



[CrackMe 10종 풀이]

Written by Osiris (email, msn - mins4416@naver.com)

by beistlab (<http://beist.org>)

Synopsis

샤프를 쓰는 사람이라면 누구나 샤프심이 나오지 않아서 고생했던 적이 있을 것이다. 그럴 때는 샤프를 흔들어 보고 소리가 나지 않으면 샤프심을 넣어본다. 그래도 샤프심이 나오지 않으면 샤프를 뜯어보고 샤프심이 나오는 구멍이 막히진 않았는지 확인한다. 만약 막혀 있다면 샤프지우개 끝에 달려있는 얇은 철사를 구멍에 넣는데 그렇게 하다 보면 짧게 토막난 샤프심이 떨어져 나가고 다시 조립하면 샤프를 사용할 수 있게 된다. 난 위와 같은 경우도 Reverse Engineering(역 공학)을 통해서 문제를 해결한 한 경우가 아닐까 생각한다.

본 문서에서 다루고 있는 Crackme 10종 풀이는 위에서 언급한 샤프 고치는 법과 비슷하다고 볼 수 있다. CrackMe에서 제시하고 있는 문제들을 이런 저런 방법을 동원하여 성공 메시지가 뜨게 하면 되는 것이다.

본 문서에서 다루는 CrackMe 문제는 Key값 찾기, Keygen만들기, Keyfile만들기, 경우의수 퍼즐풀기, 입력한 값마다 다른 Serial찾기 등이 있다. 나는 이 문서에서 단순히 성공메시지만을 보기 위해서 CrackMe를 풀이 하는 것이 아니라 Key값을 생성하는 CrackMe라면 Reverse Engineering(역 공학)을 통해서 어떤 공식으로 Key값을 생성하는지 찾아내고, 그 Key값을 생성하는 프로그램을 따로 만들어, 이 문서를 보는 사람이 CrackMe에 대해 정확한 이해를 할 수 있도록 노력하였다.

이 문서를 큰 어려움 없이 보기 위해서는 기본적인 Assembly 지식은 갖추고 있어야 한다. 마지막으로 이 문서가 Reverse Engineering에 입문하려는 분들에게 도움이 되기를 있기를 바란다.

Contents

0x01. Duelist's Crackme #1

0x02. Duelist's Crackme #2

0x03. Duelist's Crackme #3

0x04. Duelist's Crackme #4

0x05. Duelist's Crackme #5

0x06. keygenning4newbies #1

0x07. CaD's Crack Me #1

0x08. Orion Crackme #1

0x09. CTM-CM #1

0x0a. Bengaly Crackme #3

0x0b. 참고사이트 & 참고문헌

0x01 Duelist #1

Crackme: <http://beist.org/research/public/crackme10/duel-cm1.zip>

Keygen: <http://beist.org/research/public/crackme10/duel-cm1-keygen.zip>

Author: Duelist

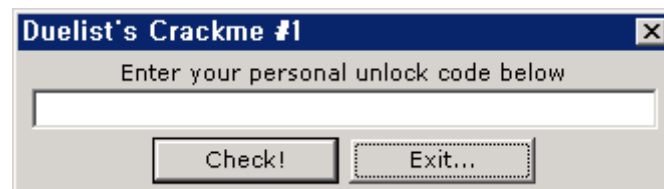
Level: ★

Protection: Serial

0x0001 목표

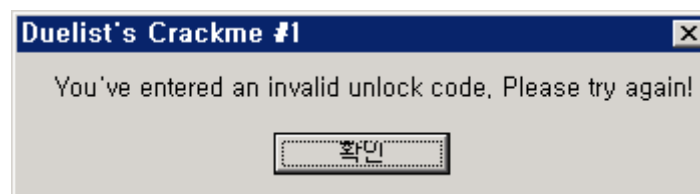
첫 번째 CrackMe에서는 GetDlgItemText같은 API를 사용합니다. XOR연산을 이용해서 만들어지는 Serial을 중점적으로 풀이 및 분석합니다.

