시리얼 번호 작성 루틴을 뽑아내서 Brute-Force 돌리기..

작성일: 2005년 가을
작성자: graylynx (graylynx at gmail.com)

크랙미 정보

```
GOAL :  Find the correct password
 No patching allowed
Difficulty : 2/10
This is an exercise on brute-attacking.
Written in WIN32ASM by Detten.

Send solutions, questions, and comments to :
 Biwc@hotmail.com
```

역시 군대에서 풀었던 크랙미.
이건 그때 인트라넷 동호회에 강좌 같은거 올리려고 솔루션을 문서로 만들어 놨네요.
부끄럽지만 살짝 첨부해 봅니다. ^^;

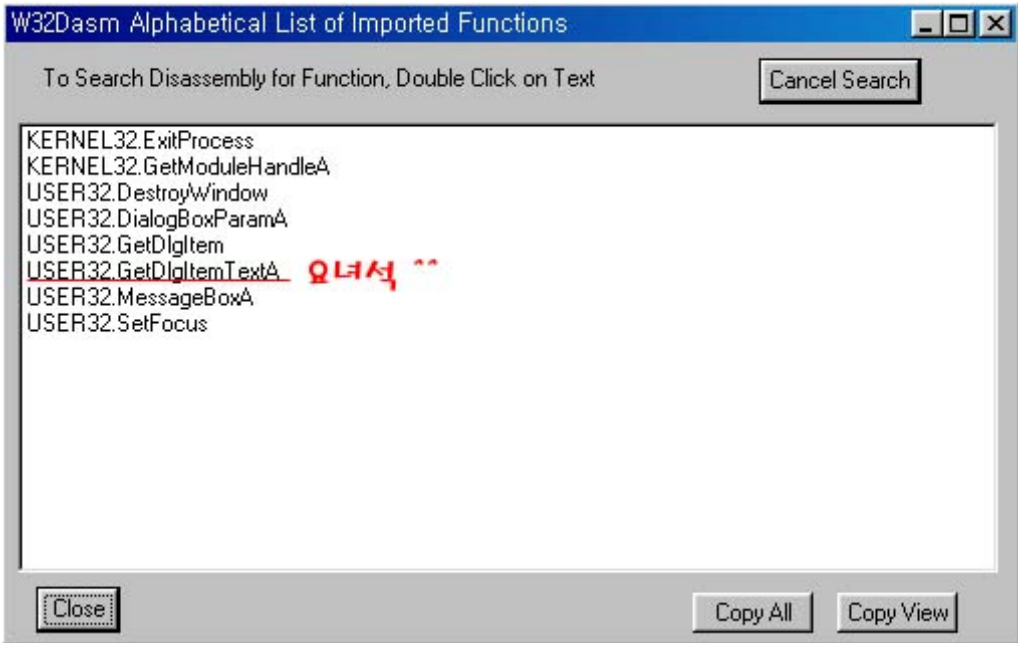
[크랙미 환경]
OS: Windows XP SP1
Disassembler: W32DASM
Debugger: SoftIce + IceDump
Assembler: MASM
Compiler: Visual C++

crackme5(by Detten) 튜토리얼
by 태병장

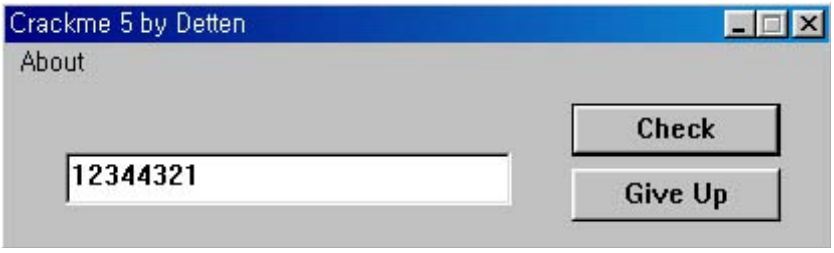
자, 리버싱팀의 오브젝트 #4 라는 이름으로 올렸던 크랙미5에 대한 튜토리얼을 시작하겠습니다. 이 크랙미는 코드 패치가 허용되지 않으며, 오로지 올바른 키값을 찾아야만 합니다.
따라서 우리는 디스어셈블 코드를 완벽하게 이해할 수 있어야만 이녀석을 공략할 수 있습니다.

