進め！リバースエンジニアリングの道

第5回 パスワードクラッキング

文●愛甲健二

IDAPro 5.0

最初に、IDAPro 5.0がフリーウェア版として公開されました※1のでご紹介します。4.9との大きな違いは、ジャンプ系の命令を適切に解釈し、逆アセンブルウインドウにおいてもグラフィカルに分岐を表示してくれる機能です。これはかなり便利で、本記事でもそれを利用する前提で解説をしています。皆さんもぜひバージョンアップをお願いします。

エラーメッセージを探せ！

今回は前回に引き続き、付録DVD-ROM収録のcrackme_ex.exe（日付制限を回避したcrackme.exe）の解析を行います。付録DVD-ROMには連載のバックナンバーPDFも収録していますので、必要に応じて参照してください。

さて、前回ユーザー名は「WizardBible」とわかったので、次はパスワードです。解析のアプローチ方法はいくつかありますが、せっかく「パスワードが間違っています」というエラーメッセージが表示されたので、このエラーメッセージを手掛かりに処理を追いましょう。IDA Proのストリングウインドウ（Strings window）から「パスワードが間違っています」という文字列を探し出しダブルクリックすると、逆アセンブラウインドウ（IDA View-A）に対象のデータ列と参照元アドレス（sub_4019F0）が表示されます（図1）。このsub_4019F0をダブルクリックすることで、実際にこのデータ列を利用しているアセンブラ命令が表示されます（図2）。

図2のアセンブラ命令を読むと、パスワードが間違っている場合はMessageBoxAが呼び出され、正解が入力されたらloc_402769へジャンプし、call [ebp+var_208]が実行されます。そして、それらはcmp ecx, 0C8C7hの結果で分岐します。となると、ecxが0C8C7hとなるような文字列が正解パスワードになりますが、別に正解パスワードを入力せずとも、jz short loc_402769をjnz short loc_402769に変えて、ecxが0C8C7hではない場合にジャンプするように書き換えても問題なさそうです。

ではOllyDbgでcrackme_ex.exeを開き、0040274Cのjz命令をダブルクリックし、jnzに変更してください（図3）。そして、OllyDbg上でcrackme_ex.exeを実行します。

問題なさそうに見えましたが、実行すると例外発生と同時にプログラムが強制終了します。停止アドレスは実行環境によりますが、パスワードを「bbbbbbbb」と入力すると下に示したような命令コードで終了します。どうやらjzをjnzに変更するだけではダメだったようです。

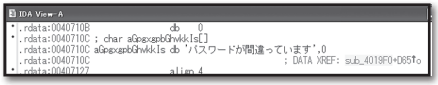

図1　実行ファイルの中に存在する文字列データ

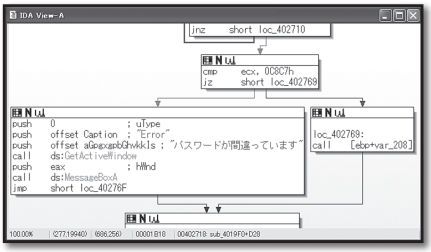

図2　「パスワードが間違っています」と表示する処理

・例外が発生した箇所

```
032D0000   19B4DD F4DD04A0   SBB DWORD PTR SS:[EBP+EBX*8+A004DDF4],ESI
032D0007   5B                POP EBX
032D0008   01AF BD4EAC5D     ADD DWORD PTR DS:[EDI+5DAC4EBD],EBP
032D000E   5C                POP ESP
```

※1　http://www.hex-rays.com/idapro/idadownfreeware.htm

図3　jz命令をjnz命令に変更する

ecxの算出方法は？

単純にjz命令を書き換えただけではダメでしたので、今度は素直にアルゴリズムを追いましょう。ecxと0C8C7hが比較されている場所から少しさかのぼり、004026FE以降をIDAProで開きます。

