OllyDbgによる実践ｿﾌﾄｸﾗｯｸ 初級編 （その1）

1. ｿﾌﾄﾌﾟﾛﾃｸﾄの種類

概ね，ｿﾌﾄﾌﾟﾛﾃｸﾄには次の3種類の型があります．

1) ｼﾘｱﾙﾅﾝﾊﾞｰ認証型
2) 試用期限型
3) ID･ﾊﾟｽﾜｰﾄﾞ型

2. ｿﾌﾄﾌﾟﾛﾃｸﾄｸﾗｯｸ概要

1) ｼﾘｱﾙﾅﾝﾊﾞｰ認証型

比較的に容易なもので，ﾁｪｯｸﾙｰﾁﾝが解読できれば簡単にCrackできます．
この構造･動作にはいくつものﾋﾝﾄとなる要素が含まれていて，その要素に
着目することでCrackの端緒を見いだします．

＊ 一般的なｼﾘｱﾙﾅﾝﾊﾞｰのﾁｪｯｸﾙｰﾁﾝ

a. ﾕｰｻﾞｰが入力したｼﾘｱﾙﾅﾝﾊﾞｰ(文字列)を取得する
b. ｼﾘｱﾙﾅﾝﾊﾞｰの妥当性をﾁｪｯｸする(ここが特定すべき個所)
c. ﾁｪｯｸの結果に応じてﾒｯｾｰｼﾞを表示する

ｴﾃﾞｨｯﾄﾎﾞｯｸｽから文字列を取得するにはWindowsが提供するAPI関数を
用いています．

＊ 文字列取得：API(Application Program Interface)関数
GetWindowText，GetDlgItemText，GetDlgItemInt の3種類です

これらのAPI関数が呼び出される瞬間を捉えることにより，ﾁｪｯｸﾙｰﾁﾝの個所を
特定します．これを実現するのがﾌﾞﾚｰｸﾎﾟｲﾝﾄの設定です．
OllyDbgでは実行ﾌｧｲﾙが使用しているAPI関数の一覧を表示することが出来ます．

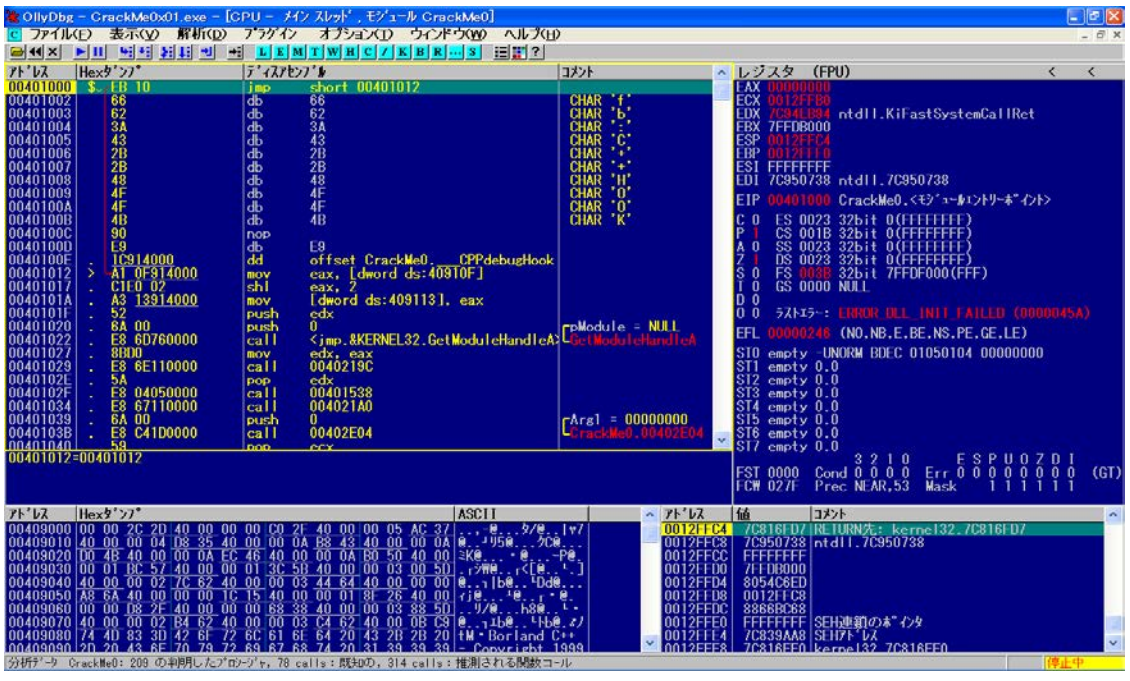
具体的な操作は後述します．

＊ ﾁｪｯｸﾙｰﾁﾝのﾄﾚｰｽの第一歩

a. OllyDbgでｿﾌﾄの実行ﾌｧｲﾙを開きます
b. F9ｷｰを押して，ｼﾘｱﾙﾅﾝﾊﾞｰ入力のｴﾃﾞｨｯﾄﾎﾞｯｸｽが出たら，ﾌｪｲｸｼﾘｱﾙﾅﾝﾊﾞｰ
を入れておきます([OK]，[登録]等のﾎﾞﾀﾝは押しません)
c. 文字列取得のAPI関数にﾌﾞﾚｰｸﾎﾟｲﾝﾄを設定します(実例を後述します)
d. ここで，[OK]，[登録]等のﾎﾞﾀﾝを押します
e. API関数が実行される直前で停止(ﾌﾞﾚｰｸ)します
f. この先にﾁｪｯｸﾙｰﾁﾝが存在するはずです
g. これからは，1命令ずつ実行(F8ｷｰを押す)して精査を行っていきます

ｱﾌﾟﾘｹｰｼｮﾝによっては簡単に解明するものもあれば，この先複雑なﾙｰﾁﾝを
仕掛けているものもあります．俗に言う『倍直』は実行ﾌｧｲﾙにｼﾘｱﾙﾅﾝﾊﾞｰを
直接書き込んでいるもので，いとも簡単にCrackできます．

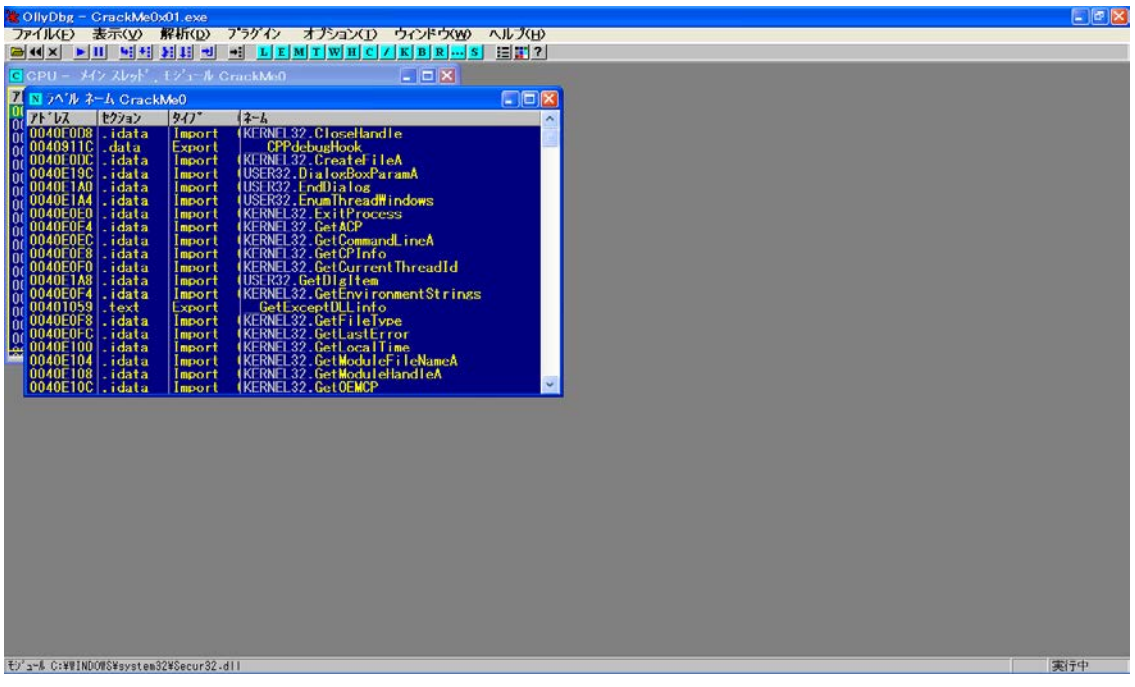
それでは"CrackMe0x01．exe"を解析してみましょう．



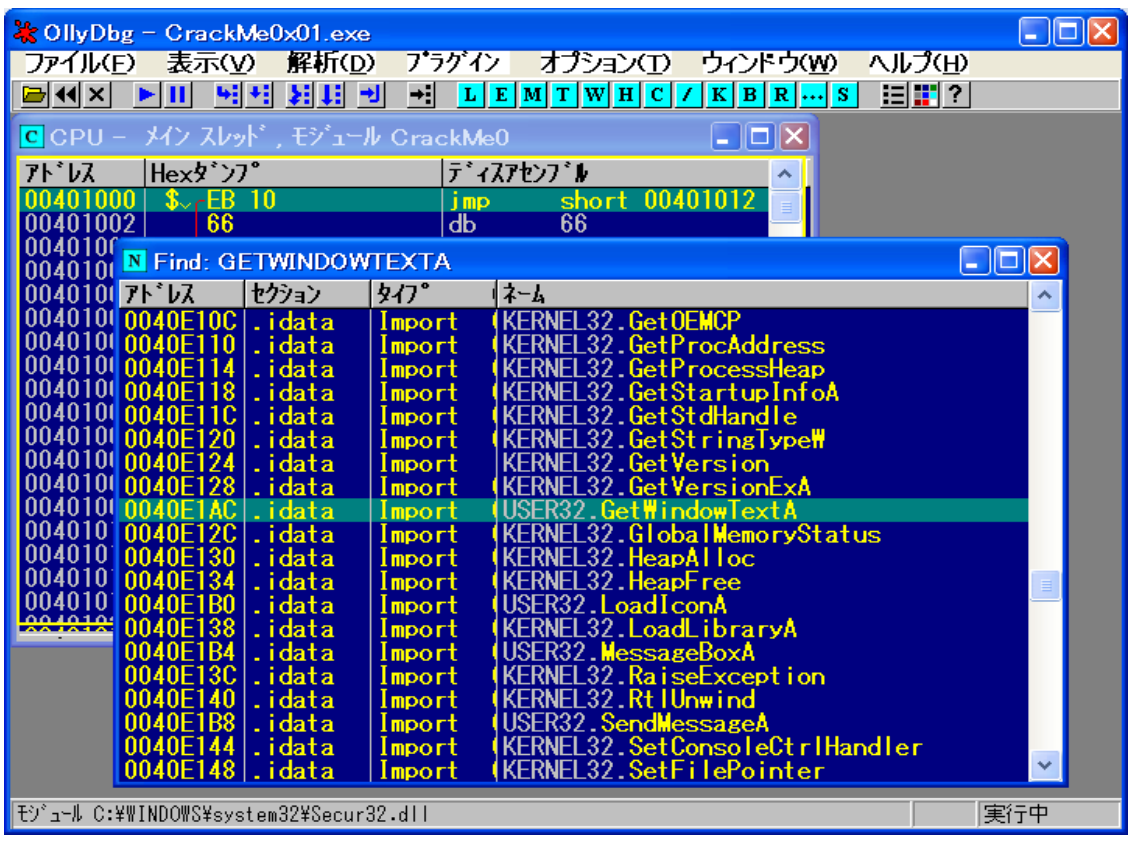
> OllyDbgで ｿﾌﾄの"CrackMe0x01．exe"を開きます．

F9ｷｰを1回押します．

> ｼﾘｱﾙﾅﾝﾊﾞｰ入力のｴﾃﾞｨｯﾄﾎﾞｯｸｽが出たら，
ﾌｪｲｸｼﾘｱﾙﾅﾝﾊﾞｰを入れておきます．
([OK]，[登録]等のﾎﾞﾀﾝは押しません)



> Ctrl+Nキーを押します.



