09 00 45 6E 61	Command1.....Ena
00401231	62 6C 65 20 40 65 00 04 A0 05 68 01 17 07 EF 01	ble Me....h....'
00401241	08 00 11 01 00 FF 03 33 00 00 00 02 06 00 4C 613.....La

La proprietà Height (altezza del controllo)

EF01->01EF->495

Finalmente alla fine troviamo

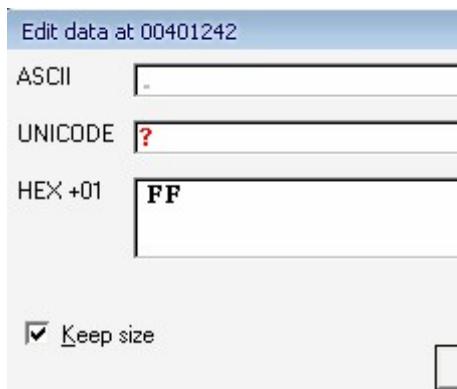
Address	Hex dump	ASCII
00401221	43 6F 6D 60 61 6E 64 31 00 04 01 09 00 45 6E 61	Command1.....Ena
00401231	62 6C 65 20 40 65 00 04 A0 05 68 01 17 07 EF 01	ble Me....h....'.
00401241	08 00 11 01 00 FF 03 33 00 00 00 02 06 00 4C 613.....La
00401251	A2 AF AF 31 00 01 01 11 00 77 77 77 2F AF AF 28	hell...www.nuv

Cioè 0800 la proprietà Enabled

00 sta per False, cambiate 00 in FF facendo tasto destro su 00

Binary -> Edit (oppure CTRL+E)

E nel falore esadecimale cambiate 00 in FF come da immagine



Date l'OK, premete F9 per runnare il processo e felici vediamo il tasto che ora è abilitato :D

Queste informazioni è utile conoscerle, ma la struttura è si può modificare facilmente con programmi adatti allo scopo, pertanto ricordiamoci soprattutto il codice spiegato nell'esempio 1.

Esempio3:

Alterare il testo inserito in una TextBox e Label.

Carichiamo in Olly l'esempio 3SetText.exe. Molto simile all'esempio 1 __vbaObjSet si occupa anche di impostare il testo nelle TextBox e Label.

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Questa volta la CALL che fa la differenza è

00401CE7 CALL NEAR DWORD PTR DS:[ECX+A4]

Simile al primo esempio, questa volta è la costante [ECX+A4] che dice di impostare il testo, poppandolo non verrà messo alcun carattere:

```
00401CDA | . 50          PUSH    EAX
00401CDB | . FF03        CALL    NEAR EBX
00401CDC | . 8BF8        MOV     EDI, EAX
00401CDF | . 68 C4174000 PUSH    3SetText.004017C4
00401CE4 | . 57          PUSH    EDI
00401CE5 | . 8B0F        MOV     ECX, DWORD PTR DS:[EDI]
00401CE6 | . FF91 A4000000 CALL    NEAR DWORD PTR DS:[ECX+A4]
00401CED | . 85C0        TEST   EAX, EAX
00401CFF | . 0F84 FF      JNE    ECX, 00401CDA
```

<&MSUBUM60.__vbaObjSet>
UNICODE "DEMO DEMO DEMO DEMO"
ntdll!7C920738

Stessa cosa per la label, solo che questa volta la costante è [ECX+A4]

Come prima poppendo la call

00401D2A CALL NEAR DWORD PTR DS:[ECX+54]

non apparirà nessun testo.

Così si possono togliere scritte non volute o diciture dai programmi :)

Per modificarne il contenuto seguite questi passi:

tasto destro all'offset **00401CE7** -> Follow in dump -> immediate constant

Per modificare il testo evidenziatelo e premete CTRL+E

(oppure tasto destro Binary->Binary Edit)

Da qui potete modificare il testo comodamente da unicode.

Abilitare una voce di menu che è disabilitata.

Nell'esempio 4MenuEnableMe, c'è un codice nel Load Form che disabilita la voce di menu, noi dobbiamo trovarla ed inibirla.

Tasto destro -> search for -> all intermodulars call

Facciamo doppioclick su **_vbaObjSet**

```
00401D98 | : 5F          PUSH    ECX
00401D99 | : FF15 1C104000 CALL    NEAR DWORD PTR DS:[<&MSUBUM60.__vbaObjSet>]
00401D99 | : 8BF0        MOV     ESI, EAX
```

MSUBUM60.__vbaObjSet

Ed ecco ancora una importante CALL e importante COSTANTE:

00401D9F CALL NEAR DWORD PTR DS:[ECX+74]

La costante [ECX+74] è molto importante perché è quella che dice se abilitare o disabilitare un menu, pertanto potete raggiungere questo punto ricercando il comando:

CALL NEAR DWORD PTR DS:[ECX+74]

e nopparlo per impedirne l'esecuzione. Il menu risulterà abilitato

Pertanto se incontrerete delle voci disabilitate a causa di un demo, ora sapete come raggiungere velocemente il punto ed abilitarla :)

Esempio5:

In questo esempio c'è un timer che lentamente (impostato a 500 ms) reimposta una TextBox Enabled = False

Metodi per inibire la disabilitazione della TextBox ce ne sono altri, ma quello che interessa a noi ora è disabilitare il timer.

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Carichiamo in Olly l'eseguibile 5Timer.exe, tasto destro -> Search for -> All referenced text string, facciamo doppio click su "Timer1",0



00401243 ASCII "5<\0" 0040125F ASCII "Timer1",0 0040127E ASCII "0x00000000\0"

Poi ancora tasto destro -> Follow in dump -> Selection



00 0B| 03 F4 01 00

Vedete il valore esadecimale evidenziato dal rettangolo rosso?