0x0002 분석 및 풀이



[그림1-1. Duelist's Crackme #1 실행화면]

[그림1-1]에서 보시는 바와 같이 Duelist's Crackme #1은 텍스트박스에 특정 코드를 입력하도록 요구하고 있습니다.



[그림1-2. 잘못된 코드가 입력되었을 때 보여주는 메시지박스]

우선 숫자 1을 입력해서 어떤 메시지박스가 뜨는지 확인해 보았습니다. 그랬더니 잘못된 코드를 입력하였고 다시 시도하라는 메시지박스를 확인할 수 있었습니다.

이제 OllyDbg를 이용해서 분석을 하도록 하겠습니다. OllyDbg를 이용해 Duelist's Crackme #1을 열어 보았습니다.

CPU - main thread, module due-cm1			
00401000	\$ 6A 00	push 0	[pModule = NULL GetModuleHandleA
00401002	. E8 EC010000	call <jmp,&KERNEL32.GetModuleHandleA>	
00401007	. A3 63214000	mov dword ptr ds:[402163], eax	[RsrcName = 4, hInst => NULL LoadIconA
0040100C	. C705 37214000	mov dword ptr ds:[402137], 4003	
00401016	. C705 3B214000	mov dword ptr ds:[40213B], due-cm1,0040	[RsrcName = IDC_ARROW hInst = NULL LoadCursorA
00401020	. C705 3F214000	mov dword ptr ds:[40213F], 0	
0040102A	. C705 43214000	mov dword ptr ds:[402143], 0	
00401034	. A1 63214000	mov eax, dword ptr ds:[402163]	
00401039	. A3 47214000	mov dword ptr ds:[402147], eax	
0040103E	. 6A 04	push 4	
00401040	. 50	push eax	
00401041	. E8 C7020000	call <jmp,&USER32.LoadIconA>	
00401046	. A3 4B214000	mov dword ptr ds:[40214B], eax	
0040104B	. 68 007F0000	push 7F00	
00401050	. 6A 00	push 0	
00401052	. E8 50020000	call <jmp,&USER32.LoadCursorA>	

[그림1-3. OllyDbg로 열었을 때 보이는 코드들]

천천히 스크롤 바를 아래로 내리면서 확인해보았습니다. 조금 내려가보니 다음과 같은 API를 확인할 수 있었습니다.

004010FB	> 6A 24	push 24	[Count = 24 (36,) Buffer = due-cm1,004020F7 ControlID = 1 hWnd GetDlgItemTextA
004010FD	. 68 F7204000	push due-cm1,004020F7	
00401102	. 6A 01	push 1	
00401104	. FF75 08	push dword ptr ss:[ebp+8]	
00401107	. E8 55020000	call <jmp,&USER32.GetDlgItemTextA>	

[그림1-4. GetDlgItemText API]

GetDlgItemText API의 대한 설명은 다음과 같습니다.

```

UINT GetDlgItemText(
    HWND hDlg, //컨트롤을 가지고 있는 윈도우의 핸들
    int nIDDlgItem, //컨트롤의 ID
    LPTSTR lpString, //문자열을 돌려받기 위한 버퍼 포인터
    int nMaxCount //버퍼의 길이. 충분한 길이의 버퍼를 제공하는 것이 좋음
);
    
```

설명

WM_GETTEXT 메시지를 컨트롤로 보내 컨트롤의 텍스트를 읽어 lpString 버퍼에 채워준다. 만약 버퍼 길이(nMaxCount)보다 문자열이 더 길면 문자열은 잘려진다.

[그림1-4]와 GetDlgItemText API설명을 보면 알 수 있듯이, 버퍼의 길이는 0x24 입니다. 그

리고 버퍼가 있는 곳은 004020F7이라는걸 알 수 있습니다. 버퍼가 있는 곳을 보겠습니다.

Address	Hex dump	ASCII
004020F7	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00402107	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00402117	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

[그림1-5. 004020F7 버퍼가 있는 곳]

[그림1-5]에서 빨간 테두리로 둘러 쌓인 곳이 버퍼입니다. API 설명대로라면 우리가 입력하는 값이 이곳에 저장될 것입니다. 실제로 값을 입력하고 버퍼에 내용이 저장되는지 확인해보겠습니다.