먼저 W32DASM으로 디스어셈블을 해봅니다. 실행파일이 패킹되어있지도 않고, 숨겨진 더미코드도 없기 때문에 올바른 역코드를 보여줍니다. 이 코드를 분석하기 전에 먼저 소프트아이스를 이용하여 실시간으로 코드를 훑어보도록 하죠.

W32DASM의 Function->Import 메뉴를 선택하면, 아래와 같은 함수들이 임포트 되어있음을 알 수 있습니다.

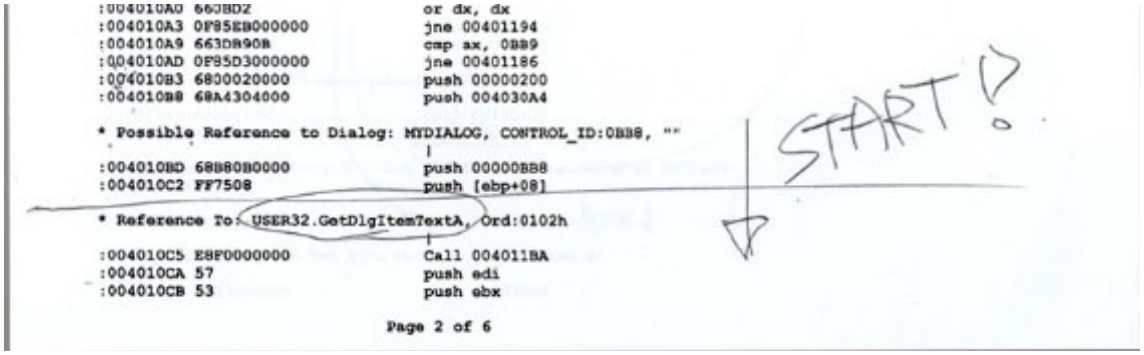


소프트아이스를 띄운다음 bpx getdlgitemtexta 명령으로 브레이크 포인트를 잡습니다. 그리고 crackme5.exe를 실행하고 아무값이나 입력하면 브레이크가 걸리면서 소프트아이스가 실행됩니다.