F401 -> 01F4 -> 500

È il valore dell'intervallo del timer (Timer1.interval = 500)

Editate F4 01 in 00 00 ed il timer verrà disabilitato :)

► I CRACK COMMERCIALI ©

Eccoci arrivati al capitolo più atteso. Attenzione che questa sezione non serve ad incitare voi lettori a craccare programmi commerciali perchè è lo scopo del reversing,bensì per poter studiare da vicino le protezioni che realmente compongono i programmi moderni.

Un po' come i crack me che sono la palestra dei reverser,il crack commerciale è il saggio,la partita,la verifica di ciò che avete imparato.

Vi faccio ancora notare che craccando questi 3 programmi che seguiranno,non dovrete andare in giro a sbandierarvi reverser! Avete solo letto e seguito un tutorial. Cercate di essere obiettivi,e capite che con la pratica e lo studio riuscirete a raggiungere grandi traguardi.

► PACIFIC ASSAULT

LINK: <http://www.gamecentersolution.com/downloadgame.aspx?AID=141&CID=21477>

Tools:

- **DktCompiler (il mio)**
- **OllyDbg (con plug-in della command line)**

Allora,iniziamo.Per prima cosa analizziamo il nostro eseguibile,aprendolo dentro il mio Dktr compiler...ecco il risultato:



Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Ottimo, non è neppure compresso. Ora, apriamo il gioco e analizziamolo, per vedere che tipo di protezione è stata implementata...

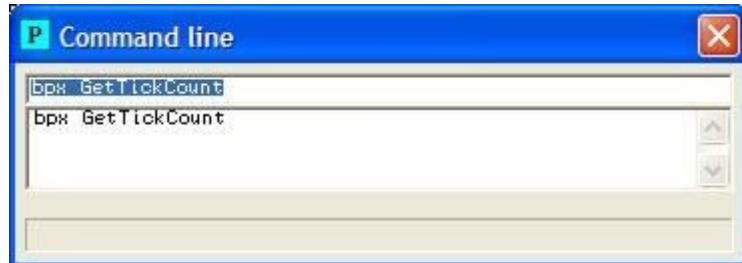


Ahi, che tristezza! Una protezione a tempo! Solo 60 minuti di gioco e poi il demo non si può più usare... No problema!

Apriamo il gioco dentro olly e cerchiamo la prima cosa logica... una funzione SetTimer che dovrebbe appunto far partire il conto alla rovescia... purtroppo non esiste nulla di simile, per cui proviamo con qualcosa di più mirato... per esempio l'API GetTickCount che dovrebbe appunto occuparsi di segnalare l'avanzamento del tempo... in questo caso in negativo, infatti ad ogni tick, il nostro counter diminuisce fino a raggiungere lo zero... la strada più veloce in questo senso è proprio quella di Noppare le chiamate all'API GetTickCount per risolvere ogni problema.

Apriamo dunque il programma, e andiamo nel menù plug-in selezionando la commandline... digitiamo

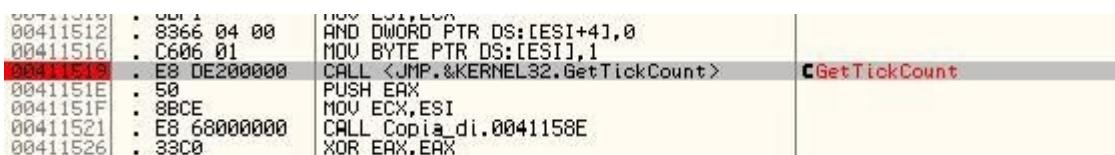
Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.



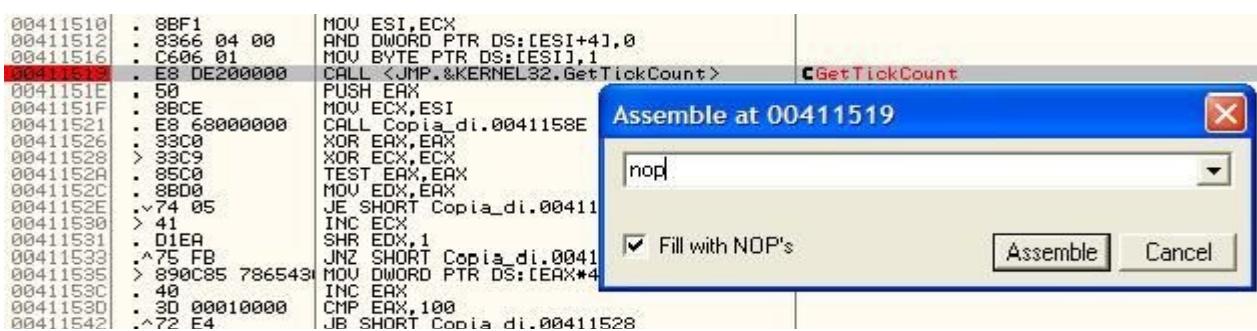
e vediamo ciò che appare...(scorrete un po la lista)

```
00407C5F CALL DWORD PTR DS:[<&USER32.MessageBoxA>] USER32.MessageBoxA
004114ED CALL DWORD PTR DS:[<&USER32.MessageBoxA>] USER32.MessageBoxA
00411513 CALL <JMP.&KERNEL32.GetTickCount> kernel32.GetTickCount
00411522 CALL <JMP.&KERNEL32.GetTickCount> kernel32.GetTickCount
0041362B CALL DWORD PTR DS:[<&USER32.LoadCursorA>] USER32.LoadCursorA
0041363B CALL ESI USER32.SetCursor
00413640 CALL EDI USER32.GetDesktopWindow
00413643 CALL DWORD PTR DS:[<&USER32.SetForegroundWindow>] USER32.SetForegroundWindow
00413652 CALL EDI USER32.GetDesktopWindow
```

queste chiamate rosse sono i breackpoint...clikkiamo su ognuno di essi,e verremo portati al codice corrispondente...per esempio...



Ottimo,dobbiamo noppare no?E faccimolo! Doppio click e scriviamo NoP



Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

benissimo, ora ecco come appare...