Address	Hex dump	ASCII
004020F7	74 68 69 73 20 69 73 20 62 75 66 66 65 72 3F 00	this is buffer?.
00402107	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00402117	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00402127	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

[그림1-6. 값이 입력된 후의 버퍼]

"this is buffer?"라는 문자열을 입력해보니 해당 영역에 값이 저장된 것을 확인할 수 있습니다. 이제 [그림1-4]의 GetDlgItemText API 이후의 코드를 보겠습니다.

0040110C	, 33C0	xor eax, eax	EAX 초기화
0040110E	> 80B8 F7204001	cmp byte ptr ds:[eax+4020F7], 0	0과 비교해 입력된 값의 유무를 확인
00401115	, 74 18	je short due-cm1,0040112F	값이 없다면 0040112F로 분기
00401117	, 80B0 F7204001	xor byte ptr ds:[eax+4020F7], 43	[주소가 가리키는 곳의 값] xor 43
0040111E	, 80B0 F7204001	xor byte ptr ds:[eax+4020F7], 1E	[주소가 가리키는 곳의 값] xor 1E
00401125	, 80B0 F7204001	xor byte ptr ds:[eax+4020F7], 55	[주소가 가리키는 곳의 값] xor 55
0040112C	, 40	inc eax	EAX = EAX + 1
0040112D	, E2 DF	loopd short due-cm1,0040110E	0040110E로 돌아감
0040112F	> 83F8 00	cmp eax, 0	EAX와 0을 비교
00401132	, 75 18	jnz short due-cm1,0040114C	EAX가 0이 아니면 분기

[그림1-7. GetDlgItemText 이후 코드]

[그림1-7]은 00401107이후 바로 이어지는 코드입니다. [그림1-7]에서 볼 수 있듯이 각 코드가 주석을 달아 놓았습니다.

```
0040110C xor eax, eax
//같은 값을 XOR연산하기 때문에 0이 됩니다.
0040110E cmp byte ptr ds:[eax+4020F7], 0
//1byte 단위의 내용[eax+4020F7]을 0과 비교합니다.
00401115 je short due-cm1,0040112F
//만약 0040110E에서 비교한 내용이 같다면 0040112F로 분기합니다.
00401117 xor byte ptr ds:[eax+4020F7], 43
```

//1byte 단위의 내용[eax+4020F7]을 43과 xor 연산합니다.
0040111E xor byte ptr ds:[eax+4020F7], 1E
//1byte 단위의 내용[eax+4020F7]을 43과 xor 연산합니다.
00401125 xor byte ptr ds:[eax_4020F7], 55
//1byte 단위의 내용[eax+4020F7]을 43과 xor 연산합니다.
0040112C inc eax
//eax를 1증가 시킵니다.
0040112D loopd short due-cm1.0040110E
//0040110E로 되돌아 갑니다.
0040112F cmp eax, 0
//eax값을 0과 비교합니다.
00401132 jnz short due1-cm1.0040114C
//eax가 0이 아니라면 0040114C로 분기합니다.
//0이라면 분기하지 않고 진행합니다.(0이 아닌 경우는 입력 값이 없는 경우입니다.)

이해를 돕기 위해 실제 과정에 대해 하나하나 짚어가면서 풀어보겠습니다. 우리가 텍스트 박스에 aaaa를 넣었다고 가정합시다. 그러면 aaaa는 앞서 확인했던 GetDlgItemText API로 인해 004020F7 영역의 버퍼에 저장될 것입니다. 확인해보겠습니다.

Address	Hex dump	ASCII
004020F7	61 61 61 61 00 00 00 00 00 00 00 00 00 00 00 00	aaaa
00402107	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

[그림1-8. 입력한 aaaa]

[그림1-8]에서 보이는 것처럼 버퍼에 잘 들어간 것을 확인하였습니다. 그리고 입력한 aaaa 값이 0040110C~00401132의 명령어들을 거치면서 어떻게 변하는지 확인해보겠습니다.