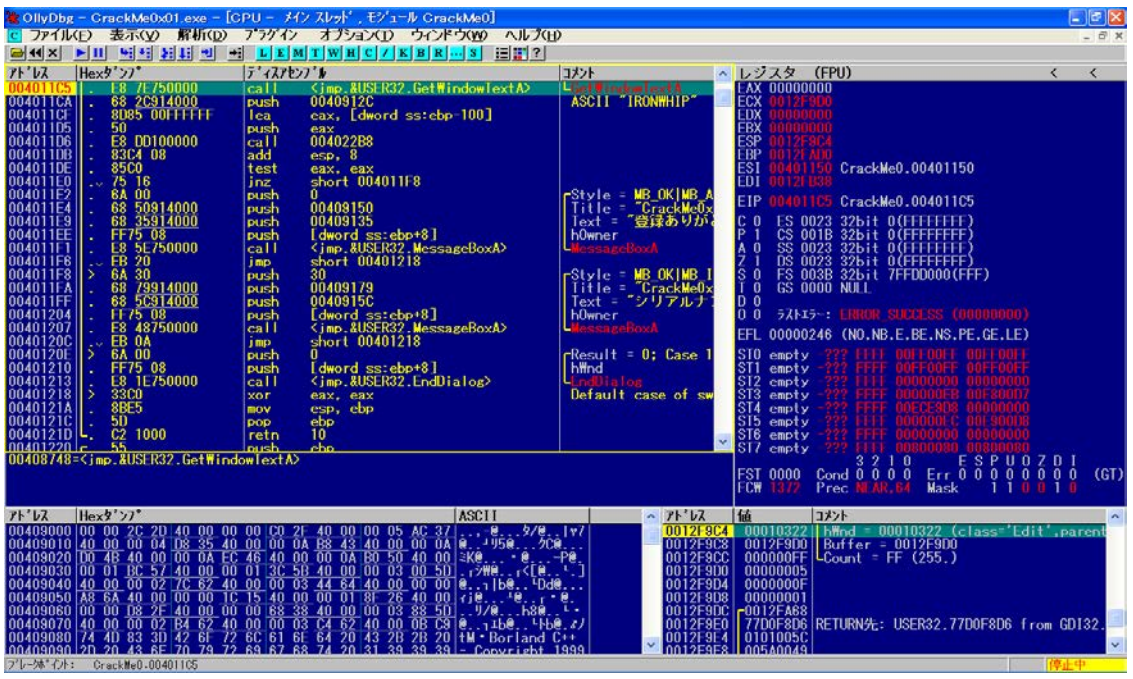
> ここでキーボードで「GETWINDOWTEXTA」とタイプします.

GetWindowTextA を選択した状態で右クリック→
[全ての参照にブレークポイントをセット]を選択すると,
GetWindowTextA を呼び出している命令行の全てに
ブレークポイントが設定されます.

この時点で[Find]画面は消してもOK.

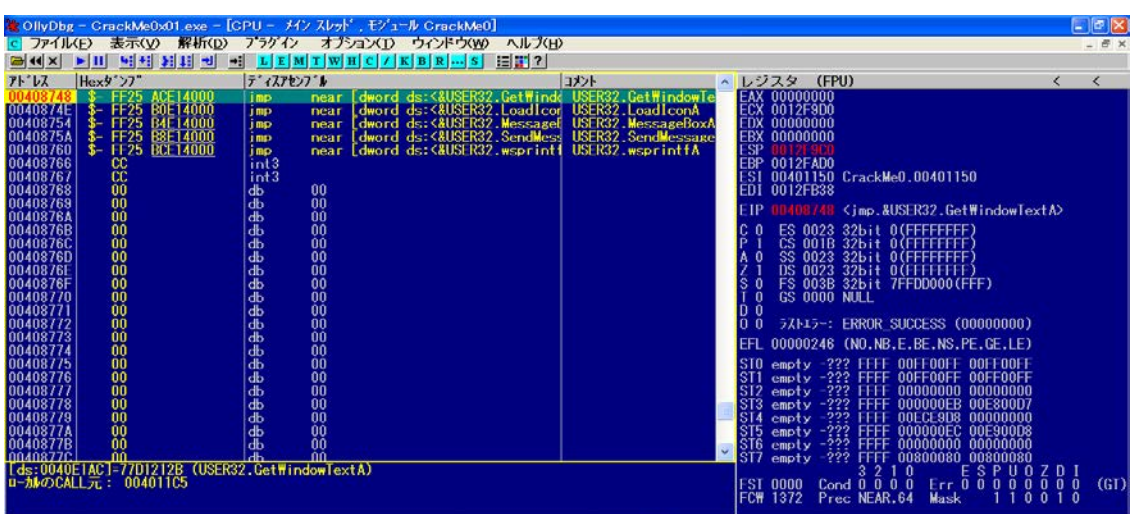
[CPU]画面を拡大します.

【注記】:API関数名の末尾の"A"はANSIコード形式の
文字列に対応した関数を表します.
因みに, "W"はUnicode形式を表します.



> API関数が実行される直前で停止(ブレーク)しました.

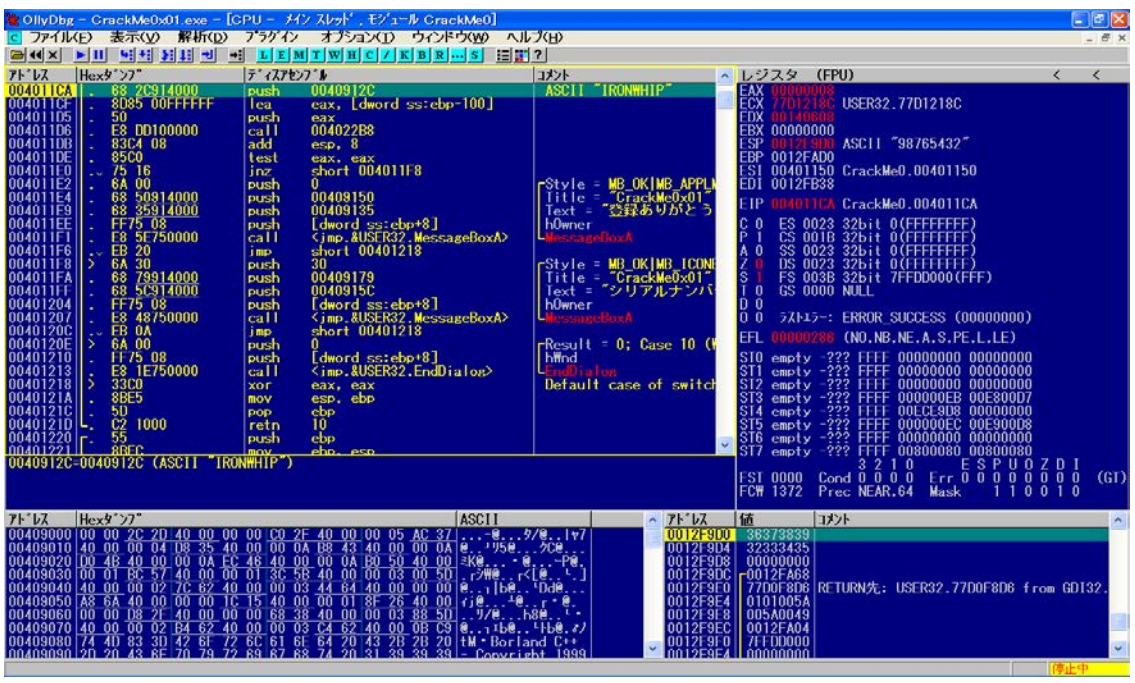
この call命令でタイプした"98765432"の文字列を取得して
いるようです.



> F8キーを1回押します.

次のAPI関数が実行される直前で再度停止(ブレーク)
しました.

call命令の内部でまた停止したようです.



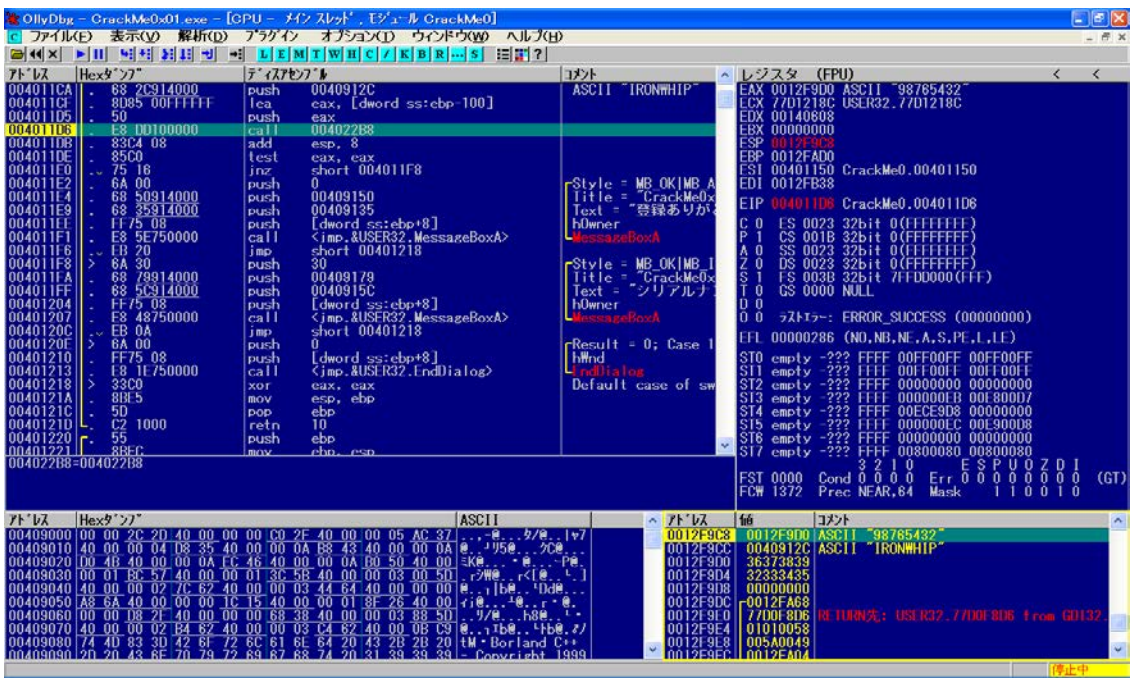
> F9キーを1回押します.

call命令を出ました.

```
004011C5  E8 7E750000       call     <jmp. &USER32. GetWindowTex>
004011CA  68 2C914000       push     0040912C     ;ASCII "IRONWHIP"
004011CF  8D85 00FFFFFF     lea      eax, [dword ss:ebp-100]
004011D5  50                push     eax
004011D6  E8 DD100000       call     004022B8
```

004011CA の push命令で ﾒﾓﾘｰ0040912C に文字列"IRONWHIP"を書き込んでいます.
004011CF の lea命令(実効ｱﾄﾞﾚｽの取得)で文字列"98765432"のｱﾄﾞﾚｽを eaxﾚｼﾞｽﾀに書き込んでいます
004011D5 の push命令でﾚｼﾞｽﾀ eax に文字列"98765432"を書き込んでいます.
004011D6 の call命令で比較をしているようです.

ここまで解析すれば, ﾁｪｯｸﾙｰﾁﾝを回避するﾊﾟｯﾁも作成できるかもしれません.

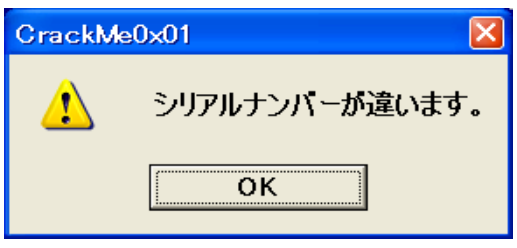


> F8キーを3回押して 00411D6 まで進めてみます.

ｽﾀｯｸｳｨﾝﾄﾞｳに注目します.

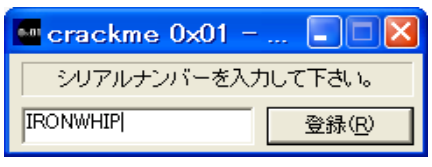
0012F9C8　0012F9D0　ASCII "98765432"
0012F9CC　0040912C　ASCII "IRONWHIP"

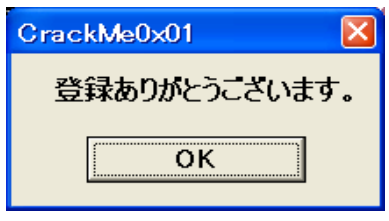
"IRONWHIP"がｼﾘｱﾙﾅﾝﾊﾞｰのようです.