00411512	:	8366 04 00	AND DWORD PTR DS:[ESI+4],0
00411516	:	C606 01	MOV BYTE PTR DS:[ESI],1
00411519	90		NOP
0041151A	90		NOP
0041151B	90		NOP
0041151C	90		NOP
0041151D	90		NOP
0041151E	:	50	PUSH EAX
0041151F	:	8BCE	MOV ECX,ESI

Facciamolo anche con le altre ed il gioco è fatto!

► AXE, THE ADVANCED HEX EDITOR 3.4b

Link: <http://www.axe-editor.com/downloads/installAXE.exe>

Ecco un crack di un prog commerciale, anche utile :D

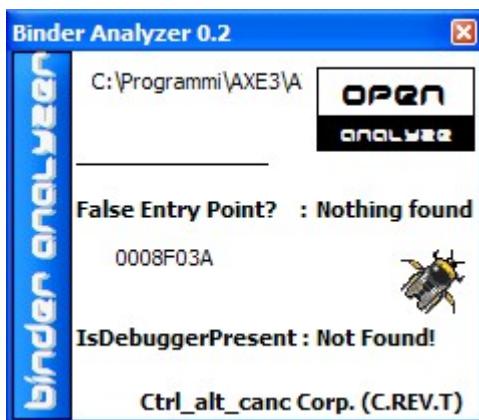
Tools

IDA Pro

Binder Analyzer (by Me)

Calcolatrice di Windows

Allora, per prima cosa analizziamo superficialmente il nostro target...



che non presenta alcun segno di protezione... benissimo.

Apriamo l'exe dentro IDA Pro, e mettiamoci ad analizzarlo:

<Http://crevt.altervista.org> <http://ctrlaltcancorp.altervista.org>

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.



cerchiamo la **GetWindowTextA**, proprio quella che legge preleva il serial number.
Poco più sotto è presente anche la riga

`call?GetWindowTextA@Cwnd@@QBEXAAVCString@@@Z`

che appunto preleva il nome.

Ora analizziamo il codice lì attorno...

```
.text:0042BBFF call?GetDlgItem@Cwnd@@QBEPAV1@H@Z ; Cwnd::GetDlgItem(int)
.text:0042BC04 mov ecx,eax
.text:0042BC06 call?GetWindowTextA@Cwnd@@QBEXAAVCString@@@Z
.text:0042BC0B mov edx,[esp+14h]
.text:0042BC0F push offsetNotFound ; "NOTFOUND"
.text:0042BC14 push edx
.text:0042BC15 callds:_stricmp
.text:0042BC1B add esp,8
.text:0042BC1E cmp eax,ebx
.text:0042BC20 jnz short loc_42BC2E
.text:0042BC22 push ebx
.text:0042BC23 push ebx
.text:0042BC24 push offsetNoMatchesFound ; "No matches found."
.text:0042BC29 call?AfxMessageBox@@YGH_PBDII@Z ; AfxMessageBox(char const*, uint, uint)
.text:0042BC2E loc_42BC2E: ; CODE XREF: .text:0042BC20j
.text:0042BC2E mov eax,[esp+18h]
.text:0042BC32 lea edi,[esi-64h]
.text:0042BC35 push edi
.text:0042BC36 push offsetLu ; "%lu"
.text:0042BC3B push eax
.text:0042BC3C callds:sscanf
.text:0042BC42 add esp,0ch
.text:0042BC45 lea ecx,[esp+14h]
.text:0042BC49 lea ebp,[esi-68h]
.text:0042BC4C push ecx
.text:0042BC4D mov ecx,ebp
```

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

```
.text:0042BC4F call??4CString@@QAEABV0@ABV0@@Z ; CString::operator=(CStringconst&)
.text:0042BC54 push ecx
.text:0042BC55 mov ecx,esp
.text:0042BC57 mov [esp+14h],esp
.text:0042BC5B push ebp
.text:0042BC5C call??0CString@@QAE@ABV0@@Z ; CString::CString(CStringconst&)
.text:0042BC61 mov edx,[edi]
.text:0042BC63 mov eax,[esi-60h]
.text:0042BC66 push edx
.text:0042BC67 push eax
.text:0042BC68 mov byteptr [esp+74h],1
.text:0042BC6E callsub_410A10
.text:0042BC77 mov ecx,eax
.text:0042BC7E callsub_41FBB0
.text:0042BC7F cmp eax,ebx
.text:0042BC7F jz loc_42BFA
```

Ottimo, queste righe sono interessanti... infatti vediamo una chiamata alla sub 41FBB0, e una successiva comparazione (cmp) di eax e ebx, in ultimo luogo un salto condizionato... rechiamoci col mouse sopra la sub 41FBB0