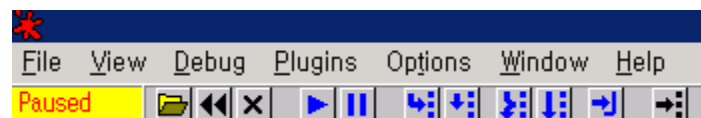
우선 0040110C~00401132에 각각 브레이크포인트를 설정합니다. 브레이크포인트를 설정하는 방법은 브레이크포인트를 설정할 곳에서 F2키를 누르거나 마우스 우 클릭 후 Breakpoint ► Toggle 을 선택하면 됩니다. 그러면 [그림1-4]처럼 주소 부분이 빨간색으로 변하는걸 볼 수 있습니다. 바로 그 상태가 브레이크포인트가 설정된 상태 입니다.



[그림1-9. 우 클릭 팝업메뉴의 브레이크포인트 설정하기]

브레이크포인트를 설정하면 [그림1-7]처럼 보이는 것을 확인할 수 있습니다. 그리고 나서 단축아이콘들 중에 실행버튼을 눌러봅니다. 여러 가지 단축아이콘들이 있는데 차차 설명하

했습니다.



[그림1-10. OllyDbg의 단축 아이콘들]

실행버튼을 누르면 CrackMe 문제 화면이 뜨는걸 볼 수 있습니다. 그러면 텍스트박스에 aaaa를 입력하고 체크버튼을 눌러보겠습니다.

0040110C	, 33C0	xor eax, eax	EAX 초기화
0040110E	> 80B8 F7204000	cmp byte ptr ds:[eax+4020F7], 0	0과 비교해 입력된 값의 유무를 확인
00401115	, 74 18	je short due-cm1,0040112F	값이 없다면 0040112F로 분기
00401117	, 80B0 F7204000	xor byte ptr ds:[eax+4020F7], 43	[주소가 가리키는 곳의 값] xor 43
0040111E	, 80B0 F7204000	xor byte ptr ds:[eax+4020F7], 1E	[주소가 가리키는 곳의 값] xor 1E
00401125	, 80B0 F7204000	xor byte ptr ds:[eax+4020F7], 55	[주소가 가리키는 곳의 값] xor 55
0040112C	, 40	inc eax	EAX = EAX + 1
0040112D	, E2 DF	loopd short due-cm1,0040110E	0040110E로 돌아감
0040112F	> 83F8 00	cmp eax, 0	EAX와 0을 비교
00401132	, 75 18	jnz short due-cm1,0040114C	EAX가 0이 아니면 분기

[그림1-11. 실행 중 브레이크포인트가 설정된 곳에서 일시 정지된 상태]

그러면 [그림1-11]에서 보시는 바와 같이 브레이크포인트가 설정된 곳에서 일시 정지될 것입니다. 그리고 Registers (FPU)를 확인해보면 현재 레지스터의 값을 알 수 있습니다.

Registers (FPU)	
EAX	00000004
ECX	77D121BC USER32,77D121BC
EDX	00140608
EBX	00000000
ESP	0012FBBC
EBP	0012FBC8
ESI	004010D1 due-cm1,004010D1
EDI	0012FC30
EIP	0040110C due-cm1,0040110C

[그림1-12. 0040110C에서 일시 정지된 상태의 Registers (FPU)]

[그림1-12]를 보면 EAX가 000000004인걸 볼 수 있습니다. 0040110C가 진행되면 아마도 EAX는 00000000이 될 것입니다. 왜냐하면 같은 값을 XOR시키면 0이 되기 때문입니다. 코드가 있는 곳에서 F8키를 눌러 한번 Step-Over로 진행 시킨 후 어떤 변화가 생기는지 확인해 보겠습니다.