・IDAPro（004026FE）

```
004026FE  xor   ecx, ecx
00402700  lea   eax, [ebx+6]
00402703  mov   edx, 40h
00402708  jmp   short loc_402710
0040270A
00402710 loc_402710:
00402710  movzx edi, byte ptr[eax-5]
00402714  movzx esi, byte ptr[eax-6]
00402718  add   esi, edi
0040271A  movzx edi, byte ptr[eax-4]
0040271E  add   esi, edi
00402720  movzx edi, byte ptr[eax-3]
00402724  add   esi, edi
00402726  movzx edi, byte ptr[eax-2]
0040272A  add   esi, edi
0040272C  movzx edi, byte ptr[eax-1]
00402730  add   esi, edi
00402732  movzx edi, byte ptr[eax+1]
00402736  add   esi, edi
00402738  movzx edi, byte ptr[eax]
0040273B  add   edi, ecx
0040273D  add   eax, 8
00402740  dec   edx
00402741  lea   ecx, [edi+esi]
00402744  jnz   short loc_402710
00402746  cmp   ecx, 0C8C7h
0040274C  jz    short loc_402769
```

難しい命令はありませんので、本連載の第3回、第4回で覚えた知識を駆使し004026FE以降を読み進めましょう。004026FE以降より、ecx=0、eax=何かのアドレス、edx=40hが初期値となり、その後loc_402710へジャンプし、00402710以降繰り返し処理が行われます。

繰り返し処理では、「1バイトずつ8バイト分のデータを加算する」というルーチンを1回とし、edxが0になるまで行われ、最後にその結果ecxが0C8C7hになったか否かによって処理を分岐します。edxの初期値は40hなので、つまりは分岐が行われる前に、200h（40h×8）バイト分の値の加算値がecxに格納されます。

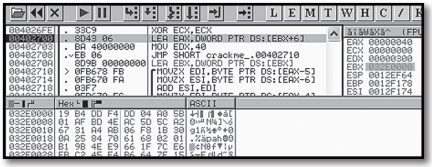

図4　左下のダンプウインドウに謎の200hバイトが存在

以上の処理の概要ですが、IDAProだけではどうしてもわかりにくい場合は、OllyDbgで004026FEにブレイクポイントを仕掛け、1命令ずつ処理を追っていきながらレジスタを確認すると、より詳細を把握しやすいと思います。

さて、こうなると今度はアドレスebxが指す200hバイトのデータ列が気になります。私の環境では004026FE時点でのebxの値は032E0000で、その先のデータ列は「19 B4 DD F4 DD ... 2A B3 4A 78 1E」となっています（図4）。パスワードは「bbbbbbbb」と入力しました。つまり、最終的なecxを0C8C7hにするためには、加算値が0C8C7hになるように謎の200hバイトを調整する必要があります。

例外の原因は？

最初に試したとおり、jzをjnzに変更するだけでは正常に実行できず、例外が発生しました。例外が発生したアドレス032D0000をもう一度確認すると、アセンブラ命令がsbbで、マシン語が「19 B4 DD F4 DD 04 A0」となっています。また、ecxの算出方法を調べていて新たに見つかった謎の200hバイトですが、このデータ列も「19 B4 DD F4 DD ...」となっています。この事実は「もし謎の200hバイトの加算値が0C8C7hになったら、謎の200hバイトそのものをマシン語として解釈、実行する」ことを意味します。

こうなると、今度は入力したパスワード「bbbbbbbb」と謎の200hバイトの関係が気になります。ecxの値がこのデータ列から算出されている以上、入力したパスワードは必ずこの200hバイトに何かしらの影響を与えるはずです。試しに入力するパスワードを「abcdefgh」に変更すると、謎の200hバイトは「0D B0 89 69 B0 ...」となります。

また、たった8文字のパスワードから200hバイトものデータ列が生成されるとは考えにくいため、入力されたパスワードによって変更される前の「もともとの200hバイト」がどこかに

あるはずです。IDAProでアドレス004026FEより前へ戻っていくと、いかにも怪しいデータ列がスタックへ連続して格納されているのが確認できます（図5）。

以上から、次の仮説を立てます。まずスタックへ積まれているデータ列がもともとの200hバイトで、これに対して入力されたパスワードを使い「何らかのアルゴリズム（アルゴリズムX）」で謎の200hバイトにデコード（もしくは復号）します。そして、このデコードされたデータ（謎の200hバイト）の加算値ecxと0C8C7hを比較し、その結果が真ならば謎の200hバイトをマシン語として実行し、偽ならエラーメッセージを表示して終了します（図6）。