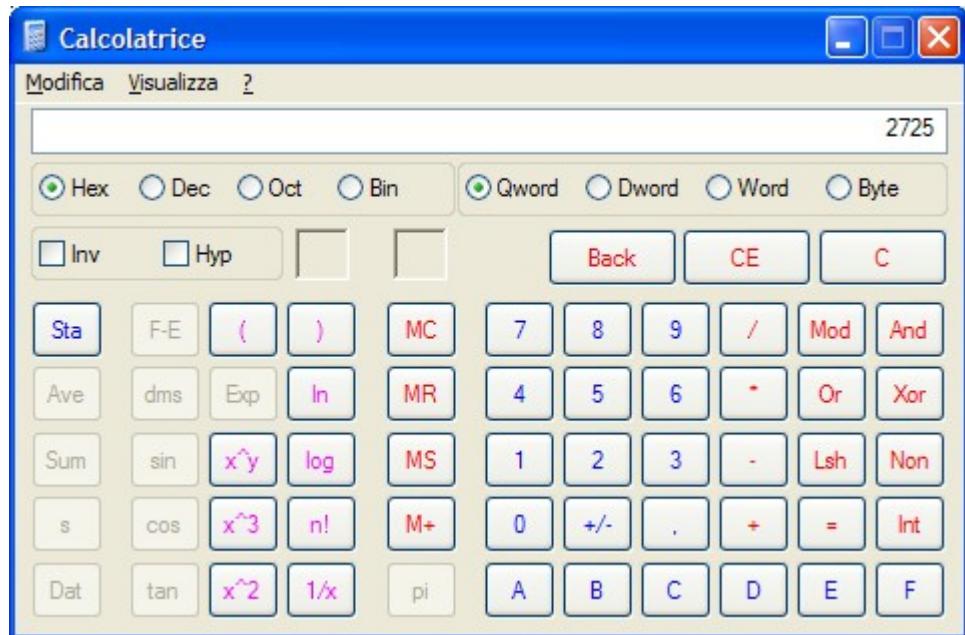
Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

```
sub_42F620    proc near                ; CODE XREF: sub_42EE40+67C↑p  
               ; sub_42F740+198↓p ...  
  
var_C          = dword ptr -0Ch  
var_4          = dword ptr -4  
arg_4          = dword ptr 8  
arg_8          = dword ptr 0Ch  
  
        mov     eax, large fs:0  
        push    0FFFFFFFh  
        push    offset loc_4DAE48  
        push    eax  
        mov     large fs:0, esp  
        push    ebx  
        push    esi  
        push    edi  
        mov     eax, [esp+18h+arg_8]  
        mov     [esp+18h+var_4], 0  
        mov     ecx, [eax-8]  
        test   ecx, ecx  
        jz     loc_42F70E  
        mov     esi, [esp+18h+arg_4]  
        test   esi, esi  
        jz     loc_42F70E  
        cmp     esi, 2725h  
        jnz    short loc_42F68D  
        lea     ecx, [esp+18h+arg_8]  
        mov     [esp+18h+var_4], 0FFFFFFFh  
        call   sub_4B3861  
        mov     eax, 1
```

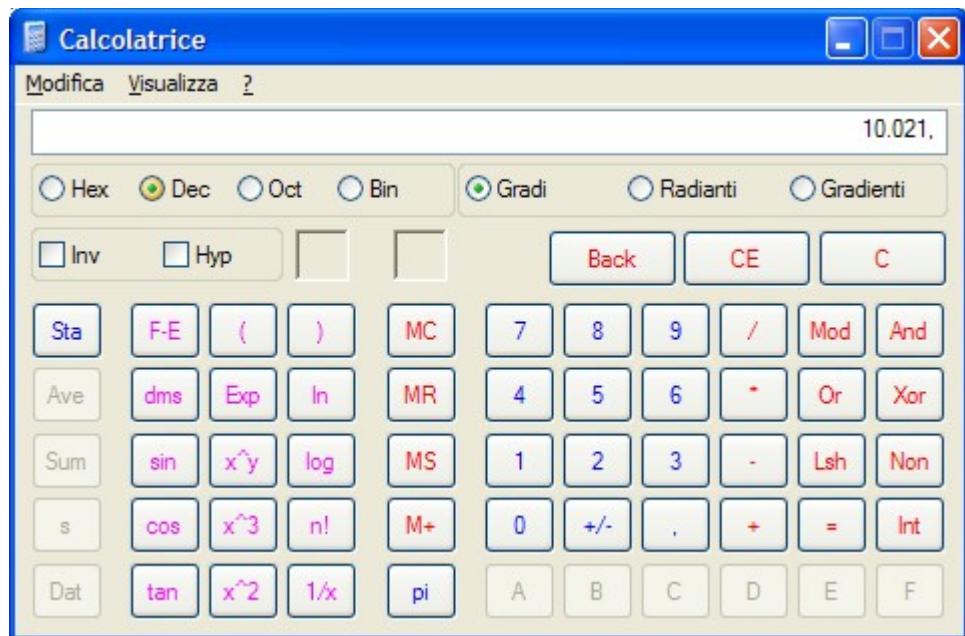
eheh....guardiamo bene quella stringa...prima viene messo in esi il valore inserito nella finestra di registrazione e ora viene paragonato con il valore 2725h,che prendendo la calcolatrice di windows...

