Phoenix's Crackme 1 by Phoenix87		
Data	by Andy74	
03 / gennaio / 2004	***UIC's Home Page***	**Published by Quequero**
	Un po' caotici i commenti ma il tute e' ben scritto e ben commentato, grazie Andy	
....	**E-mail:** Andy74@wolf-web.com **Nick: Andy74**	....
Difficoltà	**( )NewBies ( )Intermedio ( )Avanzato ( )Master**	

Salve, reverseremo il cracme di "Phoenix87" (Phoenix's Crackme 1) che come dicevano le voci ERA incrackabile.

Phoenix's Crackme 1
by Phoenix87

Written by Andy74

Introduzione

Salve a tutti, sono al mio primo tutorial e spero mi perdonerete se lo troverete difficile o incomprensibile, ma almeno ci provo. Se comunque lo state leggendo è evidente che Quequero me lo ha publicato, lo ringrazio quindi anticipatamente per l'occasione che mi ha dato. Sono entrato nel mondo del reverse engineering da qualche anno ormai, e mi sono divertito a reversare qualche programmino per me e i miei amici, ma non mi ero mai proposto di scrivere un tutorial a disposizione di quelli che, come a me, piaceva imparare, e visto che io ho imparato da quelli degli altri adesso mi sono deciso (scusate se mi sto dilungando, arrivo al punto). Dunque ho scelto proprio questo cracme di Phoenix87 perchè penso che non esista e non esisterà mai l'anticrack perfetto per eccellenza, a meno di programmi client-server basati su tecnologie com, com+, atl, ecc..., dove si passa nel ramo dell'hacking, ma questo potrebbe essere l'argomento di un altro tutorial (sempre che questo abbia successo). Ma lasciamo che il tempo faccia il suo corso e passiamo a reversare questo crackme.

Tools usati

Tools usati:
ProcDump32 V. 1.6.2 Final (credo esista una versione più aggiornata)
w32Dasm V. 10 (può andar bene anche una versione precedente, ma non precedente la 8.93)

per i download potete andare qui.

URL o FTP del programma

Quequero lo conoscete? :)
sezione crackme! :)

Notizie sul programma

Richiede un nome utente e un seriale per la registrazione.

Essay

Bene, prima di cominciare vi voglio dire che non sono un genio in questo campo, mi cimento nel reversing solo per puro divertimento e per ampliare le mie conoscenze sul crack e sull'anticrack. Sono un programmatore e devo dire che quello che ho imparato reversando mi ha illuminato e chiarito ciò che alle volte facevo programmando senza sapere con precisione quello che accadeva a livello macchina (il più delle volte anche con effetti sbalorditivi).
Ma andiamo oltre.
Innanzi tutto diamo un'occhiata a con che cosa abbiamo a che fare... lanciate il programmino che dobbiamo reversare e dategli un'occhiata...
Ci si presenta una semplice finestra con poche informazioni, sul menù ci sono due voci "file" e "Help".
Benissimo troverete facilmente la voce "Register", selezionatela ed ecco il nostro vero nemico!
Vi consiglio, per capire bene con che cosa avete a che fare, di eseguire delle prove di diverso genere e segnarvi come si comporta il programma che avete davanti.
Le cose che potrete notare è che ci sono diversi controlli sui campi inseriti quindi i messaggi:

"You've forgotten to digit the name!"
o
"You've forgotten to digit the serial!"
ed infine
"Sorry! The serial you have entered is invalid. Work harder, reverser!!!"
simpatico ragazzo, ci prende anche in giro! :)