> F9キーを押してみます.

間違いないようです.





> OllyDbg を終了します.

ｱﾌﾟﾘｹｰｼｮﾝを再起動して正しいｼﾘｱﾙﾅﾝﾊﾞｰを入力します.

的中でした!!

次回は　　2) 試用期限型　　です

印刷時のﾍﾟｰｼﾞﾚｲｱｳﾄは考慮していませんので悪しからず.

以上

2) 試用期限型

概ね，その試用期限の取り方で2種類があります．

+ 初回起動日から一定日数まで試用可能なもの
+ 試用期限を過ぎると試用できなくなるもの

＊ 一般的な試用期限のﾁｪｯｸﾙｰﾁﾝ

a. 現在日時を取得する
b. 試用期限の妥当性をﾁｪｯｸする(ここが特定すべき個所)
c. ﾁｪｯｸの結果に応じてﾒｯｾｰｼﾞを表示する

現在日時を取得するにはWindowsが提供するAPI関数を
用いています．

＊ 現在日時取得：API(Application Program Interface)関数

GetLocalTime，GetSystemTime の2種類です

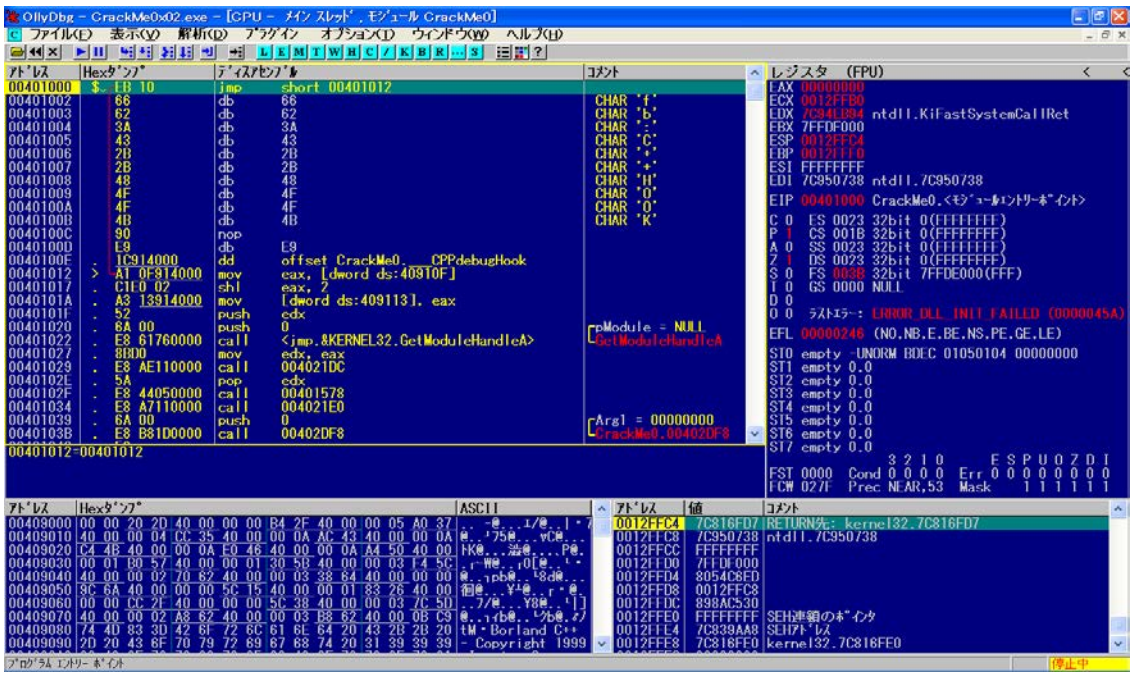
これらのAPI関数が呼び出された後のﾁｪｯｸﾙｰﾁﾝの個所を特定します．

＊ ﾁｪｯｸﾙｰﾁﾝのﾄﾚｰｽの第一歩

a. OllyDbgでｿﾌﾄの実行ﾌｧｲﾙを開きます
b. 現在日時取得のAPI関数にﾌﾞﾚｰｸﾎﾟｲﾝﾄを設定します
c. ここで，[OK]，[登録]等のﾎﾞﾀﾝを押します
d. API関数が実行される直前で停止(ﾌﾞﾚｰｸ)します
e. この先にﾁｪｯｸﾙｰﾁﾝが存在するはずです

ｱﾌﾟﾘｹｰｼｮﾝによってはPCの日時設定を操作すると使用期限内で試用可能な
ものもありますが，これはPCにいろんな障害を起こす恐れがあります．
ただし，ｲﾝﾀｰﾈｯﾄｻﾊﾞｰから現在日時を取得している場合は通用しません．

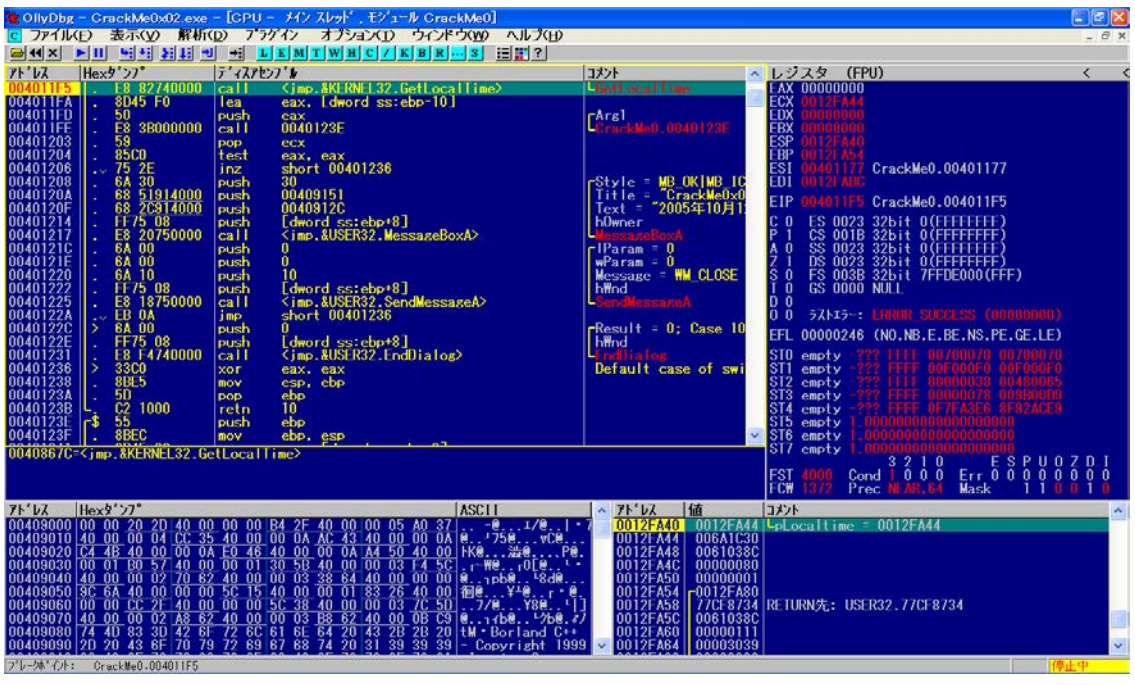
それでは"CrackMe0x02．exe"を解析してみましょう．



＞ OllyDbgで ｿﾌﾄの"CrackMe0x01．exe"を開きます．

試用期限のﾁｪｯｸは起動直後に行われるので，
F9ｷｰを押す前にAPI関数(GetLocalTime)に
ﾌﾞﾚｰｸﾎﾟｲﾝﾄを設定します．

F9ｷｰを1回押します．

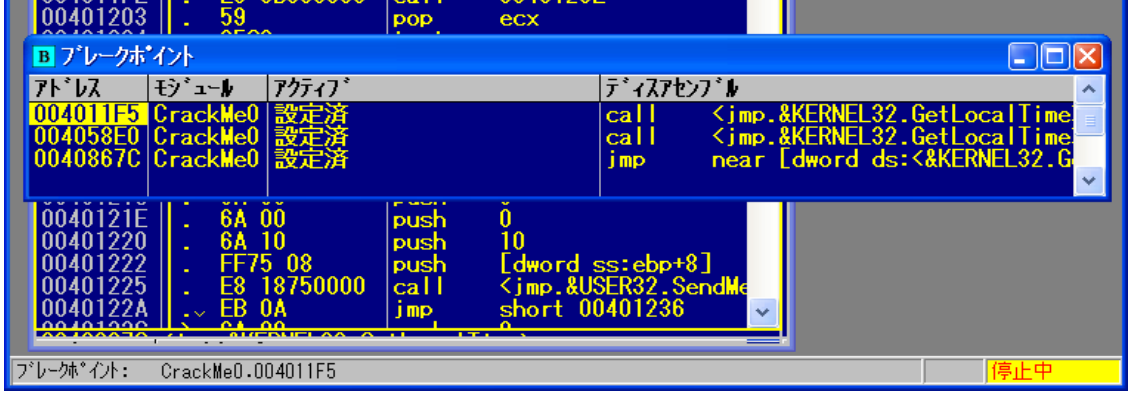


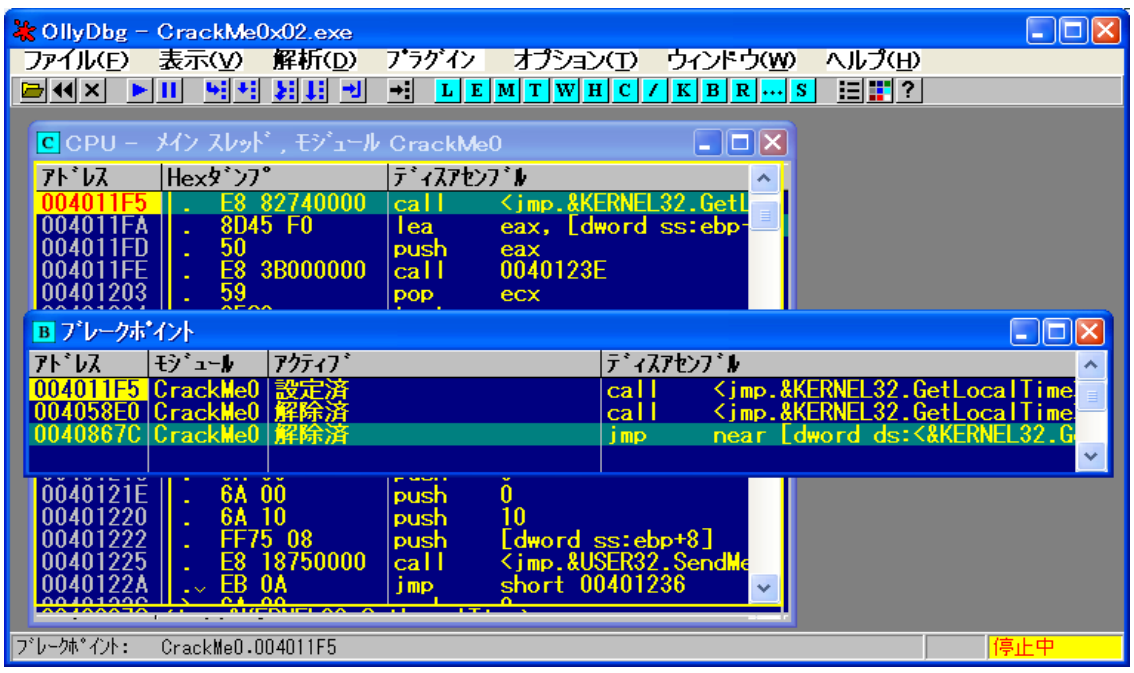
＞ "004011F5"でﾌﾞﾚｰｸしました．

この "GetLocalTime"を実行する前に，不要な
ﾌﾞﾚｰｸﾎﾟｲﾝﾄを一時的に無効化すると，思わぬ個所で
ﾌﾞﾚｰｸしないので，ｽﾑｰｽﾞなﾄﾚｰｽが出来るようになります．

＞ Alt+Bｷｰを押します．

ﾊｲﾗｲﾄしている"004011F5"以外を一時的に無効化します．





> スペースキーで解除済みにします.