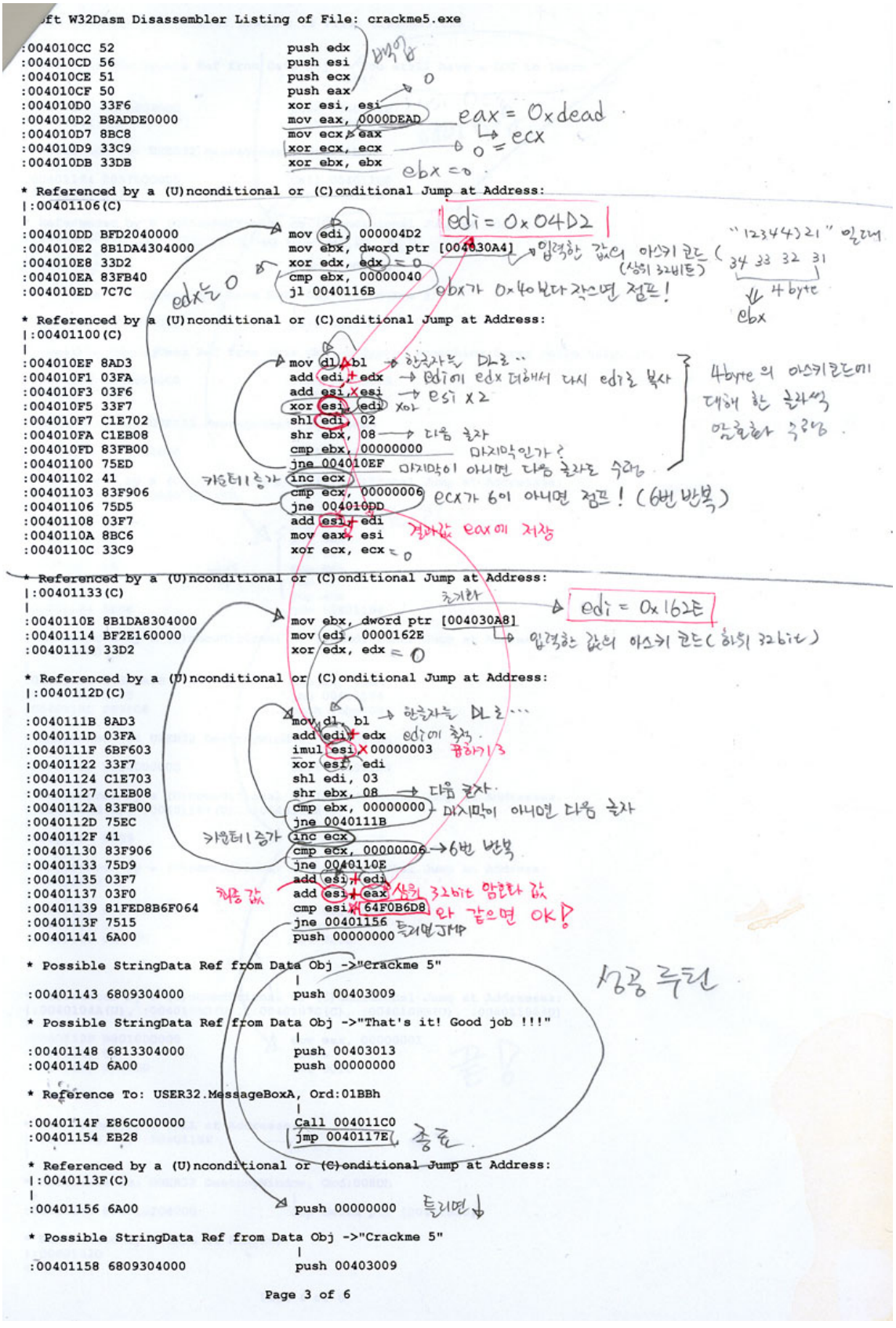


(저는 12344321 을 입력하였습니다)