selezioniamo la scientifica,e settiamo come unità “hex”.Copiamo 2725

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

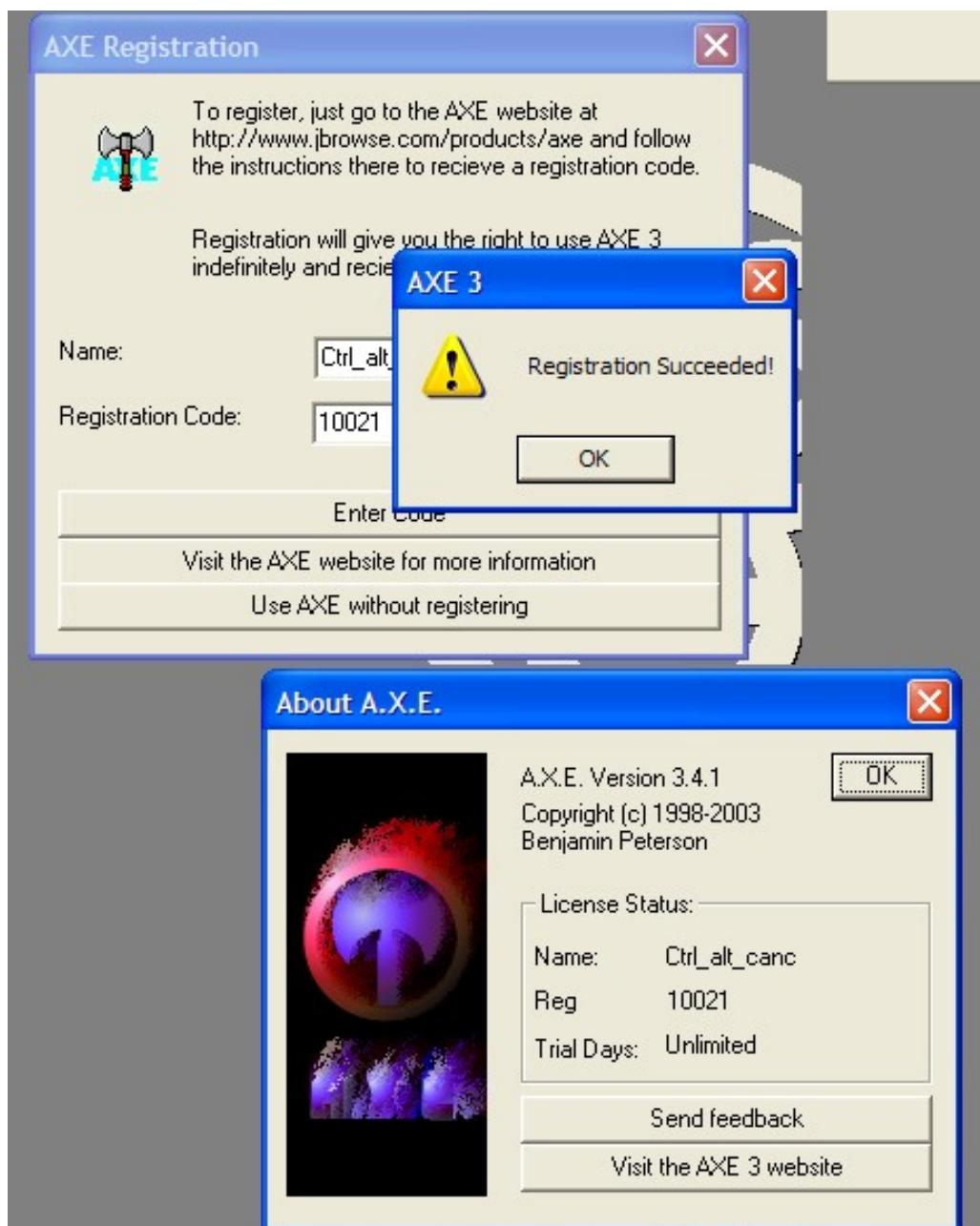


ora clikkiamo su Dec...



il valore giusto è 10021...proviamo ad inserirlo nella interfaccia di registrazione

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.



Crakkato!

►DAMA DELUX 3.1

Link: <http://www.temporale.net/DAMA31S.ZIP>

Tool:

- **OllyDbg 1.10**
- **IDA Pro 5.1 (o inferiori)**

Premessa

Se aprite l'exe noterete le
“protezioni” che i programmati hanno inserito.Ecco quindi i nostri obiettivi:

- Rimuovere il nag iniziale con quel fastidioso countdown di 20 secondi
- Rimuovere la limitazione delle mosse (Provate a finire una partita,ad un certo punto appare un messagebox che dice che la mossa la farà lui -.-')

Passo uno – Eliminare il nag

Dunque,è evidente (fidatevi) che non essendo stato fatto uso di una funzione ShowWindow nel programma per mostrare il nag,probabilmente si lavora sulla FormActivate.

Ora,aprendo il nostro Dama dentro un hex editor qualsiasi e cercando il riferimento a quella stringa,si viene portati all'indirizzo 004020A8 .

Apriamo IDA e diamo una occhiata...

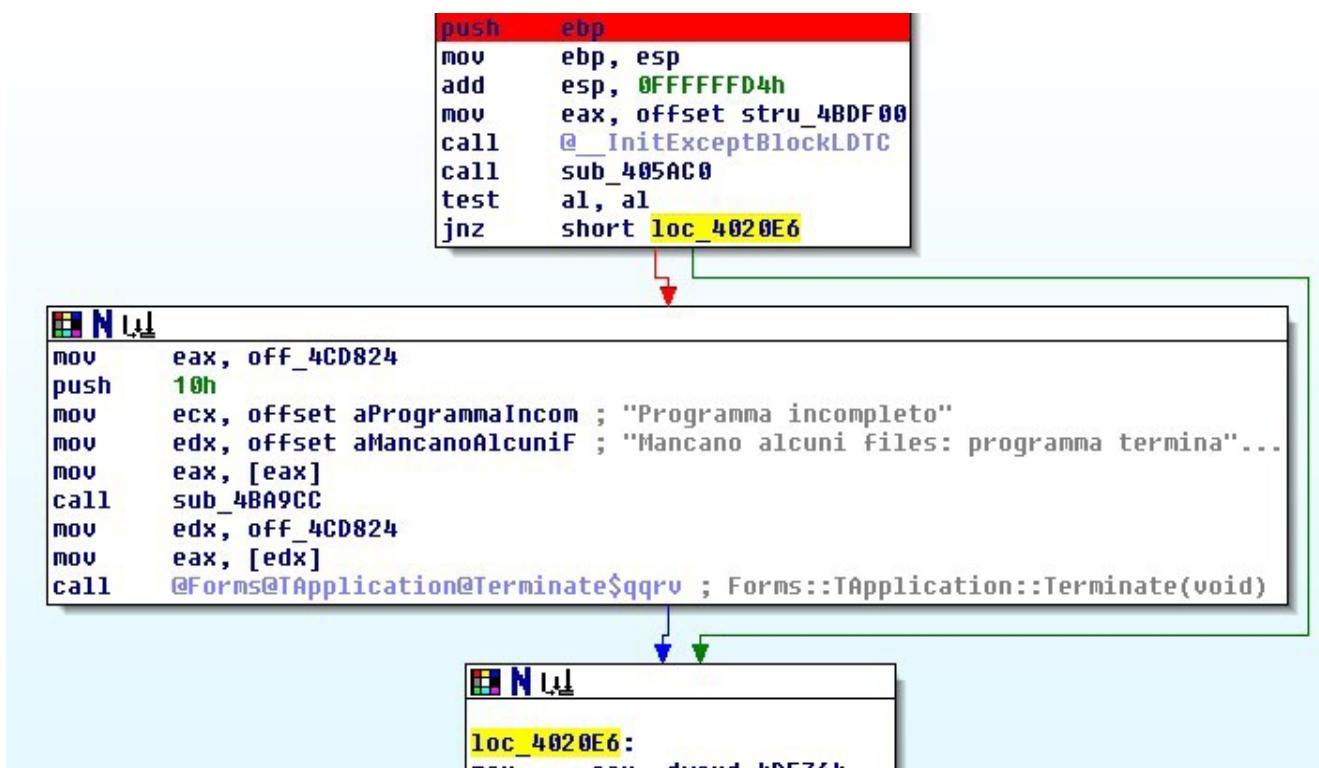
Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

```
.text:004020A8 var_10          = dword ptr -10h
.text:004020A8 var_8           = dword ptr -8
.text:004020A8 var_4           = dword ptr -4
.text:004020A8
.text:004020A8     push    ebp
.text:004020A9     mov     ebp, esp
.text:004020AB     add     esp, 0FFFFFFD4h
.text:004020AE     mov     eax, offset unk_4BDF00
.text:004020B3     call    sub_4AFF88
.text:004020B8     call    sub_405AC0
.text:004020BD     test    al, al
.text:004020BF     jnz    short loc_4020E6
.text:004020C1     mov     eax, off_4CD824
.text:004020C6     push    10h
.text:004020C8     mov     ecx, offset aProgrammaIncom ; "P
.text:004020CD     mov     edx, offset aMancanoAlcuniF ; "M
.text:004020D2     mov     eax, [eax]
.text:004020D4     call    sub_4BA9CC
.text:004020D9     mov     edx, off_4CD824
.text:004020DF     mov     eax, [edx]
```