Un'altra cosa che potrete notare è che i caratteri del seriale sono in maiuscolo... non che ci interessi più di tanto, ma è sempre un'informazione!
Bene una volta che vi sarete segnati quello che vi interessa (nel nostro caso un po' di messaggi) procediamo con il passo successivo.
Una delle prime cose che cercheremo di fare da bravi reverser è quello di disassemblare il codice, quindi apriamo il nostro bel w32Dasm e apriamo il file interessato. Subito ci accorgiamo che qualcosa non va... tra le reference string non vediamo un bel ninete (e questo non va bene visto che noi ci siamo segnati dei messaggi da cercare); questo può voler dire molte cose, ad esempio che il file è patchato, il PE è stato modificato, e tante altre belle cose. Come procediamo? bene innazi tutto potremmo utilizzare qualche programma quali gtw256, unpack22, studPE ed altri che ci rivelano eventuali packer o patcher utilizzati per proteggere il file. Risparmiatevi la fatica, nessuno di quelli che ho elencato riesce a trovare con che programma è stato protetto il file (sempre che lo sia). La soluzione allora è un'altra (ma non solo questa, ce ne sono un'infinità, tutto sta a trovare quella che vi piace di più o che vi è più comoda ;)), un programma, pur se protetto, quando viene lanciato ha la necessità, per essere eseguito correttamente, di essere ripristinato nel suo stato originale. In memoria quindi troveremo sicuramente quello che cerchiamo! Come arrivarci? utiliziamo il ProcDump! Lanciate quindi questo tool di cui sarete sicuramente in possesso e dumpate il processo che vi interessa. Bene immagino che avete anche salvato il nuovo eseguibile, ma se non sarete fortunati (come me) avrete il mio stesso problema, l'eseguibile dumpato non si lancia (se siete più fortunati di me sono contento per voi! :)) .
Non ci faremo certo fermare da questi piccoli problemi, anche perchè ora vi confesso un segreto... questo file dumpato non ci servirà... o almeno solo in parte... :) ma procediamo.

Disassemblate il vostro file dumpato con il w32Dasm e... oh... sorpresa... tra le reference string adesso appare qualcosa, proprio quello che stavamo cercando! :)
Ok con un veloce double-click rechiamoci nella locazione di memoria dove si trova la stringa che ci interessa, io direi proprio quella con il messaggio "Sorry! The serial you have..." (sto ipotizzando che voi tutti sappiate utilizzare il w32Dasm).
Se date un'occhiata ai messaggi che compaiono in questa sezione di codice ne troverete alcuni familiari ed uno nuovo nuovo che credo vi piacerà ancor di più! ;)
Ok... adesso dobbiamo prendere nota di quello che abbiamo scoperto e farne buon uso... prima vi ho detto che io non riuscivo ad eseguire il programma dumpato, non so voi, ma se riuscite a lanciarlo potete lavorare direttamente su questo, per gli altri che come me il dumpato non funziona dovranno disassemblare di nuovo l'originale, ma prima segnatevi qualche indirizzo... direi che ce ne basta uno... date un'occhiata al codice risalite di qualche istruzione e prendete nota dell'indirizzo, direi che qualche istruzione dopo il SendMessageA che troverete andrà bene; facciamo subito dopo il "jne 00401584". Quindi l'indirizzo che vi dovrete segnare sarà "004014CC" .
Una volta disassemblato l'originale andate all'indirizzo che vi siete salvati e procediamo finalmente nei passi che ci permetteranno di scovare i segreti di questo piccolo programma.
Prima di continuare però volevo farvi notare quanto abbiamo sudato per trovare solo il punto in cui attaccare il programma!
Bene lanciate il programma attraverso il w32Dasm e posizionate un bel BreakPoint all'indirizzo 004014CC.
Eseguite tranquillamente e passiamo finalmente a capire il codice. Avviate la finestra di registrazione e mettete delle informazioni a caso nella edit del name e del serial e registrate, il Dasm si brekerà all'inirizzo specificato e ci ritroveremo ad analizzare il codice.

```
:004014CC   mov  ebx, 00401094
:004014D1   push 00403050
:004014D6   push 0000003C
:004014D8   push 0000000D
:004014DA   push 00000BB9
:004014DF   push [ebp+08]
non appena arriverete qui si aprirà la maschera della chiamata alle API e scopriamo che il programma lancia un messaggio (SendDlgItemMessageA) alla finestra di registrazione ((HWND) 00140a68 (Window"Register Phoenix1")) prendendo il testo (WM_GETTEXT) di uno dei controlli (00000bb9), bene se volete potreste usare un programma tipo "spy" e verificare che il controllo è la edit del nome! ;)
:004014E2   call dword ptr [0040326E]
:004014E8   test eax, eax (controlla che la stringa non sia vuota)
:004014EA   je   00401568
e qui abbiamo una bella chiamata a funzione, ma che strano! ;) bene, l'analizzaremo più avanti
:004014EC   call 0040158D
:004014F1   mov  ebx, 004010B8
:004014F6   push 00403050
:004014FB   push 0000003C
:004014FD   push 0000000D
:004014FF   push 00000BBA
:00401504   push [ebp+08]
anche qui una chiamata alle API stessa cosa, solo che stavolta cambia il controllo... volete indovinare voi? ;)
:00401507   call dword ptr [0040326E]
:0040150D   test eax, eax (solito controllo di stringa vuota!)
:0040150F   je   00401568
:00401511   mov  ebx, 004010DE
bene... qui abbiamo un altro controllino sulla stringa, in questo momento in eax abbiamo la lunghezza della stringa del seriale, potete controllare inserendo semplicemente seriali di diversa lunghezza, e vi confermerete che è proprio quello! quindi controlla la lunghezza della stringa, ma con che valore? andiamo a scoprirlo! (forse è questo che intendeva per incrackabile, il mio errore iniziale è stato di non dare peso a questo valore, e tra poco saprete perchè)
:00401516   cmp  eax, dword ptr [00403251]
:0040151C   jne  00401568
un'altra chiamata a funzione, non vi sembra strano che dopo il recupero delle stringhe e dei primi controlli su di essi ci siano delle chiamate a funzioni ? ;) ma analizzaremo anche questa più avanti
:0040151E   call 004015B7
:00401523   mov  ebx, 004010DE
ecco il controllo che ci interessa, se eax è uguale a zero non va bene! ricordatevelo mi raccomando
:00401528   test eax, eax
:0040152A   je   00401568
:0040152C   push 00000040
:0040152E   push 00401023
:00401533   push 0040118F
:00401538   push [ebp+08]
:0040153B   call dword ptr [0040326A]
:00401541   push [ebp+08]
```