소프트아이스가 떳을 때 getdlgitemtexta함수 내에서 브레이크가 걸린 상태이므로, p ret명령을 내려서 (함수 밖으로 빠져 나가는 명령) 우리가 원하는 코드로 접속할 수 있습니다. 이제 우리는 F8과 F10을 적절히 이용하여 우리가 입력한 12344321 문자열이 어떠한 연산과정을 거치는지 레지스터와 메모리 값을 추적해가며 조사할 수 있습니다. 하지만 이 크랙미는 W32DASM으로 디스어셈블이 가능하기 때문에, 그것의 디스어셈블 코드로 분석을 하겠습니다. 밑의 그림은 이미 제가 손으로 분석한 것을 스캐너로 받아서 첨부한 것입니다.



```
ft W32Dasm Disassembler Listing of File: crackme5.exe

:004010CC 52                    push edx
:004010CD 56                    push esi
:004010CE 51                    push ecx
:004010CF 50                    push eax
:004010D0 33F6                  xor esi, esi
:004010D2 B8ADDE0000            mov eax, 0000DEAD
:004010D7 8BC8                  mov ecx, eax
:004010D9 33C9                  xor ecx, ecx
:004010DB 33DB                  xor ebx, ebx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401106(C)
|
:004010DD BFD2040000            mov edi, 000004D2
:004010E2 8B1DA4304000          mov ebx, dword ptr [004030A4]
:004010E8 33D2                  xor edx, edx
:004010EA 83FB40                cmp ebx, 00000040
:004010ED 7C7C                  jl 0040116B

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401100(C)
|
:004010EF 8AD3                  mov dl, bl
:004010F1 03FA                  add edi, edx
:004010F3 03F6                  add esi, esi
:004010F5 33F7                  xor esi, edi
:004010F7 C1E702                shl edi, 02
:004010FA C1EB08                shr ebx, 08
:004010FD 83FB00                cmp ebx, 00000000
:00401100 75ED                  jne 004010EF
:00401102 41                    inc ecx
:00401103 83F906                cmp ecx, 00000006
:00401106 75D5                  jne 004010DD
:00401108 03F7                  add esi, edi
:0040110A 8BC6                  mov eax, esi
:0040110C 33C9                  xor ecx, ecx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401133(C)
|
:0040110E 8B1DA8304000          mov ebx, dword ptr [004030A8]
:00401114 BF2E160000            mov edi, 0000162E
:00401119 33D2                  xor edx, edx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040112D(C)
|
:0040111B 8AD3                  mov dl, bl
:0040111D 03FA                  add edi, edx
:0040111F 6BF603                imul esi, 00000003
:00401122 33F7                  xor esi, edi
:00401124 C1E703                shl edi, 03
:00401127 C1EB08                shr ebx, 08
:0040112A 83FB00                cmp ebx, 00000000
:0040112D 75EC                  jne 0040111B
:0040112F 41                    inc ecx
:00401130 83F906                cmp ecx, 00000006
:00401133 75D9                  jne 0040110E
:00401135 03F7                  add esi, edi
:00401137 03F0                  add esi, eax
:00401139 81FED8B6F064          cmp esi, 64F0B6D8
:0040113F 7515                  jne 00401156
:00401141 6A00                  push 00000000

* Possible StringData Ref from Data Obj ->"Crackme 5"
                                |
:00401143 6809304000            push 00403009

* Possible StringData Ref from Data Obj ->"That's it! Good job !!!"
                                |
:00401148 6813304000            push 00403013
:0040114D 6A00                  push 00000000

* Reference To: USER32.MessageBoxA, Ord:01BBh
                                |
:0040114F E86C000000            Call 004011C0
:00401154 EB28                  jmp 0040117E

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040113F(C)
|
:00401156 6A00                  push 00000000

* Possible StringData Ref from Data Obj ->"Crackme 5"
                                |
:00401158 6809304000            push 00403009

                        Page 3 of 6
```



```
* Possible StringData Ref from Data Obj ->"You still have a LOT to learn "
                                        ->"!"
                                        |
:0040115D 682B304000                    push 0040302B
:00401162 6A00                          push 00000000

* Reference To: USER32.MessageBoxA, Ord:01BBh
                                        |
:00401164 E857000000                    Call 004011C0
:00401169 EB13                          jmp 0040117E

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004010ED(C)
|
:0040116B 6A00                          push 00000000

* Possible StringData Ref from Data Obj ->"Crackme 5"
                                        |
:0040116D 6809304000                    push 00403009

* Possible StringData Ref from Data Obj ->"Typing something first could help..."
                                        |
:00401172 684B304000                    push 0040304B
:00401177 6A00                          push 00000000

* Reference To: USER32.MessageBoxA, Ord:01BBh
                                        |
:00401179 E842000000                    Call 004011C0

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00401154(U), :00401169(U)
|
:0040117E 58                            pop eax
:0040117F 59                            pop ecx
:00401180 5E                            pop esi
:00401181 5A                            pop edx
:00401182 5B                            pop ebx
:00401183 5F                            pop edi
:00401184 EB0E                          jmp 00401194

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004010AD(C)
|
:00401186 663DBA0B                      cmp ax, 0BBA
:0040118A 7508                          jne 00401194
:0040118C FF7508                        push [ebp+08]

* Reference To: USER32.DestroyWindow, Ord:008Dh
                                        |
:0040118F E814000000                    Call 004011A8

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004010A3(C), :00401184(U), :0040118A(C)
|
:00401194 EB09                          jmp 0040119F

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401069(C)
|
:00401196 B800000000                    mov eax, 00000000
:0040119B C9                            leave
:0040119C C21000                        ret 0010


* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040104A(U), :0040105D(U), :0040107C(C), :00401095(U), :00401194(U)
|
:0040119F B801000000                    mov eax, 00000001
:004011A4 C9                            leave
:004011A5 C21000                        ret 0010


* Referenced by a CALL at Addresses:
|:00401058    , :0040118F
|

* Reference To: USER32.DestroyWindow, Ord:008Dh
                                        |
:004011A8 FF2510204000                  Jmp dword ptr [00402010]

* Referenced by a CALL at Address:
|:00401020
|
```