Se settiamo un brekpoint e poi premiamo F9,ecco che IDA brekka prima di mostarre il dialog. Benissimo,lasciamo pure aperto IDA e apriamo Olly,dentro il quale caricheremo sempre Dama delux.

Ctrl+G e ci rechiamo al suddetto indirizzo.Analizziamo ciò che si vede dentro IDA,che con la sua funzione di grafico ci fa capire perfettamente il codice...ripeschiamo IDA e premiamo la barra spaziatrice,ed ecco quello che appare:

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.



Bene bene...quel jnz controlla la mancanza di file necessari,e in caso affermativo lo segnala al programma. Altrimenti prosegue,ma questa routine non ci interessa...piuttosto,proviamo (sta volta dentro olly) a settare un breakpoint sul Push Ebp (004020A8) e runniamo il prog con F9.

Olly brekka,ma noi steppiamo con f8,e in corrispondenza della call
004020ED FF92 D8000000 CALL DWORD PTR DS:[EDX+D8]

Ecco che appare la finestra del nag. :) trovato il punto da patchare!! Sostituiamo con un NOP e...magia! Sparito il nag.

Fase 2 – Limitazione Mosse

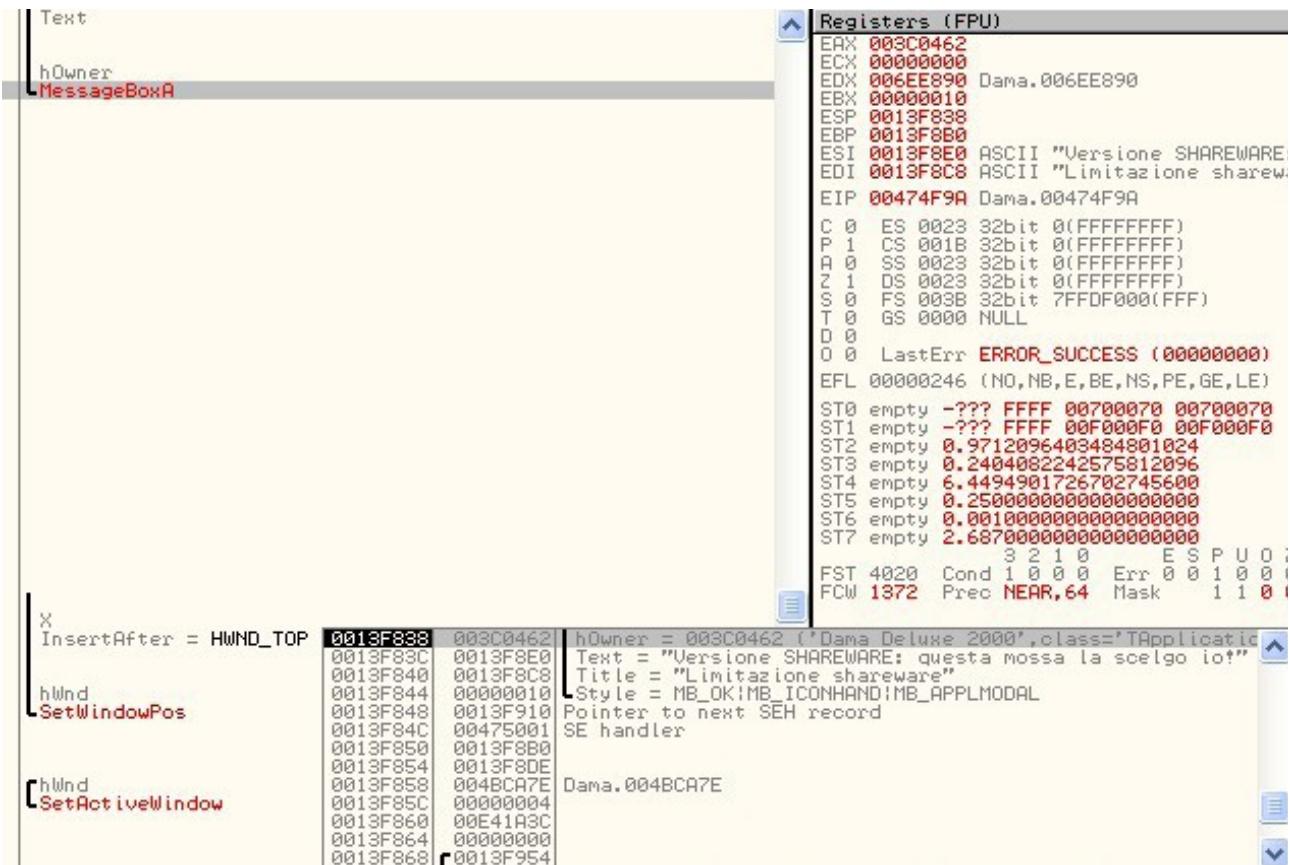
Abbiamo detto prima che appare un message box no? Ottimo,dentro olly settiamo come di consueto un bp:



Proprio qui,dopo un tot di mosse olly brekka...