仮説の確認

では、立てた仮説が本当に正しいのか、IDA ProとOllyDbgを用いて確認していきましょう。まず、アドレス004026FEからさかのぼり、関数の先頭004019F0まで戻ります。OllyDbgで004019F0にブレイクポイントをセットし、ユーザー名「WizardBible」とパスワード「bbbbbbbb」を入力すると004019F0で処理が止まります。その状態でスタックを確認すると、第1引数にパスワードとして入力された文字列が渡されており、呼び出し元は00402891となっています。00402891の少し上には、lstrcmpAでユーザー名と「WizardBible」を比較している箇所があります。

004019F0以降を読み進めていくと、最初の方は200hバイトのデータをスタックへ格納する処理が延々と続きます。そして、00402690でVirtualAllocを呼び出してメモリ空間を確保し、rep movsdを使い、その空間へでスタックに置かれた200hバイトのデータ列がコピーされます（図7）。続いてcall 00401000で00401000へ処理が飛びますが、この際引数として、パスワード「bbbbbbbb」と、Virtual Allocで確保し、200hバイトのデータ列が格納されたメモリアドレスがスタックへ格納されています。00401000のルーチンが終わると、200hバイトのデータ列は変更されていますので、どうやら00401000以降が復号処理のようです。

となると、あとは「00401000以降のアルゴリズムはいったい何か？」がわかれば全容が見えるのですが、暗号アルゴリズムは一般的にかなり複雑で、一から解読するのはかなり骨が折れます。しかし、実行時に暗号処理ならではの特徴的なデータ列が見られることが多いため、怪しいデータ列をGoogleで検索しながら見ていくことで、アルゴリズムの特定が可能です。

例えば、00401075では00409030のアドレスにある値「39 31 29 21 19 11 09 ...」を参照していますが、このデータ列をGoogleで検索すると「DES加密算法」というページがヒットします。また、004011B0で参照されている00409068の値「0E 11 0B 18 01 05 ...」を検索すると「Outlook. Safeword Cards,... How do they work」というWebサイトがいちばん最初にヒットし、このサイトにはDESアルゴリズムに関する解説が書かれています。

つまり、00401000のルーチンはDES関係のアルゴリズムであると簡単に特定できます。アルゴリズムさえ特定できれば、わざわざアセンブルコードは読まずともWebからソースコードを探してそのまま利用できます。

これで、図6の仮説のとおりに処理が行われていることが確認でき、かつ、アルゴリズムXはDESのようだとわかりました。

パスワードクラッキング手法

さて、仮説が正しいことはわかりましたが、

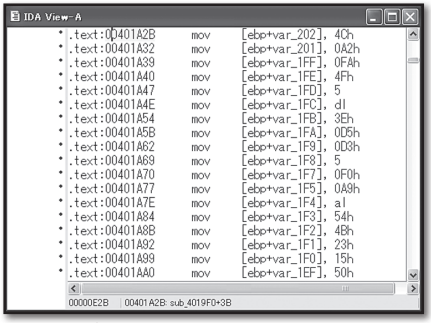

図5　怪しいデータ列がスタックへ格納されている

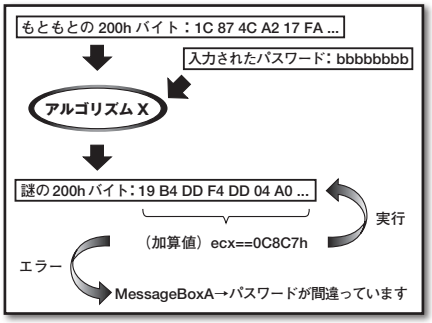

図6　推測できる処理フロー

そもそもの目的は本当のパスワードを特定することです。ここで少し考えてみましょう。

謎の200hバイトは平文（実行可能なマシン語）、もともとの200hバイトは暗号文、入力されるパスワードは鍵、アルゴリズムはDES、そして0C8C7hと比較する処理は、正常なマシン語に復号されたことを確認するチェックサム計算（のようなもの）とわかりました。では、このような状態、つまり、暗号文と平文のチェックサムとアルゴリズムがわかっている状態で、鍵となる復号パスワードを求めることは可能でしょうか？