(DELキーを使用するとブレークポイントが削除されます)

[B]画面は消してもOK.

[C]画面を拡大します.

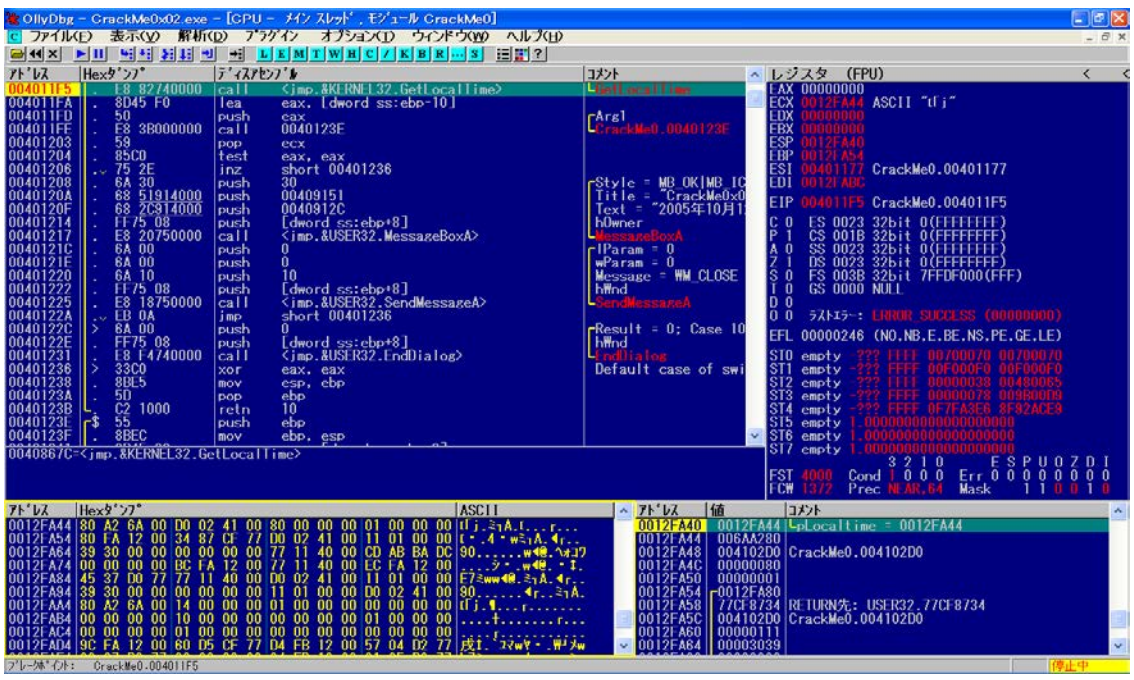
```
004011F4   51              push    ecx                ; pLocaltime
004011F5   E8 82740000     call    <jmp.&KERNEL32.GetLocalTime>
004011FA   8D45 F0         lea     eax, [dword ss:ebp-10]
```

> ディスアセンブル欄
ラベル pLocaltime の内容を引数として call命令に渡しています.

```
0012FA40   0012FA44  ¥pLocaltime = 0012FA44
```

(この値は環境によって変わります)

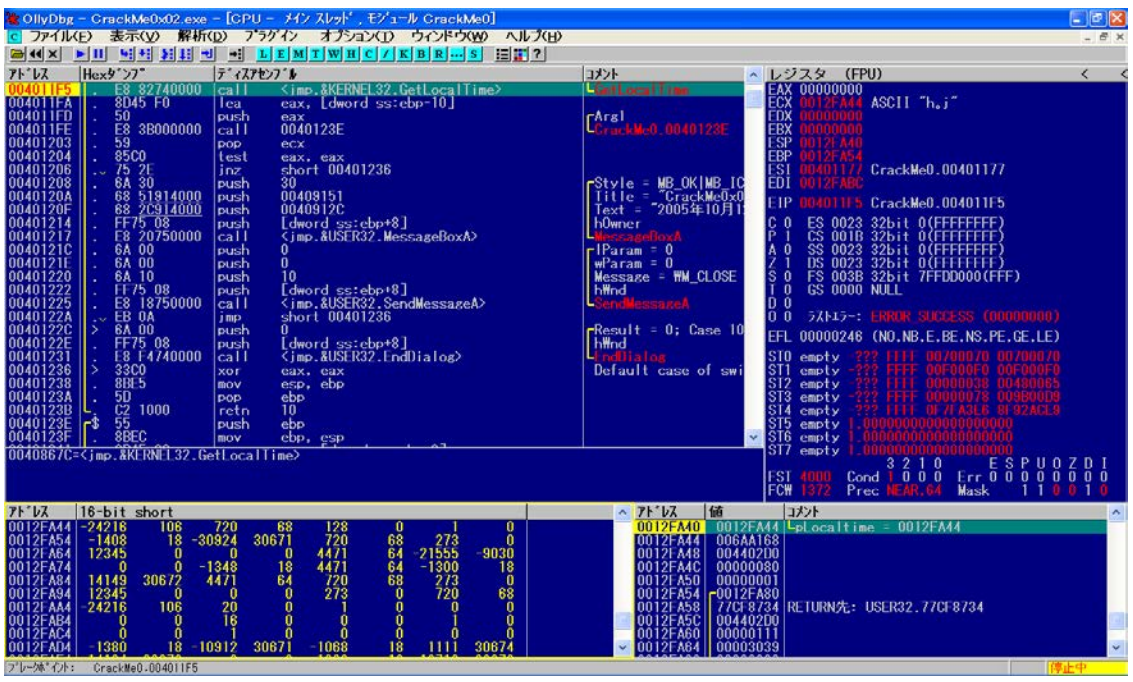
> スタックウィンドウを見れば, その引数が "0012FA44"であることが判ります. "GetLocalTime"が実行されると現在日時がこの領域に格納されます.



> この領域をダンプウィンドウに表示させます.

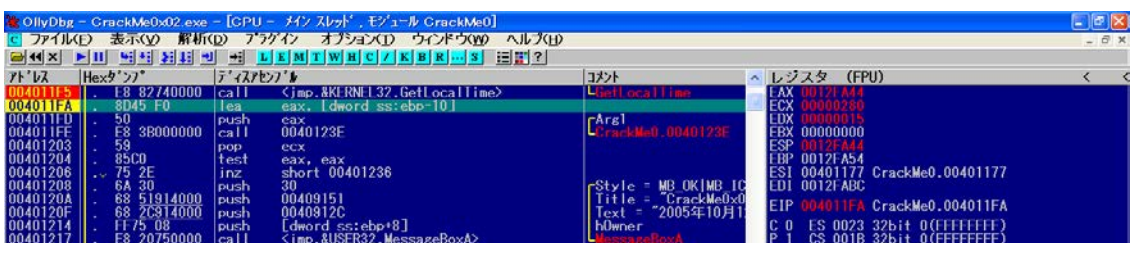
スタックウィンドウの "0012FA44"の個所を右クリックします.

[ダンプ画面へ]をクリックします.

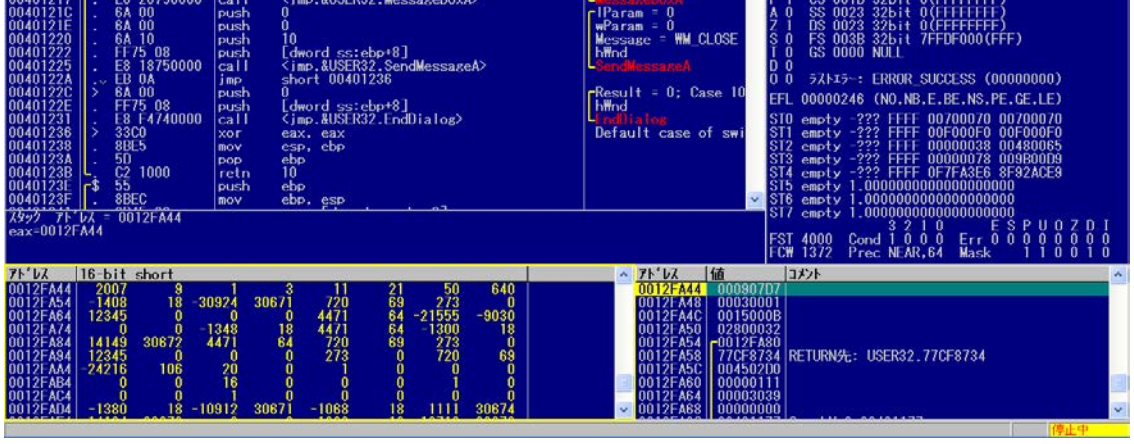


> ダンプウィンドウが Hex/ASCII(16バイト)表示になっていたので, Short/符号あり 10進表示に変更します.

ダンプウィンドウで右クリック→[Short]→[符号あり 10進]をクリックします.



> F8キーを押して, "004011F5" のcall命令を実行します.



> ﾀﾞﾝﾌﾟｳｨﾝﾄﾞｳから

0012FA44	2007	9	1	3	11	21	50	643
	↓	↓	↓	↓	↓	↓	↓	↓
	年	月	曜日 （月曜日）	日	時	分	秒	ミリ秒

この配列は，C言語のSYSTEMTIME構造体の構成で決められています．

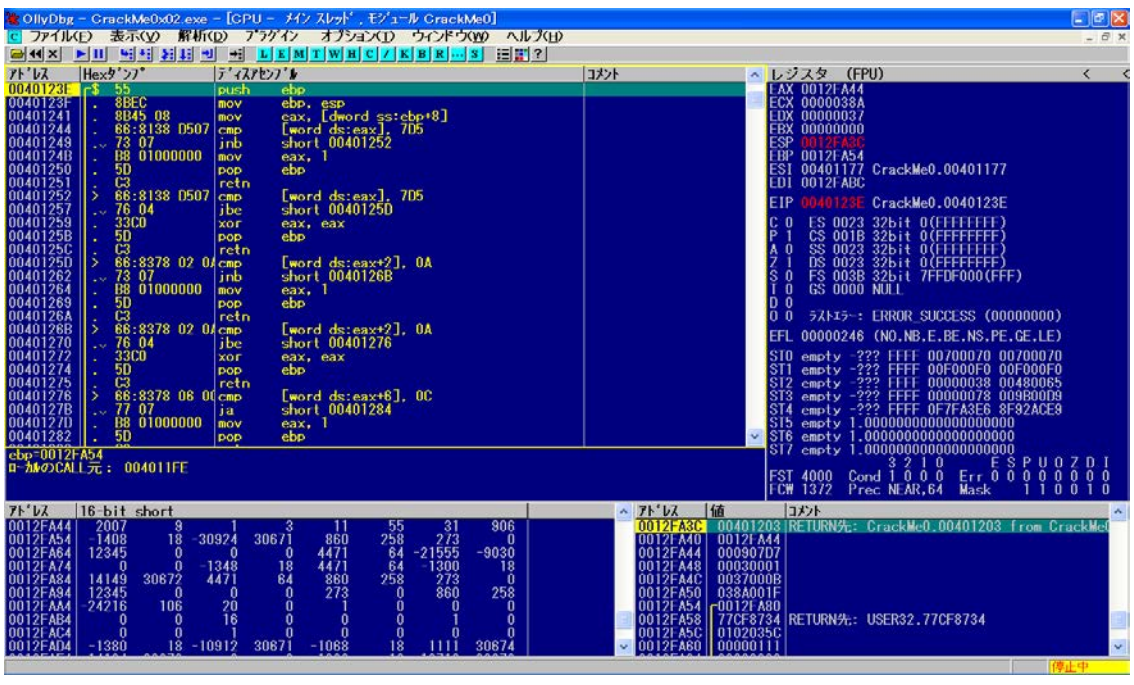
```
004011F5   E8 82740000    call    <jmp.&KERNEL32.GetLocalTime>
004011FA   8D45 F0        lea     eax, [dword ss:ebp-10]
004011FD   50             push    eax
004011FE   E8 3B000000    call    0040123E
00401203   59             pop     ecx
00401204   85C0           test    eax, eax
00401206   75 2E          jnz     short 00401236
00401208   6A 30          push    30
0040120A   68 51914000    push    00409151
0040120F   68 2C914000    push    0040912C
                ;Text = "2005年10月12日以降は起動できません。"
00401214   FF75 08        push    [dword ss:ebp+8]
00401217   E8 20750000    call    <jmp.&USER32.MessageBoxA>
```