00474F9A |. E8 F16D0400 CALL <JMP.&USER32.MessageBoxA>;\MessageBoxA
analizziamo la situazione da olly:

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.



Eheh,sia nello stack che nei registri appare il testo della messagebox.Se ora steppiamo,arriviamo qui:



Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Ottimo, notiamo nulla? Se guardiamo nei registri, proprio agli indirizzi pushati qui:

00406C5C lea ecx, [ebp-008C]

00406C62 lea edx, [ebp-74]

ci sono le righe "Limitazione shareware" e "Versione SHAREWARE: questa mossa la scelgo io!" .

```
Registers (FPU)
EAX 00430462 Dama.00430462
ECX 00000000
EDX 006EE890 Dama.006EE890
EBX 00000010
ESP 0013F838
EBP 0013F800
ESI 0013F8E0 ASCII "Versione SHAREWARE"
EDI 0013F8C8 ASCII "Limitazione shareware"
EIP 00474F9A Dama.00474F9A
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty -??? FFFF 00700070 00700070
ST1 empty -??? FFFF 00F000F0 00F000F0
ST2 empty 0.9712096403484801024
ST3 empty 0.2404082242575812096
ST4 empty 6.4494901726702745600
ST5 empty 0.25000000000000000000000000000000
ST6 empty 0.00100000000000000000000000000000
ST7 empty 2.35900000000000000000000000000000
          3 2 1 0   E S P U O
FST 4020 Cond 1 0 0 0 Err 0 0 1 0 0
FCW 1372 Prec NEAR,64 Mask 1 1 0
```

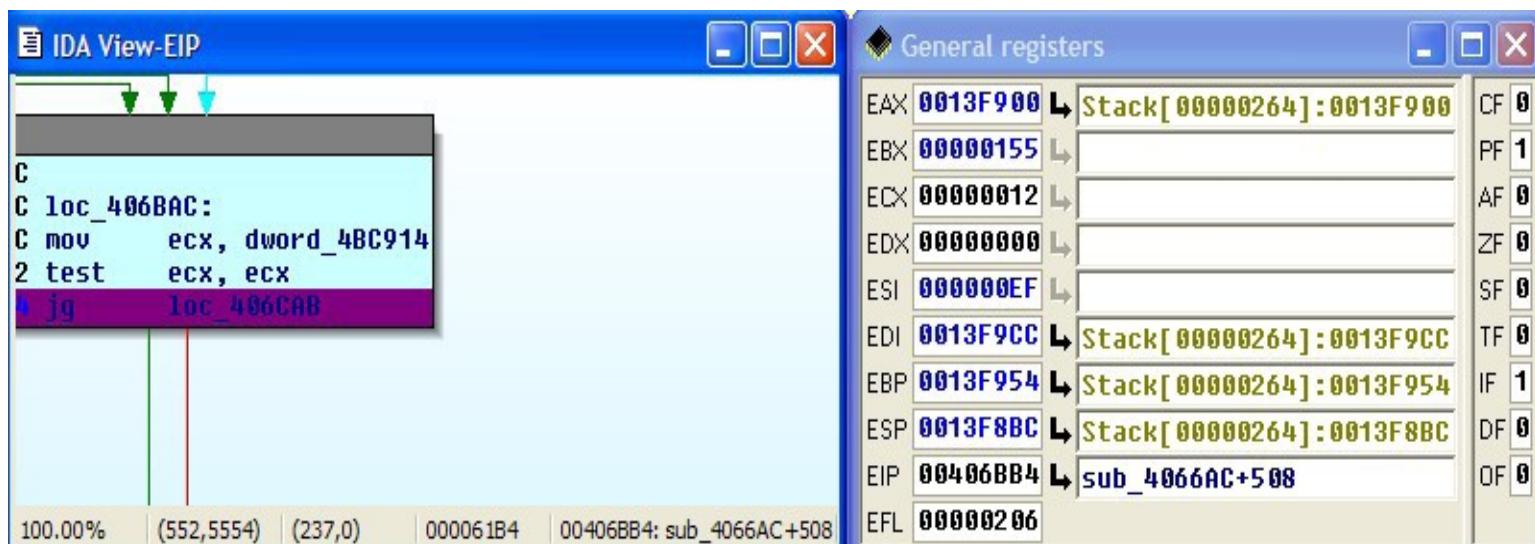
Intuiamo subito che all'indirizzo 004BA9CC abbiamo una funzione ponte per l'API MessageBoxA (cè la call no?).