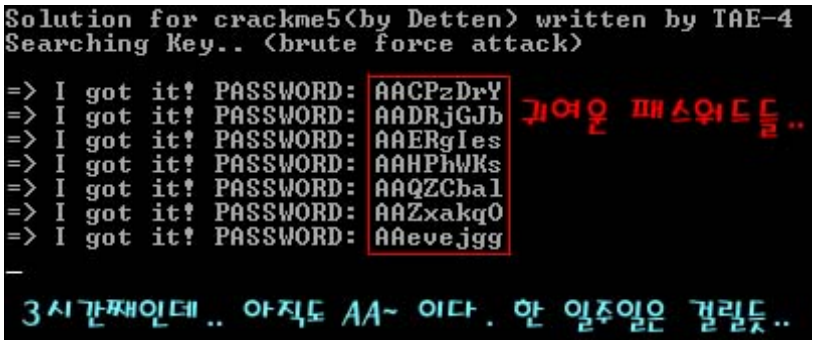
실제로 저는 디버거 상에서 주석 달면서 내려가는것 보다 이렇게 프린트해놓고 손으로 분석하는게 집중이 잘되더군요. (사실 키보드를 잡고 있을 충분한 시간이 없어서 여기에 적응되어버린 건지도,,)

이제 어떻게 해야 하느냐? 두가지 방법이 있습니다. 위 암호화 루틴을 역으로 분석하여 복호화 루틴을 만드는 것과 순차적으로 입력가능한 모든 경우의 수를 대입해보는 방법이 있습니다. 어떤 방법이 더 어려울지는 짐작이 가시겠죠? 전자의 방법은 알고리즘을 완벽하게 파악해야함을 물론이고, 상당한 수학적 능력이 요구됩니다. 후자의 방법은 시간이 오래 걸리긴 하지만, 복호화 루틴없이 암호화 루틴만으로 복호화 코드를 알아 낼 수 있습니다. 우리는 이미 암호화 루틴은 알고 있기 때문에 두번째 방법을 이용 하는것이 훨씬 쉽겠죠? 저도 머리 쓰는것을 싫어하기 때문에 brute force 공격을 시도하겠습니다.

먼저 위 루틴과 동일한 기능의 암호화 함수를 만들었습니다. 이미 W32DASM이 어셈블리어로 알려주었으므로 MASM을 이용하면 더 쉽게 작성할 수 있습니다. 아니, MASM을 사용할 수 밖에 없죠. (crack.asm) 저역시 icedump를 이용하여 코드를 덤프한뒤 점프코드들에 대해 적절히 라벨명을 수정하는것 만으로 간단하게 만들 수 있었습니다.

MASM으로 라이브러리 형태로 컴파일한 뒤(ml /coff crack.asm) 확장자를 .lib으로 바꿔줍니다. 함수에 대한 프로토타입을 long crack(char *sText); 헤더파일(crack.h)에 정의하고, 우리가 C로 작성할 무차별 대입 프로그램(solution.c)에서 include 시켜줍니다. 이 함수는 우리가 파라미터로 넘겨준 8자리 문자열을 가지고 크랙미에서 하는것과 똑같은 연산을 하여 32비트 키값을 리턴합니다. 크랙미에서는 미리 메모리에 저장해둔 키값 0x64F0B6D8 와 비교하므로, 우리는 모든 값을 차례대로 대입하면서 리턴값이 0x64F0B6D8 와 같은지를 체크하면 되겠습니다.

처음에는 숫자를 입력해봤습니다만, 답이 안나오더군요. readme.txt에서도 word(단어) 라고 말하고 있으므로 8자리의 대입가능한 모든 알파벳으로 다시 공격을 시도했습니다. 결과는...? 빙고~!