Ok, ora il nostro primo traguardo è quello di scoprire quanto deve essere la lunghezza del seriale da inserire, ed anche qui siamo fortunati, l'istruzione era "cmp eax, dword ptr [00403251]", bene allora partiamo dall'inizio e cerchiamo questo valore per tutto il codice. Lo troveremo in tre punti, ma uno dei tre è quello che abbiamo già analizzato , quindi lanciamo il programma dal w32Dasm e breckiamo gli altri due. Con somma soddisfazione scopriremo che quello che ci interessa è quello in 00401369. Come al solito ci spostiamo un po' prima, credo che dopo la sequenza di call (3 in cascata) basterà, quindi lanciate nuovamente e breckate in 00401329.

```
:00401329   push 00403000
:0040132E   push 00000003
:00401330   push 00000404
:00401335   push 000003EA
:0040133A   push [ebp+08]
:0040133D   call dword ptr [0040326E]
:00401343   push [ebp+08]
l'unica funzione che viene chiamata tra il nostro break e l'inizializzazione della locazione di memoria che ci interessa (00403251) è questa... vogliamo analizzarla prima di andare avanti? io direi proprio di si!
:00401346   call 00401684
```

---

questa è la funzione che ci interessa...

```
:00401684 push ebp
:00401685 mov ebp, esp
:00401687 push 00403038
se date un'occhiata alla finestra alla destra del w32Dasm (dove c'è l'interpretazione in tempo reale dell'indirizzo 0040325E) vedrete che si sta per eseguire una "call kernel32.GetLocalTime", ecco l'arcano mistero. Questo vuol dire che la lunghezza del seriale sarà definita in base al giorno in cui si eseguirà la registrazione. Questo è quello che inizialmente mi ha portato fuori strada, in principio avevo preso il valore contenuto in quella locazione di memoria come costante, e ho quindi basato il keygen che avevo implementato su questo, ma il giorno dopo, mentre scrivevo questo tutorial mi sono accorto che il keygen non funzionava più, così ho scoperto che il valore che avevo ipotizzato come costante non lo era affatto, e sono andato alla ricerca della funzione che generava questo valore. Ma andiamo avanti...
:0040168C call dword ptr [0040325E]
```

il resto del codice non fa altro che formattare la data per visualizzarla sulla finestra principale del programma e salvare il giorno e il mese in due locazioni di memoria che però scopriremo ora...

```
:00401346   call 00401684
proseguiamo da qui!
:0040134B   xor  edx, edx
adesso controllate cosa c'è nella locazione di memoria 00403048 e 0040304C... mese e giorno attuali (se non vi fidate potete modificare la data del vostro orogloggio di sistema e ricontrollare... ovvio... rilanciando il programma, infatti questa funzione viene lanciata all'avvio dello stesso). Vediamo cosa succede.

una xor tra il mese e il valore 13 (0000000D)
:0040134D   xor  dword ptr [00403048], 0000000D
sommiamo il giorno
```

```
:00401354   mov  eax, dword ptr [00403048]
:00401359   add  eax, dword ptr [0040304C]
dividiamo per 2
:0040135F   mov  ebx, 00000002
:00401364   div  ebx
sommiamo 5
:00401366   add  eax, 00000005
ed ecco la lunghezza del nostro seriale! :)
:00401369   mov  dword ptr [00403251], eax
```

Bene, parte del lavoro è svolto, forse la parte più difficile... ora andiamo avanti a controllare cos'altro fa con il nome e come genera il seriale!
Riprendiamo la funzione che avevamo trovato sotto alla cattura del nome!