Registers (FPU)	
EAX	00000000
ECX	77D121BC USER32, 77D121BC
EDX	00140608
EBX	00000000
ESP	0012FBBC
EBP	0012FBC8
ESI	004010D1 due-cm1, 004010D1
EDI	0012FC30
EIP	0040110E due-cm1, 0040110E

[그림1-13. Step-Over 진행 후 Registers (FPU)의 화면]

EAX 레지스터가 00000000이 된 것을 확인할 수 있습니다. [그림1-12]에서 EAX가 00000004인 것은 우리가 입력한 aaaa의 길이가 4이기 때문입니다. 6개의 문자를 입력하면 EAX 레지스터는 분명 00000006이 될 것입니다. 천천히 F8키를 누르면서 Step-Over로 계속 진행하겠습니다.

0040110C	, 33C0	xor eax, eax	EAX 초기화
0040110E	> 80B8 F7204000	cmp byte ptr ds:[eax+4020F7], 0	0과 비교해 입력된 값의 유무를 확인
00401115	, 74 18	je short due-cm1, 0040112F	값이 없다면 0040112F로 분기
00401117	, 80B0 F7204000	xor byte ptr ds:[eax+4020F7], 43	[주소가 가리키는 곳의 값] xor 43
0040111E	, 80B0 F7204000	xor byte ptr ds:[eax+4020F7], 1E	[주소가 가리키는 곳의 값] xor 1E
00401125	, 80B0 F7204000	xor byte ptr ds:[eax+4020F7], 55	[주소가 가리키는 곳의 값] xor 55
0040112C	, 40	inc eax	EAX = EAX + 1
0040112D	, E2 DF	loopd short due-cm1, 0040110E	0040110E로 돌아감
0040112F	> 83F8 00	cmp eax, 0	EAX와 0을 비교
00401132	, 75 18	jnz short due-cm1, 0040114C	EAX가 0이 아니면 분기

[그림1-14. 0040110E로 한 단계 진행된 화면]

한 단계 진행된 상태에서 메모리 내용이 표시되는 부분의 바로 윗부분을 보면 코드에서 참조하는 주소가 가지고 있는 데이터 값과 현재 코드가 어디에서 분기 되어 왔는지 정보를 알려주는 내용이 있습니다.

ds:[004020F7]=61 ('a')		
Jump from 0040112D		
Address	Hex dump	ASCII
004020F7	61 61 61 61 00 00 00 00 00 00 00 00 00 00 00 00	aaaa,.....

[그림1-15. 0040110E 코드의 내용과 버퍼의 내용]

지금 버퍼(004020F7)에 우리가 처음에 입력한 aaaa가 들어 있는 것과 0040110E 코드가 0과 비교하려고 하는 데이터가 0x61 ('a')라는 것을 [그림1-15]를 통해 확인할 수 있습니다. 비교하려는 데이터는 다르기 때문에 00401115에서 분기하지 않을 것입니다. 계속 진행해 보겠습니다.

0040110C	, 33C0	xor eax, eax	EAX 초기화
0040110E	> 80B8 F720400	cmp byte ptr ds:[eax+4020F7], 0	0과 비교해 입력된 값의 유무를 확인
00401115	, 74 18	je short due-cm1,0040112F	값이 없다면 0040112F로 분기
00401117	, 80B0 F720400	xor byte ptr ds:[eax+4020F7], 43	[주소가 가리키는 곳의 값] xor 43
0040111E	, 80B0 F720400	xor byte ptr ds:[eax+4020F7], 1E	[주소가 가리키는 곳의 값] xor 1E
00401125	, 80B0 F720400	xor byte ptr ds:[eax+4020F7], 55	[주소가 가리키는 곳의 값] xor 55
0040112C	, 40	inc eax	EAX = EAX + 1
0040112D	, E2 DF	loopd short due-cm1,0040110E	0040110E로 돌아감
0040112F	> 83F8 00	cmp eax, 0	EAX와 0을 비교
00401132	, 75 18	jnz short due-cm1,0040114C	EAX가 0이 아니면 분기

[그림1-16. 00401115로 한 단계 진행된 화면]

예상했던 대로 분기하지 않았습니다. [그림1-17]을 보면 Jump is NOT taken 문구를 확인할 수 있고, [그림1-16]의 주소 오른쪽의 화살표가 회색으로 분기될 곳을 가리키고 있습니다.