> ﾁｪｯｸﾙｰﾁﾝのﾄﾚｰｽを進めます．

call命令直後に比較･分岐命令がある場合はcall命令の内部にﾁｪｯｸﾙｰﾁﾝが潜んでいる可能性が高く，次のcall命令 "call 0040123E"の内部を精査する必要があります．

そこで，F8ｷｰを2回押します．

"004011FE"まで進みました．



> この call命令を調べるために，詳細ｽﾃｯﾌﾟ実行します．

F7ｷｰ(詳細ｽﾃｯﾌﾟ実行)を押します．

"0040123E"に制御が移りました．

> "call 0040123E"の内部を精査します．

```
現在年のﾁｪｯｸ(1)
0040123E   55             push    ebp
0040123F   8BEC           mov     ebp, esp
00401241   8B45 08        mov     eax, [dword ss:ebp+8]
00401244   66:8138 D507   cmp     [word ds:eax], 7D5
00401249   73 07          jnb     short 00401252
0040124B   B8 01000000    mov     eax, 1
00401250   5D             pop     ebp
00401251   C3             retn
現在年のﾁｪｯｸ(2)
00401252   66:8138 D507   cmp     [word ds:eax], 7D5
00401257   76 04          jbe     short 0040125D
00401259   33C0           xor     eax, eax
0040125B   5D             pop     ebp
0040125C   C3             retn
現在月のﾁｪｯｸ(1)
0040125D   66:8378 02 0A  cmp     [word ds:eax+2], 0A
00401262   73 07          jnb     short 0040126B
00401264   B8 01000000    mov     eax, 1
00401269   5D             pop     ebp
0040126A   C3             retn
現在月のﾁｪｯｸ(2)
0040126B   66:8378 02 0A  cmp     [word ds:eax+2], 0A
00401270   76 04          jbe     short 00401276
00401272   33C0           xor     eax, eax
00401274   5D             pop     ebp
00401275   C3             retn
```

> 現在年のﾁｪｯｸ(1)

+ 現在年＞2005(7D5h)の時，
+ "00401252"にｼﾞｬﾝﾌﾟします．
+ 現在年≦2005(7D5h)の時，戻り値 1 が返ります．

+ (試用期限内)

> 現在年のﾁｪｯｸ(2)

+ 現在年≦2005(7D5h)の時，
+ "0040125D"にｼﾞｬﾝﾌﾟします．
+ 現在年＞2005(7D5h)の時，戻り値 0 が返ります．

+ (試用期限切れ)

> 現在月のﾁｪｯｸ(1)

+ 現在月＞10(0Ah)の時，
+ "0040126B"にｼﾞｬﾝﾌﾟします．
+ 現在年≦10(0Ah)の時，戻り値 1 が返ります．

+ (試用期限内)

> 現在月のﾁｪｯｸ(2)

+ 現在月≦10(0Ah)の時，
+ "00401276"にｼﾞｬﾝﾌﾟします．
+ 現在月＞10(0Ah)の時，戻り値 0 が返ります．

+ (試用期限切れ)

```
現在日のﾁｪｯｸ(1)
00401276   66:8378 06 0C cmp     [word ds:eax+6], 0C
0040127B    77 07           ja      short 00401284
0040127D    B8 01000000     mov     eax, 1
00401282    5D              pop     ebp
00401283    C3              retn
現在日のﾁｪｯｸ(2)
00401284    33C0            xor     eax, eax
00401286    5D              pop     ebp
00401287    C3              retn
```

> 現在日のﾁｪｯｸ(1)
+ 現在日＞12(0Ch)の時,
+ "00401284"にｼﾞｬﾝﾌﾟします.
+ 現在日≦12(0Ch)の時, 戻り値 1 が返ります.

+ (試用期限内)
> 現在日のﾁｪｯｸ(2)
+ 現在月＞10(0Ah)の時, 戻り値 0 が返ります.

+ (試用期限切れ)

```
現在年のﾁｪｯｸ(1)
0040123E   55              push    ebp
0040123F   8BEC            mov     ebp, esp
00401241   8B45 08         mov     eax, [dword ss:ebp+8]
00401244   66:8138 D507    cmp     [word ds:eax], 7D5
00401249   73 07           jnb     short 00401252
0040124B   B8 01000000     mov     eax, 1
00401250   5D              pop     ebp
00401251   C3              retn
```

> ｸﾗｯｸﾊﾟｯﾁの作成(1)

⇒ ﾁｪｯｸﾙｰﾁﾝの最初の比較命令のｵﾍﾟﾗﾝﾄﾞを書き換えると
試用期限が延長されるようです.

7D5h→0FFFFh に変更します.

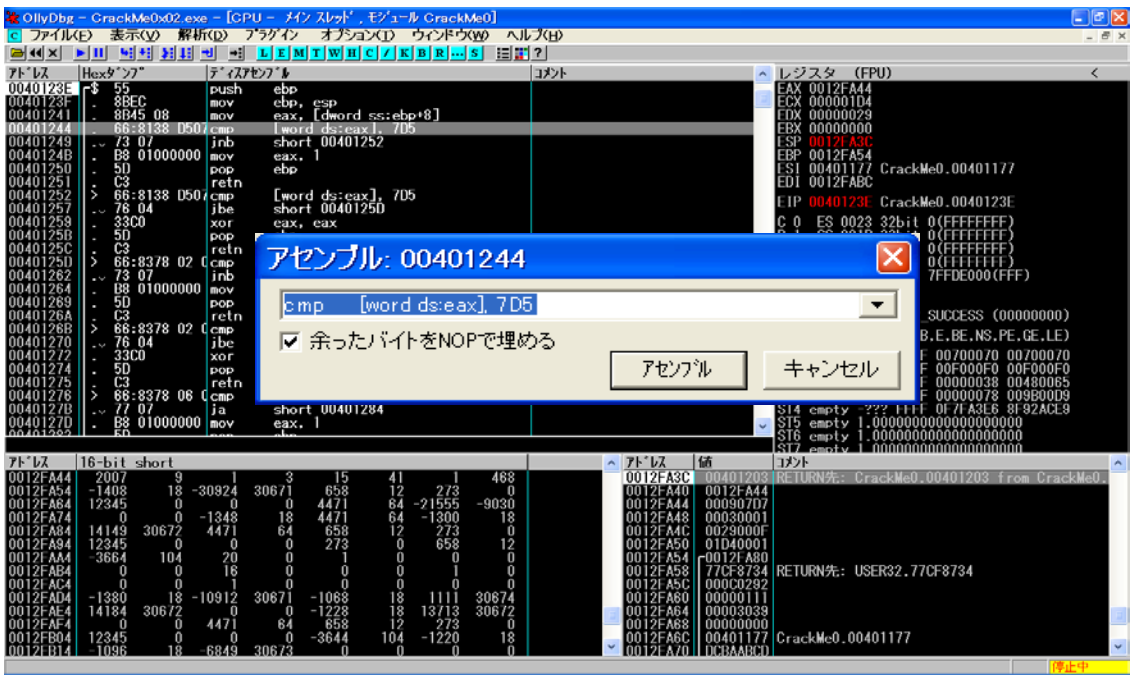
```
現在年のﾁｪｯｸ(1)
0040123E   55              push    ebp
0040123F   8BEC            mov     ebp, esp
00401241   8B45 08         mov     eax, [dword ss:ebp+8]
00401244   66:8138 D507    cmp     [word ds:eax], 7D5
00401249   73 07           jnb     short 00401252
0040124B   B8 01000000     mov     eax, 1
00401250   5D              pop     ebp
00401251   C3              retn
```

> ｸﾗｯｸﾊﾟｯﾁの作成(2)

⇒ ﾁｪｯｸﾙｰﾁﾝの最初の分岐命令のｵﾍﾟﾗﾝﾄﾞを書き換えると
常に試用期限内と判断されるようです.

00401252→0040124B に変更します.

> では, OllyDbgでの実際の書き換えを行います.



> ｸﾗｯｸﾊﾟｯﾁの作成(1)

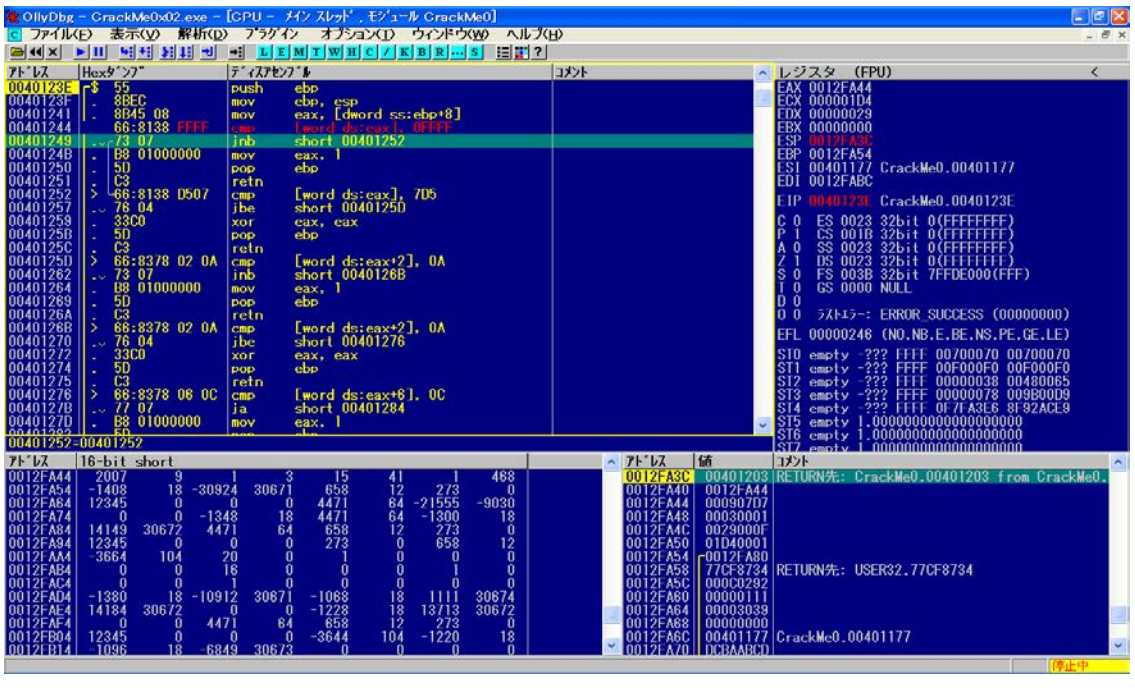
書き換えたい行 "00401244"をﾀﾞﾌﾞﾙｸﾘｯｸします.

(同行をｸﾘｯｸしてSpaceｷｰを押してもOK)

> 7D5→0FFFF に変更します.

16進数の表記で最初がA～Fで始まるときは"0"を頭に
付けます.
(末尾の"h"は不要)

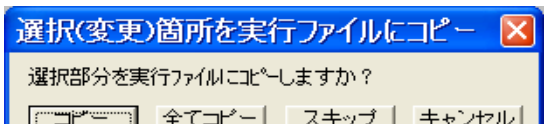
[ｱｾﾝﾌﾞﾙ]ﾎﾞﾀﾝを押します.