Questa è la funzione che viene eseguita dopo la cattura del nome inserito!

```
vengono azzerate delle variabili
:0040158D xor ecx, ecx <-- contatore dei caratteri del nome
:0040158F xor eax, eax <-- risultato dell'algoritmo finale
:00401591 mov byte ptr [00403250], 00 <-- il risultato finirà qui (00403250)
qui comincia un ciclo (il jmp in 004015AD ci riporta qui), vengono presi a due a due i caratteri del nome, in particolare la coppia identificata da ecx:
ecx = 1, primo e secondo carattere
ecx = 2, secondo e terzo carattere
e via dicendo
:00401598 movzx ebx, word ptr [ecx+00403050]
controllo del fine carattere, se lo trova esce
:0040159F cmp bl, 00
:004015A2 je 004015AF
:004015A4 cmp bh, 00
:004015A7 je 004015AF
somma il valore in formato word (quindi 2 byte) al risultato finale
:004015A9 add ax, bx
e incrementa il contatore dei caratteri
:004015AC inc ecx
fine del ciclo
:004015AD jmp 00401598
in eax c'è il risultato del nostro algoritmo, formato da due byte, ke xora l'uno con l'altro nella parte bassa del registro
:004015AF xor al, ah
come avevamo già detto il risultato in 00403250 (ricordate questo indirizzo mi raccomando...)
:004015B1 mov byte ptr [00403250], al
:004015B6 ret
```

Perfetto, fin qui tutto apposto, ora non ci rimane che capire il vero algoritmo di generazione del seriale... procediamo con l'ultima funzione che avevamo trovato, quella subito dopo la cattura del seriale...

```
azzeriamo ed inizializiamo come al solito qualche variabile
:004015B7 xor eax, eax
in ebx ci mettiamo 59 (3B)
:004015B9 mov ebx, 0000003B
ma guarda... 00403251... e cosa ci troveremo mai dentro? ;)
lo sapete già, la lunghezza del nostro seriale, quello che ci siamo calcolati prima...
comunque sommiamo al precedente 59 questo valore!
:004015BE add ebx, dword ptr [00403251]
cosa vi avevo detto poco fa? ricordate questo indirizzo... qui c'è il risultato del calcolo precedente! lo mettiamo in al (parte bassa di eax)...
:004015C4 mov al, byte ptr [00403250]
...e lo dividiamo per 4
:004015C9 mov cl, 04
:004015CB div cl
attenzione adesso... sommiamo il primo byte di ebx con il secondo byte di eax, ma nel secondo byte di eax non c'è il risultato della divisione precedente, bensì il
resto...
:004015CD add bl, ah
quindi il risultato in 00403255
:004015CF mov byte ptr [00403255], bl
prendiamo il primo carattere del nostro seriale
:004015D5 mov al, byte ptr [00403050]
e attenzione... abbiamo il nostro primo carattere... viene confrontato con il risultato precedente, e se non è uguale il ciclo finisce subito, basta seguire il
salto
:004015DA cmp al, byte ptr [00403255]
:004015E0 jne 0040161B
inizializiamo una variabile a uno (il primo carattere lo abbiamo già controllato)
:004015E2 xor ecx, ecx
:004015E4 inc ecx <-- tra l'altro qui comincia un ciclo, lo jmp in 00401619 ce lo rivela
assicuriamoci che non sia finito il seriale, in tal caso è finito anche il ciclo
:004015E5 cmp byte ptr [ecx+00403050], 00
:004015EC je 0040161E
prendiamo l'ultimo carattere che abbiamo generato...
:004015EE mov al, byte ptr [00403255]
... e xoriamolo con il valore che abbiamo generato dall'algoritmo del nome
:004015F3 xor al, byte ptr [00403250]
il valore generato dall'algoritmo del nome viene anche decrementato
:004015F9 dec byte ptr [00403250]
:004015FF xor ah, ah
dividiamo per 26 (1A)
:00401601 mov bl, 1A
:00401603 div bl
e sommiamo 65 (41) al resto della divisione
:00401605 add ah, 41
:00401608 shr ax, 08
questo è l'ennesimo carattere del seriale, quindi viene confrontato con quello del nostro seriale ed in caso non sono uguali il ciclo termina in modo errato
:0040160C cmp al, byte ptr [ecx+00403050]
:00401612 jne 0040161B
salviamo il carattere generato e ripetiamo il ciclo
:00401614 mov byte ptr [00403255], al
:00401619 jmp 004015E4
eax viene azzerato, non va bene
:0040161B xor eax, eax
:0040161D ret
eax viene inizializzato a uno, ora va bene
:0040161E mov eax, 00000001
:00401623 ret
```