アルゴリズムにもよりますが、残念ながら一般的に使われている暗号アルゴリズムを用いられれば、この条件下で鍵となるパスワードを算出することはまず不可能です。となると、残る手段はパスワードクラッキングしかありません。

パスワードクラッキングには大きく分けて、「類推」「辞書」「総当たり」の3つの方法があります。

類推とは、ターゲットの個人情報などからパスワードを類推する方法で、例えば管理者の好きな映画、誕生日、出身地などを推測し、1つずつ試すことです。技術的な壁がなく誰でも簡単に試せますが、重要なシステムにおいて使われていることは稀で、基本的には成功しない前提で試されます。

次に辞書を使った攻撃です。これは、何百万行にも及ぶ単語（ワード）が記述されたテキストファイルを用意し、1行ずつ順番に試していく手法です。単語の意味が書かれてある一般的な辞書とは異なり、よく使われる比較的出現頻度の高い単語が1行ずつ載っています。総当たりでは現実的な時間内で解析できない（すべてのパターンを試すのに年単位の時間がかかる）場合に使われます。

最後に総当たり攻撃ですが、これはその名のとおり範囲内で再現できるすべてのパターンを試す方法です。パスワードが大文字英字6文字だとわかっていれば、AAAAAA～ZZZZZZの範囲内のすべてのパターンを1つずつ試していけばいずれは解答にたどり着けますが、すべてのパターンを試すため、現実的な時間内には終了しないことが多々あります。

パスワードを破ってみよう

これでcrackme_ex.exeのパスワードを解析するためのすべての知識は揃いました。ここから先は第6回目までの宿題とさせていただきたいと思います。どのような方法を用いても構いませんので、見事パスワードを解析してみてください。

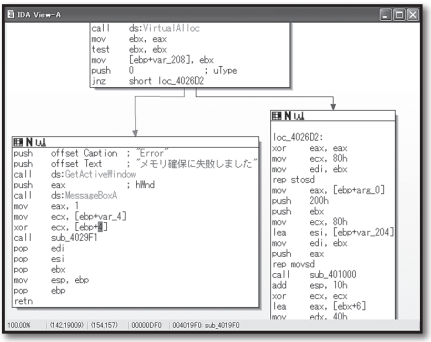

図7　VirtualAlloc呼び出し後の処理

基本的な処理の流れは図6のとおりで、アルゴリズムXはDESです。DESのソースコードはWebで探せばいくつか見つかると思いますが、OpenSSLライブラリ※2を用いる方が簡単かもしれません。また、パスワードクラックを行うプログラムを自作するのではなく、crackme_ex.exe自体を少し改造してうまくクラックルーチンを追加してもよいかもしれません。

あと、パスワードクラックの手法としては、8文字以上のテキストを総当たりするのはかなり時間がかかりますので、今回は辞書攻撃を選ぶべきでしょう。そのための辞書もDVD-ROMに収録していますのでぜひ使ってください。

最後にWindows7を使っている方へ。これは私の確認ミスなのですが、crackme_ex.exe（crackme.exe）は、Windows7上では正しく動作しません。正確には、間違ったパスワードが入力されているうちは問題なく動作するのですが、正しいパスワードが入力された場合に「正解」を示すメッセージボックスが表示されず、プログラムが強制終了してしまいます。よって、正しいパスワードがわかったら、XPモード上で実行するか、WindowsVista以前のバージョンで動作をお試しください。本当に申し訳ありません。

さて、今回はcrackme.exeのパスワード認証部分の解析を行いましたが、いかがだったでしょうか。次回は、実際にパスワードクラックするプログラムを作成し、本当のパスワードを見つけたいと思います。では、またお会いしましょう。

※2　OpenSSLプロジェクト公式サイト　http://www.openssl.org/