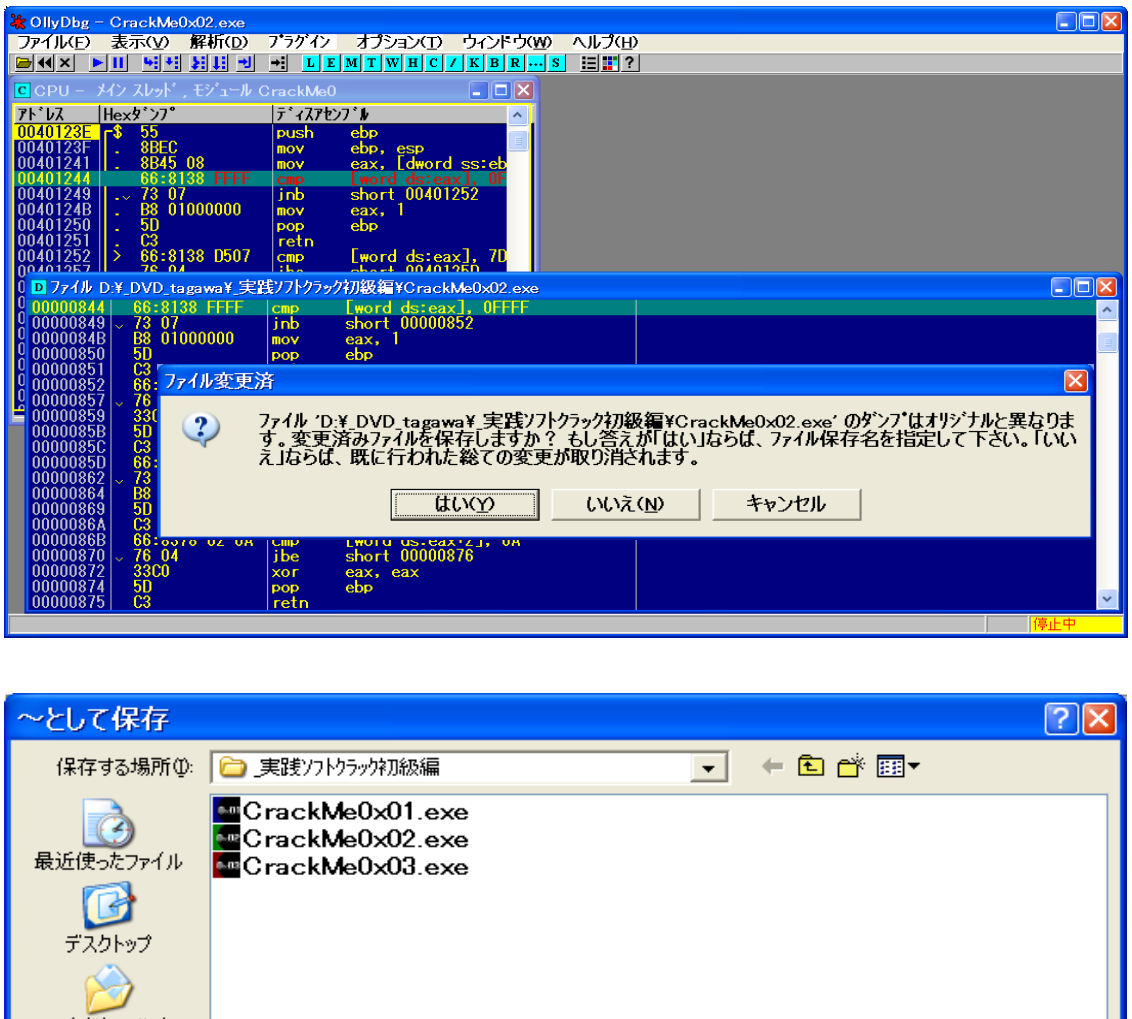
> ﾒﾓﾘｰ内の書き換えが完了しました.

ﾊｰﾄﾞﾃﾞｨｽｸに名前を付けて保存します.

逆ｱｾﾝﾌﾞﾙｳｨﾝﾄﾞｳで右ｸﾘｯｸ
[実行ﾌｧｲﾙにｺﾋﾟｰ]→[全ての変更箇所]をｸﾘｯｸします.



> 出てきたﾀﾞｲｱﾛｸﾞの[全てのｺﾋﾟｰ]をｸﾘｯｸします.

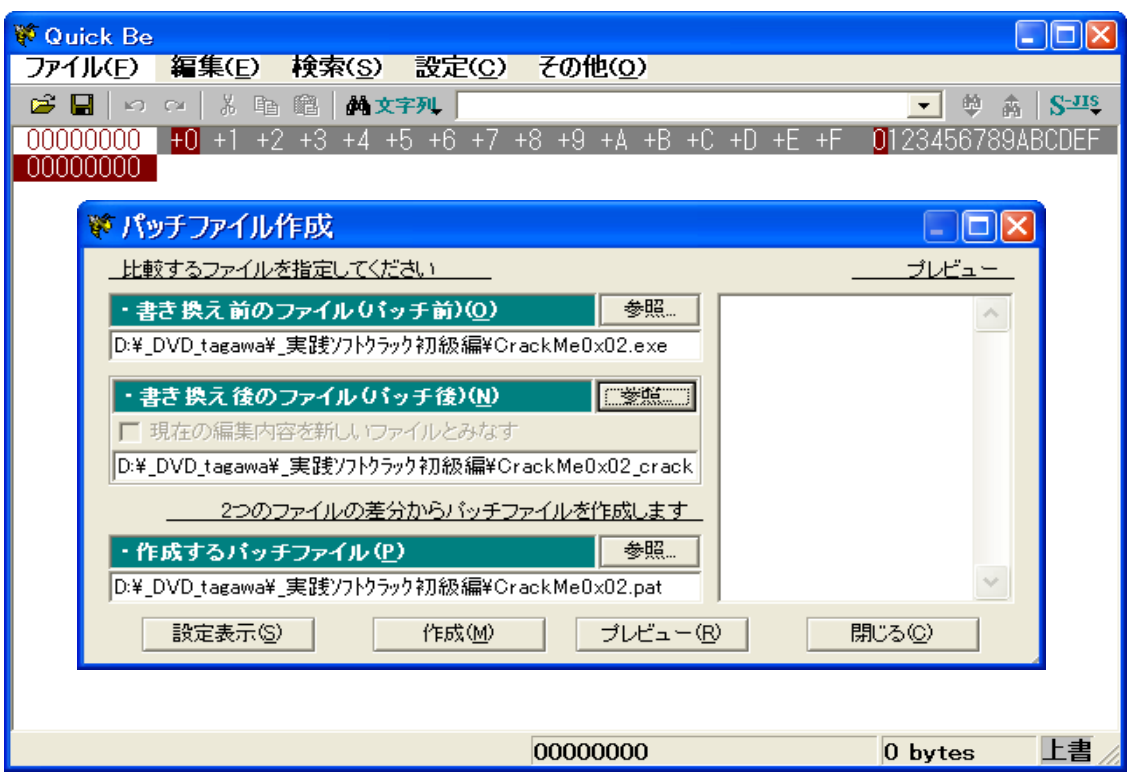


> [D]画面の[×]を押すと，[ファイル変更済み]のﾀﾞｲｱﾛｸﾞが出てきますので，[はい]ﾎﾞﾀﾝを押します．

> ﾘﾈｰﾑして[保存]を押します．

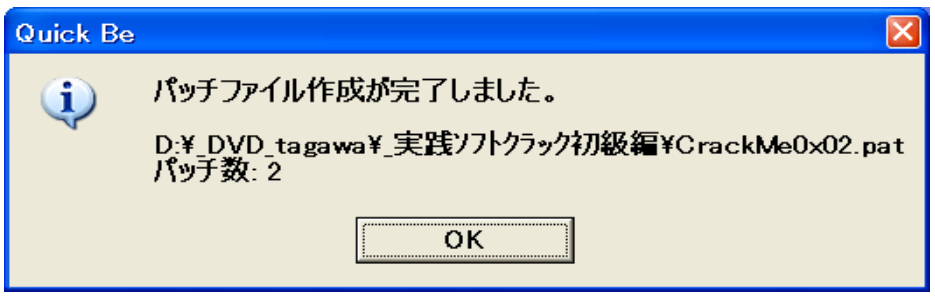
（上書き保存をしても，OllyDbgではﾊﾞｯｸｱｯﾌﾟﾌｧｲﾙを作成します）

これで，OllyDbgによるｸﾗｯｸは完了です．



> 次に quickbe でｸﾗｯｸﾊﾟｯﾁﾌｧｲﾙを作成します．

設定をして，[作成]を押します．



> ﾊﾟｯﾁﾌｧｲﾙ作成が完了しました．

```
*== TARGET_FILE ================
FILENAME CrackMe0x02.exe
* FileSize: 48640 bytes
* LastMod.: 2005/03/12 22:41:44
*================================
00000847: D5 FF
00000848: 07 FF
```

> ｸﾗｯｸﾊﾟｯﾁの作成(2)

こちらもｸﾗｯｸﾊﾟｯﾁの作成(1)と同様の方法で行います．

結果は以下の通りでした．

```
*== TARGET_FILE ================
FILENAME CrackMe0x02.exe
* FileSize: 48640 bytes
* LastMod.: 2005/03/12 22:41:44
*================================
00000849: 73 EB
0000084A: 07 00
```

次回は　　3) ID･ﾊﾟｽﾜｰﾄﾞ型　です

印刷時のﾍﾟｰｼﾞﾚｲｱｳﾄは考慮していませんので悪しからず.

以上

3) ID･ﾊﾟｽﾜｰﾄﾞ型

ｱﾌﾟﾘｹｰｼｮﾝによっては，登録の際にﾊﾟｽﾜｰﾄﾞの他にﾕｰｻﾞｰIDなど，
ﾕｰｻﾞｰ固有の情報の入力を求めるものがあります．
これらのﾌﾟﾛﾃｸｼｮﾝはﾕｰｻﾞｰID毎に正当なﾊﾟｽﾜｰﾄﾞを生成し，入力
されたものと比較するように設計･実装されています．

＊ 一般的なﾁｪｯｸﾙｰﾁﾝ

a. ﾕｰｻﾞｰIDとﾕｰｻﾞｰが入力したﾊﾟｽﾜｰﾄﾞ(文字列)を取得する
b. ﾕｰｻﾞｰIDからﾊﾟｽﾜｰﾄﾞを生成し，入力されたﾊﾟｽﾜｰﾄﾞとの妥当性を
ﾁｪｯｸする(ここが特定すべき個所)
c. ﾁｪｯｸの結果に応じてﾒｯｾｰｼﾞを表示する

ｴﾃﾞｨｯﾄﾎﾞｯｸｽから文字列を取得するにはWindowsが提供するAPI関数を
用いています．

＊ 文字列取得：API(Application Program Interface)関数

GetWindowText，GetDlgItemText，GetDlgItemInt の3種類です

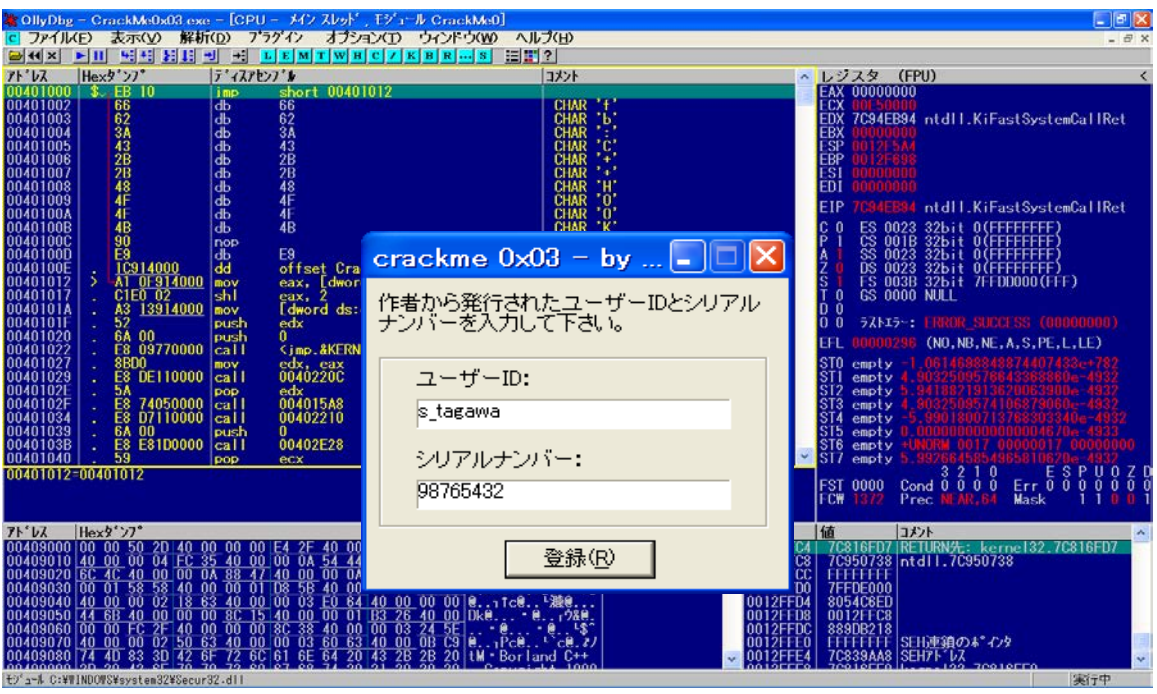
これらのAPI関数が呼び出される瞬間を捉えることにより，ﾁｪｯｸﾙｰﾁﾝの個所を
特定します．