Ora, ritorniamo al fedele IDA, e risaliamo fino al codice che controlla quella routine:

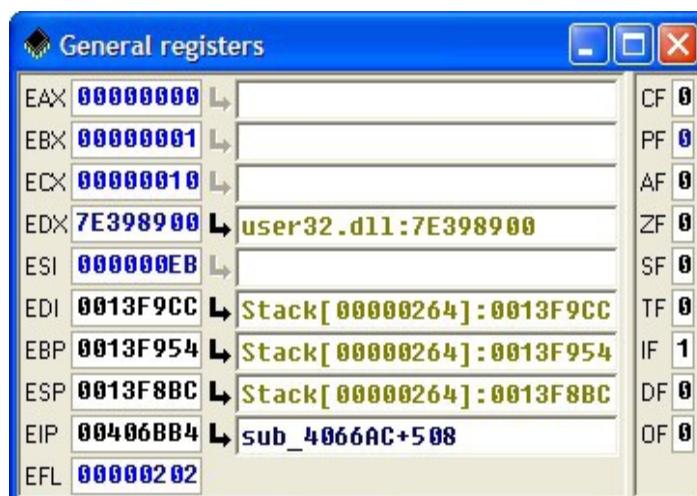
```
.text:00406BAC loc_406BAC: ; CODE XREF
.text:00406BAC                 ; sub_406BAC
.text:00406BAC                 mov    ecx, dword_4BC914
.text:00406BB2                 test   ecx, ecx
.text:00406BB4                 jg    loc_406CAB
.text:00406BBA                 mov    al, byte_4CDC84
.text:00406BBF                 test   al, al
.text:00406BC1                 jz    loc_406CAB
.text:00406BC7                 mov    dl, byte_4CDCAB
.text:00406BCD                 test   dl, dl
.text:00406BCF                 jnz   loc_406CAB
.text:00406BD5                 mov    cl, byte_4CDD25
.text:00406BDB                 mov    al, byte_4CDCAD
.text:00406BE0                 cmp    cl, al
|                   jnz   loc_406CAB
.text:00406BE2                 lea    edi, [ebp+s]
.text:00406BE8                 mov    esi, offset unk_4BCA39
.text:00406BEB                 mov    ecx, 0Bh
.text:00406BF0                 lea    eax, [ebp+s]
.text:00406BF5                 rep    movsd
```

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

Uhm...tutti gli indirizzi puntano lontano dalla messagebox...settiamo un bp su jg loc_406CAB
Ecco che alla prima mossa brekka...



Occhio eh...EXC è a valore 00000012...Aspettiamo una mossa,ed ecco che brekka ancora...^^



00000010...è diminuito di due...e se notate,questo continua ad ogni mossa,fino allo zero,quando poi arriva la call alla messagebox...
Che fare quindi? Semplicissimo!Cambiamo il condizionale jg in un jmp e tutto sarà apposto!

Fine!

I TOOL DI CTRL ALT CANC

Per evitare perdite di tempo,e spreco di spazio vi rimando al mio sito sezione tool,precisamente qui: <http://crevt.altervista.org/CT.php>

► RINGRAZIAMENTI

Con questa guida ho cercato in due settimane di lavoro di tramandarvi una parte della mia esperienza nel campo di questa affascinante arte che amo chiamare Reversing.

E' un po' il bambino che c'è in noi, che vuole giocare con il suo giocattolo e lo rompe per vedere se può "fare di più".

Soltanto noi possiamo apprezzare l'immane lavoro che il nostro computer compie ogni giorno per noi, quando scriviamo un testo o quando giochiamo a Unreal Tournament.

Non mi dilungo su altri discorsetti filosofici, e passo ai ringraziamenti.

Persone come Predator (<http://predator.forumup.it>) e Evolution (<http://ogmdeveloper.org>) non possono mancare nella lista. Sono degli ottimi reverser, senza i quali non avrei imparato nulla, e grazie ai loro forum ho approfondito quest'arte abbracciato da una community molto ospitale.

Volevo ringraziare in particolare Predator che mi ha introdotto, seppur involontariamente in questo fantastico mondo, stimolandomi e aiutandomi, oltre ad aver gentilmente concesso il suo prezioso tempo per una revisione tecnica generale di questo documento. Ricordo che il tutorial del capitolo sulle protezioni, sezione Trial è creato da lui e hostato sul suo bellissimo sito.

Inoltre la parte di guida dedicata al reversing di visual basic 5 e 6 è da lui realizzata.

Ringrazio inoltre Dr. Tod, del quale tempo addietro ho studiato le guide e da cui ho appreso le interessanti nozioni sulla CPU e sui sistemi di enumerazione.

E ringrazio te, lettore che con la tua pazienza sei arrivato al termine di questa ennesima guida al reversing, che l'autore ha tentato di rendere comprensibile a tutti, e facile da leggere.

Spero di esserci riuscito.

In ultimissimo, vi rammento che la guida è totalmente gratuita e distribuibile, l'unico sforzo che vi chiedo, è di recarvi nel mio sito <http://ctrlaltcancorp.altervista.org> e di cliccare nei google adsense che stanno in basso nella colonna di destra. Non è per interesse personale, ma i soldi che guadagnerò li utilizzerò per comprarmi un dominio :-D

Vi lascio con la mia "frase":

Guida al Reversing • C.REV.T • Ctrl_alt_canc Corp.

“The end of the exe is near...the profezy is coming true!”
-2006 Ctrl_alt_canc-

Ctrl_alt_canc

Il seguente testo è a scopo illustrativo e informativo . L'autore del seguente testo non si assume responsabilità delle informazioni assimilate e delle azioni che ne possono derivare . Qualunque atto che vada contro la legge informatica n.23 art. 1,2,3,4,5,6,7,8,9,10,11,12,13 è punibile e perseguitabile penalmente
Le azioni riportate nel documento sono state testate su macchine di mia stessa proprietà. Quest'opera è stata rilasciata sotto la licenza Creative Commons Attribuzione-Non commerciale-Non opere derivate 2.5 Italia. Per leggere una copia della licenza visita il sito web <http://creativecommons.org/licenses/publicdomain>



Vorrei ricordare che il software va comprato e non rubato, dovete registrare il vostro prodotto dopo il periodo di valutazione. Non mi ritengo responsabile per eventuali danni causati al vostro computer determinati dall'uso improprio di questa guida. Questo documento è stato scritto per invogliare il consumatore a registrare legalmente i propri programmi, e non a fargli fare uso dei tantissimi file crack presenti in rete, infatti tale documento aiuta a comprendere lo sforzo immane che ogni singolo programmatore ha dovuto portare avanti per fornire ai rispettivi consumatori i migliori prodotti possibili.