공격한지 몇 분 되지 않아 패스워드를 뱉어 내는군요 :)
조합가능한 수많은 경우의 수가 있기 때문에, 패스워드가 한두개가 아니군요. 단점이 있다면 모든 패스워드를 알아내는 데에는 상당한 시간이 필요하다는 것입니다. 지금의 대문자/소문자의 조합으로도 AAAAAAAA부터 zzzzzzzz까지 연산하는데 몇 일은 족히 걸릴 듯 싶습니다. 거기다 숫자, 특수문자까지 포함된다면.. 어쩌면 제대하기 전에 결과를 못 볼지도 모릅니다 :)

실행되는 과정을 출력하게끔 하면 더욱더 느려지기 때문에(printf가 상당히 느립니다) 패스워드를 찾았을 때만 출력하도록 하는 것이 스트레스를 덜 받겠죠? Detten은 말이되는 단어(word)를 패스워드로 설정한 듯 한데요, 제가 알아낸 것은 다 말이 안되는 것들이네요. 뭐, 그래도 어때요 크랙미에 입력해보면 맞다고 하는걸요. hehe~

성공~

[첨부1] crack.h

```
#ifndef _CRACK_H_
#define _CRACK_H_

extern unsigned long crack( char *text );

#endif
```

[첨부2] crack.asm

```
.386
.MODEL FLAT
.DATA
KEY_HIGH DD  00000000h
KEY_LOW    DD00000000h

.CODE
_crack   PROC NEAR
       PUBLIC _crack

       PUSH      EBP
       MOV       EBP, ESP
       PUSH      EDI
       PUSH      EBX
       PUSH      EDX
       PUSH      ESI
       PUSH      ECX
       MOV       EAX, [EBP + 8]
       MOV       EBX, [EAX]
       MOV       KEY_HIGH, EBX
```

```
    MOV     EBX, [EAX + 4]
    MOV     KEY_LOW,  EBX
    XOR     ESI, ESI
    MOV     EAX, 0000DEADh
    MOV     ECX, EAX
    XOR     ECX, ECX
    XOR     EBX, EBX
LOOP2:
    MOV     EDI, 000004D2h
    MOV     EBX, KEY_HIGH
    XOR     EDX, EDX
    CMP     EBX, 40h
    JL      ERROR
LOOP1:
    MOV     DL,  BL
    ADD     EDI, EDX
    ADD     ESI, ESI
    XOR     ESI, EDI
    SHL     EDI, 02h
    SHR     EBX, 08h
    CMP     EBX, 00h
    JNZ     LOOP1
    INC     ECX
    CMP     ECX, 06h
    JNZ     LOOP2
    ADD     ESI, EDI
    MOV     EAX, ESI
    XOR     ECX, ECX
LOOP4:
    MOV     EBX, KEY_LOW
    MOV     EDI, 0000162Eh
    XOR     EDX, EDX
LOOP3:
    MOV     DL,  BL
    ADD     EDI, EDX
    IMUL    ESI, 03h
    XOR     ESI, EDI
    SHL     EDI, 03h
    SHR     EBX, 08h
    CMP     EBX, 00h
```

```
        JNZ       LOOP3
        INC       ECX
        CMP       ECX, 06h
        JNZ       LOOP4
        ADD       ESI, EDI
        ADD       ESI, EAX
        MOV       EAX, ESI
        JMP       EXIT
    ERROR:
        MOV       EAX, 0
    EXIT:
        POP       ECX
        POP       ESI
        POP       EDX
        POP       EBX
        POP       EDI
        MOV       ESP, EBP
        POP       EBP
        RET
_crack   ENDP

END
```

[첨부3] solution.c

```
#include <stdio.h>
#include "crack.h"
#define    CRYPTED_KEY  0x64F0B6D8

int main( void )
{
  static char sText[8] = "AAAAAAAA";
  register char bCnt;
  unsigned long lKey = 0;

  printf( "Solution for crackme5(by Detten) written by TAE-4₩n" );
  printf( "Searching Key.. (maybe it takes very long time)₩n₩n" );
  //printf( "Input Text    Crypted Key₩n" );
  //printf( "----------    ----------₩n" );
```

```
  while( sText[0] < 'z' )
  {
    for( bCnt = 7; bCnt > 0; bCnt-- )
    {
      if( sText[bCnt] == ('Z' + 1) )
        sText[bCnt] = 'a';
      if( sText[bCnt] == ('z' + 1) ) {
        sText[bCnt] = 'A';
        sText[bCnt - 1]++;
      }
    }

    lKey = crack( sText );
    //printf( "₩r %s      %X", sText, lKey );
    if( lKey == CRYPTED_KEY )
      printf( "    => I got it! Key correct!₩n" );

    sText[7]++;
  }
  return( 0 );
}
```