Perfetto, abbiamo scovato e capito come funziona l'algoritmo di generazione del seriale... non ci resta che scrivere un piccolo programmino che prende in ingresso un nome e genera il seriale che ci serve.

Io vi posto qui le funzioni che ho generato in c++ e che sembrano funzionare:

```
long ElaboraName(char *name)
{
  int caratteri = 0;
  char parziale[2];
  unsigned char finale[2];
  finale[0] = 0;
  finale[1] = 0;
  long ritorno = 0;
  while(caratteri < strlen(name))
  {
    parziale[0] = name[caratteri];
    parziale[1] = name[caratteri+1];
    if (parziale[0] == 0 || parziale[1] == 0)
    {
      break;
    }
    finale[1] = (int)
      ((
      (finale[0]+(finale[1]*256))
        +
      (parziale[0]+(parziale[1]*256))
      ) / 256);
    finale[0] += parziale[0];
    caratteri++;
  }
  ritorno = finale[0] ^ finale[1];
  return ritorno;
}

CString GeneraKiave(long valore)
{
  long valoreaggiunto;
  long secondoparziale, terzoparziale;
  char seriale[100];
  long parziale = 59;
  valoreaggiunto = CalcolaValore();
  parziale += valoreaggiunto;
  secondoparziale = valore;
  terzoparziale = 4;
  parziale += secondoparziale % terzoparziale;
  seriale[0] = (char)parziale;
  long caratteri = 1;
  long ennesimo;
  while (caratteri <= valoreaggiunto)
  {
    ennesimo = seriale[caratteri-1];
    if (secondoparziale < 0)
      secondoparziale = 255;
    ennesimo ^= secondoparziale;
    secondoparziale--;
    ennesimo = ennesimo % 26;
    ennesimo += 65;
    seriale[caratteri] = ennesimo;
    caratteri++;
  }
  seriale[valoreaggiunto] = '\0';
  return (CString)seriale;
}

long CalcolaValore()
{
  long giorno, mese;
  SYSTEMTIME stTime;
  GetLocalTime(&stTime);
  giorno = stTime.wDay;
  mese = stTime.wMonth;
  return ((((mese ^ 13) + (giorno)) / 2) + 5);
}
```

Purtroppo per la fretta di scrivere il tutorial e il rientro al lavoro non ho avuto tempo di ottimizzare il codice e di commentarvi qualche riga di codice, ho seguito però molto il codice assembler delle funzioni che abbiamo analizzato, sperando che riusciate a fare una relazione tra le procedure in c++ e il codice assembler.
Bene... credo che questo possa essere tutto un salutone a tutti e al prossimo tutorial (sempre che venga publicato almeno questo.... :))
cia! :)

Note finali

Bhè che dire... spero di non aver commesso errori e che il tutorial sia alla portata di tutti.
Volevo fare un po' di ringraziamenti, innanzi tutto ringrazio idl3, un mio caro amico, che mi ha fatto conoscere il mondo del reverse engineering, il sito di Quequero e molte altre cose che mi ha insegnato negli anni.
Un ringraziamento a Quequero, che ci dà le basi e tutto quello di cui necessitiamo per imparare a reversare, che trova sempre qualche novità per tenere allenate le nostre menti; con i crackme, con la UIC, i riddle (bellissima sezione, a quando il riddle 12? ;P) (A oggi, NdQue :), e per aver publicato (sempre che lo faccia:P) questo piccolo tutorial.
Un ringraziamento anche a tutti quelli che hanno partecipato e continuano a farlo alla UIC di Quequero, che hanno scritto tutorial e documenti per aumentare le nostre conoscenza nel mondo del reverse engineering.
Un salutone a tutti gli amici del chan #zenigata.
E infine un saluto a tutti voi che leggerete questo tut! :)
ciaaaaaaaaaaaaaaaaa! :)

Disclaimer

eventuali danni causati al vostro computer determinati dall'uso improprio di questo tutorial. 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.

Noi reversiamo al solo scopo informativo e di miglioramento del linguaggio Assembly.