＊ ﾁｪｯｸﾙｰﾁﾝのﾄﾚｰｽの第一歩

a. OllyDbgでｿﾌﾄの実行ﾌｧｲﾙを開きます
b. F9ｷｰを押して，ﾕｰｻﾞｰIDとﾊﾟｽﾜｰﾄﾞ入力のｴﾃﾞｨｯﾄﾎﾞｯｸｽが出たら，
ﾕｰｻﾞｰIDとﾌｪｲｸﾊﾟｽﾜｰﾄﾞを入れておきます
([OK]，[登録]等のﾎﾞﾀﾝは押しません)
c. 文字列取得のAPI関数にﾌﾞﾚｰｸﾎﾟｲﾝﾄを設定します
d. ここで，[OK]，[登録]等のﾎﾞﾀﾝを押します
e. API関数が実行される直前で停止(ﾌﾞﾚｰｸ)します
f. この先にﾁｪｯｸﾙｰﾁﾝが存在するはずです
g. これからは，1命令ずつ実行(F8ｷｰを押す)して精査を行っていきます

このﾊﾟｽﾜｰﾄﾞを生成するﾙｰﾁﾝを解析すれば，条件を満たすﾊﾟｽﾜｰﾄﾞを作り出す
Key Generator を作成できるということです．ただし，これには ｱｾﾝﾌﾞﾘ言語と
C言語の高度な知識と，C言語のｺﾝﾊﾟｲﾗが必要になります．

それでは"CrackMe0x03．exe"を解析してみましょう．

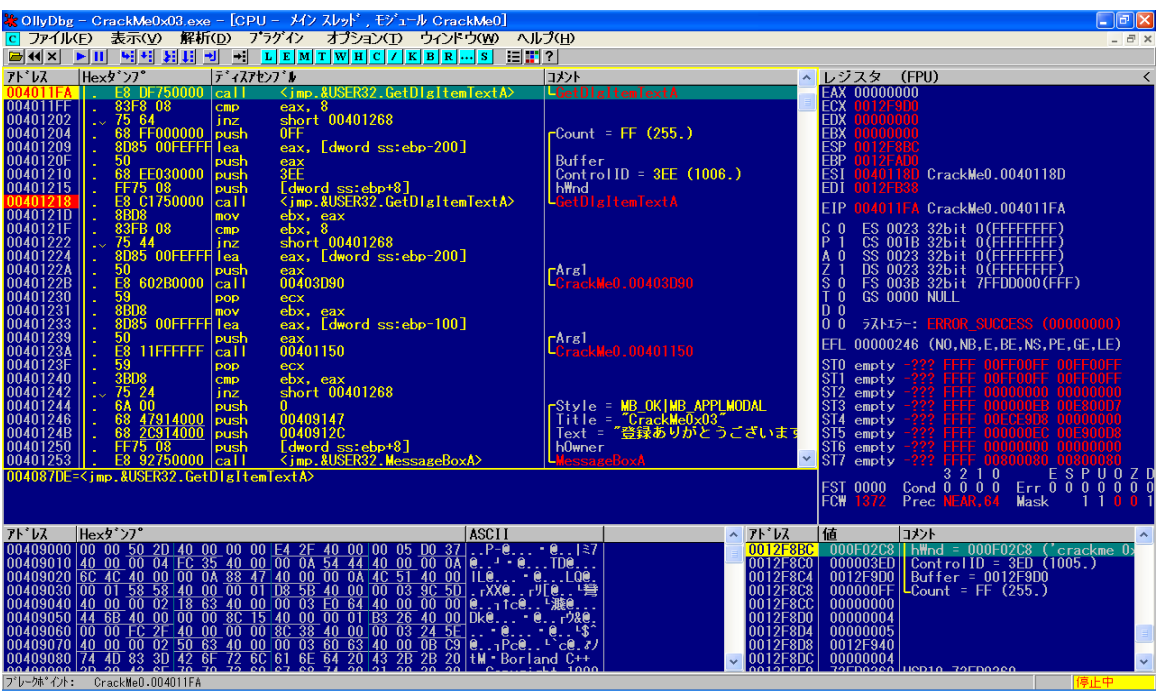


＞ OllyDbgで ｿﾌﾄの"CrackMe0x03．exe"を開きます．

F9ｷｰを1回押します．

ﾕｰｻﾞｰID：s_tagawa
ｼﾘｱﾙﾅﾝﾊﾞｰ：98765432　を入力します
(ここではﾊﾟｽﾜｰﾄﾞではなくｼﾘｱﾙﾅﾝﾊﾞｰとなっています)

([OK]，[登録]等のﾎﾞﾀﾝは押しません)



＞ 全てのAPI関数(GetDlgTextA)にﾌﾞﾚｰｸﾎﾟｲﾝﾄを設定します．
(操作の詳細は省略します)
今回はAPI関数が "GetDlgTextA"なのに注意して
下さい．

ｱﾌﾟﾘｹｰｼｮﾝのｴﾃﾞｨｯﾄﾎﾞｯｸｽの[登録]ﾎﾞﾀﾝを押します．

API関数が実行される直前("004011FA")でﾌﾞﾚｰｸ
しました．

このAPI関数によって何が取得されるのでしょうか?

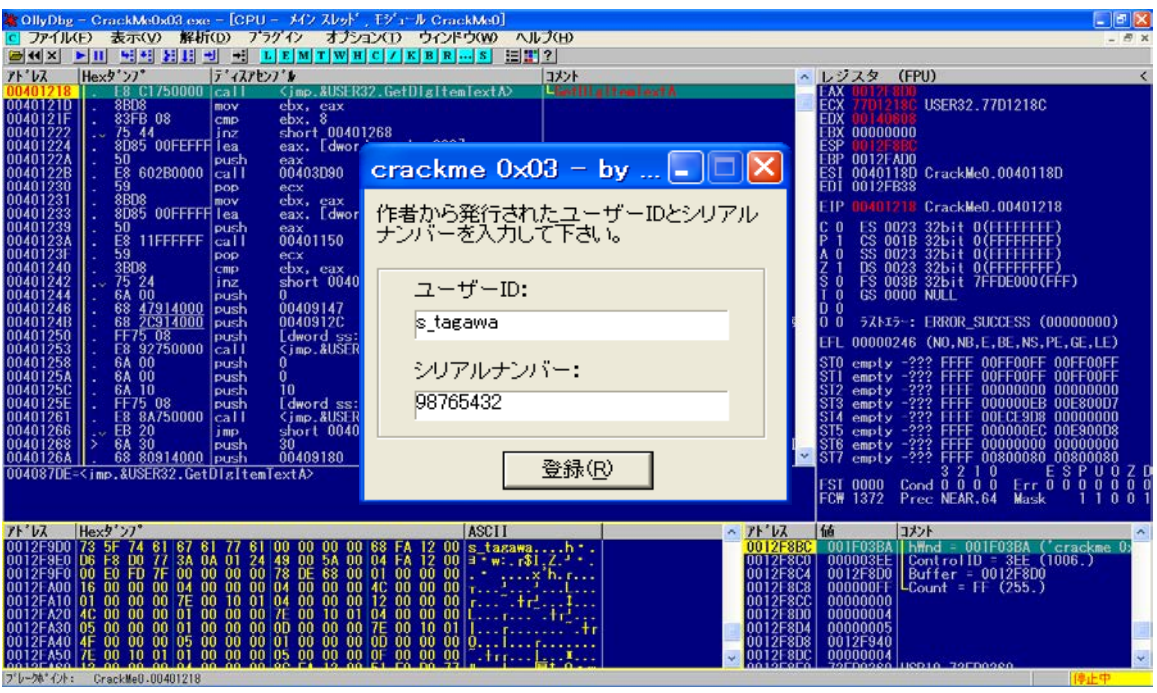
```
0012F8C0   000003EE   ControlID = 3EE (1006.)
0012F8C4   0012F8D0   Buffer = 0012F8D0
0012F8C8   000000FF   Count = FF (255.)
```

> ｽﾀｯｸｳｨﾝﾄﾞｳに注目します.

Bufferが示すｱﾄﾞﾚｽ "0012F8D0"に取得した文字列が格納されます.

この行を選択状態にして，右ｸﾘｯｸします.

[ﾀﾞﾝﾌﾟ画面へ]をｸﾘｯｸします.



> API関数(GetDlgTextA)を実行します.

[登録]ﾎﾞﾀﾝを押します.

F9ｷｰを2回押します.

次のAPI関数(GetDlgTextA)が実行される直前("00401218")でﾌﾞﾚｰｸしました.

ﾀﾞﾝﾌﾟｳｨﾝﾄﾞｳのASCII欄を見てみます.

```
0012F9D0  s_tagawa....h･.
```

> "s_tagawa"の文字列が表示されています.

> ﾌﾞﾚｰｸ前後のｺｰﾄﾞを見てみましょう.

```
ユーザーID取得･文字列長の比較
004011FA   E8 DF750000   call   <jmp.&USER32.GetDlgItemTextA>
004011FF   83F8 08       cmp    eax, 8
00401202   75 64         jnz    short 00401268
シリアルナンバーの取得･文字列の比較
00401204   68 FF000000   push   0FF    ;Count = FF (255.)
00401209   8D85 00FEFFFF lea    eax, [dword ss:ebp-200]
0040120F   50            push   eax    ;Buffer
00401210   68 EE030000   push   3EE    ;ControlID = 3EE (1006.)
00401215   FF75 08       push   [dword ss:ebp+8]      ;hWnd
00401218   E8 C1750000   call   <jmp.&USER32.GetDlgItemTextA>
0040121D   8BD8          mov    ebx, eax
0040121F   83FB 08       cmp    ebx, 8
00401222   75 44         jnz    short 00401268
入力ｼﾘｱﾙﾅﾝﾊﾞｰを数値に変換
00401224   8D85 00FEFFFF lea    eax, [dword ss:ebp-200]
0040122A   50            push   eax                    ; /Arg1
0040122B   E8 602B0000   call   00403D90
00401230   59            pop    ecx
00401231   8BD8          mov    ebx, eax
```

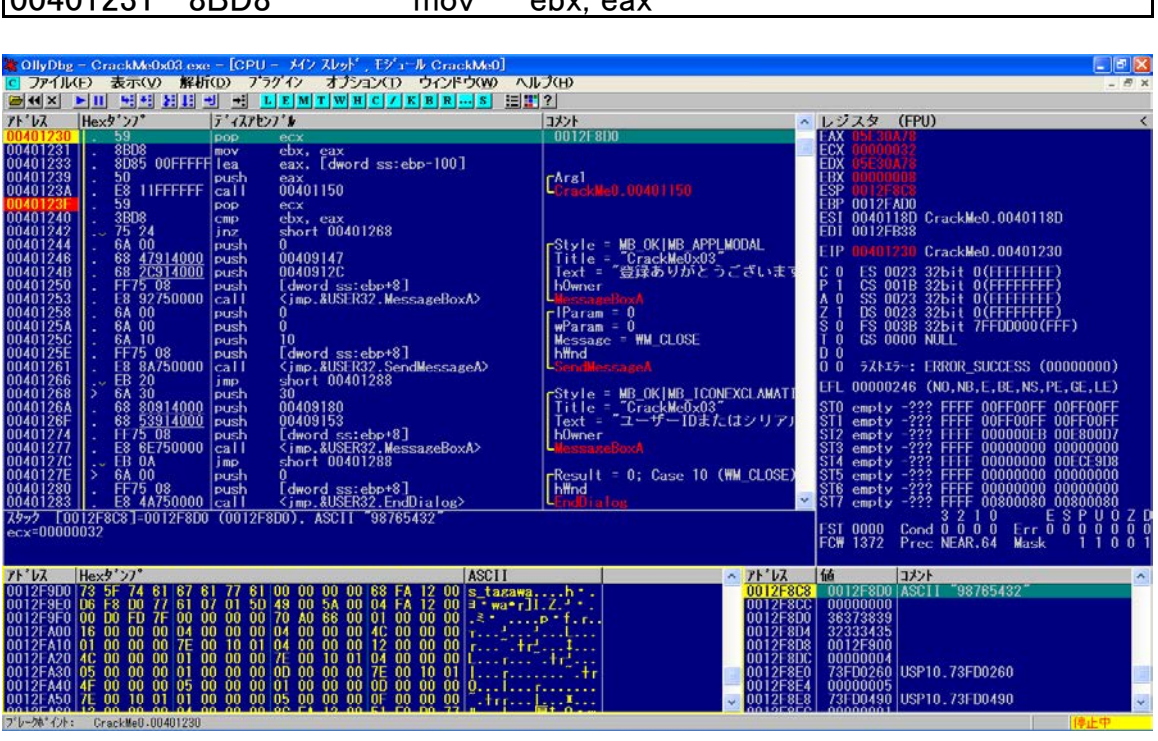
> ﾕｰｻﾞｰID取得･文字列長の比較
+ ﾕｰｻﾞｰID取得
+ 文字列長が8文字以外は登録失敗
+ 登録失敗処理へ
> ｼﾘｱﾙﾅﾝﾊﾞｰの取得･文字列の比較

+ ｼﾘｱﾙﾅﾝﾊﾞｰ取得(ﾄﾚｰｽの現在位置)

+ 文字列長が8文字以外は登録失敗
+ 登録失敗処理へ
> 入力ｼﾘｱﾙﾅﾝﾊﾞｰを数値に変換

+ 入力ｼﾘｱﾙﾅﾝﾊﾞｰのｱﾄﾞﾚｽ
+ 入力ｼﾘｱﾙﾅﾝﾊﾞｰを数値に変換

+ 戻り値を ebxﾚｼﾞｽﾀにｺﾋﾟｰ



> この戻り値を見てみます.

ここで "401230"にﾌﾞﾚｰｸﾎﾟｲﾝﾄを仕掛けます.
(左ｸﾘｯｸ+F2ｷｰ)

F9ｷｰを2回押します.

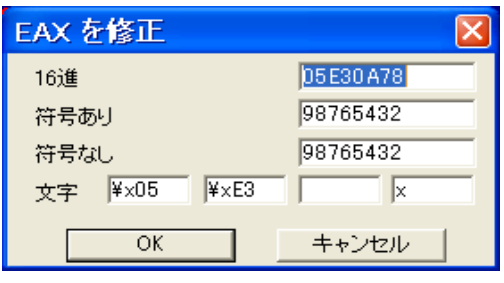
"00401218"でﾌﾞﾚｰｸしました.

ﾚｼﾞｽﾀｳｨﾝﾄﾞｳのEAXﾚｼﾞｽﾀの値が

EAX 05E30A78 となっています.

文字列 "98765432"が数値に変換されているはずです.

EAX の数値の部分をﾀﾞﾌﾞﾙｸﾘｯｸします.



> やはり，数値になっています.

Windows の電卓で計算して確認しても同様でした.

> 以降のｺｰﾄﾞを見てみましょう.

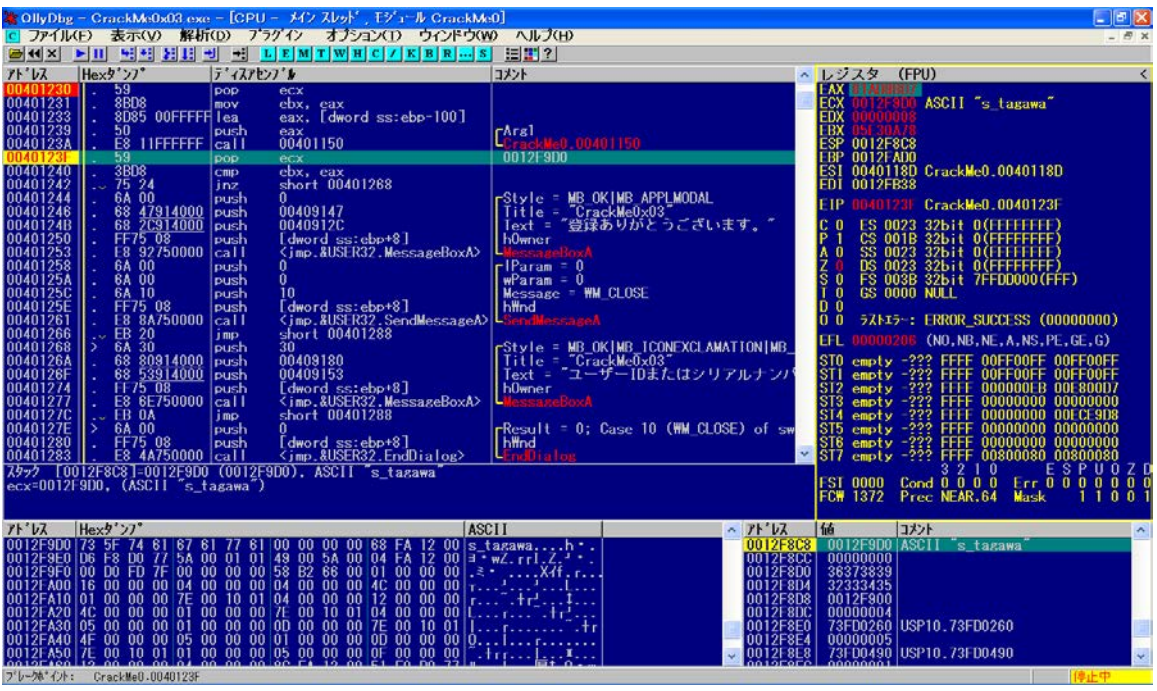
```
00401230   59              pop     ecx                ; 0012F8D0
00401231   8BD8            mov     ebx, eax
00401233   8D85 00FFFFFF   lea     eax, [dword ss:ebp-100]
00401239   50              push    eax                ; /Arg1
0040123A   E8 11FFFFFF     call    00401150
0040123F   59              pop     ecx
00401240   3BD8            cmp     ebx, eax
00401242   75 24           jnz     short 00401268
00401244   6A 00           push    0
00401246   68 47914000     push    00409147
0040124B   68 2C914000     push    0040912C
00401250   FF75 08         push    [dword ss:ebp+8]   ;hOwner
00401253   E8 92750000     call    <jmp.&USER32.MessageBoxA>
```

+ (ﾄﾚｰｽの現在位置)

+ 入力されたﾕｰｻﾞｰIDの格納ｱﾄﾞﾚｽを eaxﾚｼﾞｽﾀへｺﾋﾟｰ.
+ ﾕｰｻﾞｰIDから正規ｼﾘｱﾙﾅﾝﾊﾞｰを生成します.
+ ｽﾀｯｸから値を取り出して ecxﾚｼﾞｽﾀに書き込みます.
+ ｼﾘｱﾙﾅﾝﾊﾞｰの入力値と生成値を比較しています.
+ 登録失敗のﾒｯｾｰｼﾞを表示へｼﾞｬﾝﾌﾟします.

+ ;Title="CrackMe0x03"
+ ;Text = "登録ありがとうございます。"

+ 登録成功のﾒｯｾｰｼﾞを表示します.



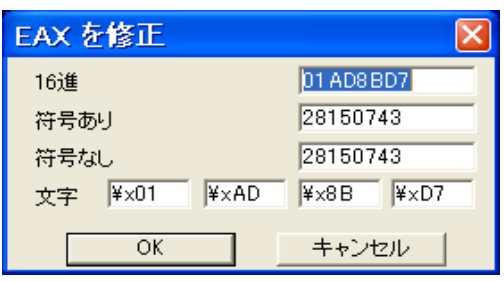
> 以上のことから, "call 00401150"命令の戻り値
(eaxﾚｼﾞｽﾀに返されます)を調べれば, 正規ｼﾘｱﾙ
ﾅﾝﾊﾞｰが判明します.

そこで, "004123F"にﾌﾞﾚｰｸﾎﾟｲﾝﾄをさらに仕掛けます.
(左ｸﾘｯｸ+F2ｷｰ)

F9ｷｰを1回押します.

"0040123F"でﾌﾞﾚｰｸしました.

EAXﾚｼﾞｽﾀの値"01AD8BD7"を10進数に変換すれば,
正規ｼﾘｱﾙﾅﾝﾊﾞｰが得られます.



> EAX の数値の部分をﾀﾞﾌﾞﾙｸﾘｯｸします.

ﾕｰｻﾞｰID: s_tagawa

ｼﾘｱﾙﾅﾝﾊﾞｰ: 28150743

Windows の電卓で計算して確認しても同様でした.

> ｸﾗｯｸﾊﾟｯﾁの作成(1)

```
00401242   75 24           jnz     short 00401268
00401244   6A 00           push    0
                           ↓
00401242   EB 00           jmp     short 00401244
00401244   6A 00           push    0
```

+ 登録失敗のﾒｯｾｰｼﾞを表示へｼﾞｬﾝﾌﾟします.

この条件ｼﾞｬﾝﾌﾟを次の行へ無条件ｼﾞｬﾝﾌﾟさせます.

> ｸﾗｯｸﾊﾟｯﾁの作成(2)

```
00401242   75 24           jnz     short 00401268
00401244   6A 00           push    0
                           ↓
00401242   90              nop
00401243   90              nop
00401244   6A 00           push    0
```

+ 登録失敗のﾒｯｾｰｼﾞを表示へｼﾞｬﾝﾌﾟします.

この条件ｼﾞｬﾝﾌﾟを何もしないに変更します.

> ｸﾗｯｸﾊﾟｯﾁの作成(3)

```
00401242   75 24           jnz     short 00401268
00401244   6A 00           push    0
                           ↓
00401242   74 24           je      short 00401268
00401244   6A 00           push    0
```

+ 登録失敗のﾒｯｾｰｼﾞを表示へｼﾞｬﾝﾌﾟします.

この条件ｼﾞｬﾝﾌﾟを等しければ分岐に変更します.

> ｸﾗｯｸﾊﾟｯﾁ(1)

```
*== TARGET_FILE ================
FILENAME CrackMe0x03.exe
* FileSize: 49152 bytes
* LastMod.: 2007/09/05 00:00:00
*================================
00000842: 75 EB
00000843: 24 00
```

> ｸﾗｯｸﾊﾟｯﾁ(2)

```
*== TARGET_FILE ================
```

> 左記のｸﾗｯﾁﾊﾟｯﾁが出来ました.

ﾕｰｻﾞｰID: 任意の 8 桁の文字列

ｼﾘｱﾙﾅﾝﾊﾞｰ: 任意の 8 桁の文字列

を入力するとが条件です.

ただし, ｸﾗｯｸﾊﾟｯﾁ(3)の場合は入力ｼﾘｱﾙﾅﾝﾊﾞｰと

生成ｼﾘｱﾙﾅﾝﾊﾞｰが不幸にして一致すると登録失敗に

FILENAME CrackMe0x03.exe
* FileSize: 49152 bytes
* LastMod.: 2007/09/05 00:00:00
*==============================
00000842: 75 90
00000843: 24 90

> ｸﾗｯｸﾊﾟｯﾁ(3)
*== TARGET_FILE ===============
FILENAME CrackMe0x03.exe
* FileSize: 49152 bytes
* LastMod.: 2007/09/05 00:00:00
*==============================
00000842: 75 74

終わります.

ｸﾗｯｸﾊﾟｯﾁは書き換えるﾊﾞｲﾄ数をできだけ

少なくするのが美しいといわれています.

> さて, 解析を少し前に戻します.

0040123A　E8 11FFFFFF　　call　　00401150

+ ﾕｰｻﾞｰIDから正規ｼﾘｱﾙﾅﾝﾊﾞｰを生成します.

> ここに注目すると, この call命令の先に Key Generator のﾙｰﾁﾝが隠されているようです.

この部分のｺｰﾄﾞを分析して(勿論, ｱｾﾝﾌﾞﾘ言語を),
『C言語』等の高級言語で記述し, ｺﾝﾊﾟｲﾗで実行ﾌｧｲﾙを
作成すれば Keygen が出来ます.

印刷時のﾍﾟｰｼﾞﾚｲｱｳﾄは考慮していませんので悪しからず.

以上