Jump is NOT taken
0040112F=due-cm1,0040112F

[그림1-17. 00401115 코드의 내용]

계속 진행하겠습니다.

0040110C	, 33C0	xor eax, eax	EAX 초기화
0040110E	> 80B8 F720400	cmp byte ptr ds:[eax+4020F7], 0	0과 비교해 입력된 값의 유무를 확인
00401115	, 74 18	je short due-cm1,0040112F	값이 없다면 0040112F로 분기
00401117	, 80B0 F720400	xor byte ptr ds:[eax+4020F7], 43	[주소가 가리키는 곳의 값] xor 43
0040111E	, 80B0 F720400	xor byte ptr ds:[eax+4020F7], 1E	[주소가 가리키는 곳의 값] xor 1E
00401125	, 80B0 F720400	xor byte ptr ds:[eax+4020F7], 55	[주소가 가리키는 곳의 값] xor 55
0040112C	, 40	inc eax	EAX = EAX + 1
0040112D	, E2 DF	loopd short due-cm1,0040110E	0040110E로 돌아감
0040112F	> 83F8 00	cmp eax, 0	EAX와 0을 비교
00401132	, 75 18	jnz short due-cm1,0040114C	EAX가 0이 아니면 분기

[그림1-18. 00401117로 한 단계 진행된 화면]

현재 EAX 레지스터값은 0입니다. 그러면 00401117 코드에서 ds:[eax+4020F7]은 ds:[0+4020F7]이 되고 그것은 버퍼의 첫 번째 자리를 나타냅니다. 0x61 ('a') 입니다. 이것과 0x43을 XOR시키는 게 이번 코드가 하는 일입니다. 계산을 해보니 0x22가 나왔습니다. 버퍼에 찾아가서 바뀌었는지 확인해 보겠습니다.

Address	Hex	dump	ASCII
004020F7	22 61 61 61	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	"aaa....."

[그림1-19. 버퍼의 첫 번째 값이 0x43과 XOR되어 바뀐 화면]

버퍼에서 0x61이 0x22가 된 것을 확인 하였습니다. 그러면 0040111E~00401125의 XOR도 어차피 EAX 레지스터가 0이기 때문에 버퍼의 첫 번째 자리를 XOR하게 될 것입니다. 그럼

[그림1-23. 0040112D 코드의 내용]

[그림1-23]에서 Loop is taken 이라는 내용을 확인할 수 있습니다. 목적지인 0040110E로 분기합니다.

일단 여기까지 일어난 일들을 정리할 필요가 있습니다. 순차적으로 각각의 코드가 진행되었지만 그 모든 게 각자 따로 노는 게 아니라 조화를 이루기 때문입니다.

0040110C에서 EAX 레지스터를 0으로 초기화 시켰습니다. 그리고 0040110E에서 버퍼의 값과 0을 비교해서 분기여부를 결정하였습니다. 이 부분은 상당히 중요합니다. 0040112D에서 0040110E로 무조건 분기하는 Loop에서 중간에 버퍼의 값과 0을 비교해서 언젠가 조건이 만족하면 Loop를 빠져나가기 때문입니다.

00401117~00401125의 연속된 XOR연산도 눈 여겨 볼 필요가 있습니다. 어떤 값을 XOR 연산시킨 후 다시 거꾸로 XOR시키면 원형으로 복원 할 수 있게 됩니다. 여기서 XOR연산에 대해서 언급한 이유는 마지막에 프로그래밍을 통해서 확인 하도록 하겠습니다. 그렇게 XOR 연산을 3번 끝내고 난 후 0040112C에서 EAX 레지스터를 1증가 시키게 되는데 이것은 버퍼의 다음 번째 데이터를 읽어오기 위함입니다.

정리해보면 버퍼의 데이터를 한 개 한 개 읽어오면서 3번 XOR시키고 버퍼에서 읽어오는 값이 0이라면 분기하여 Loop 밖으로 빠져나갑니다. 그리고 다른 코드를 실행할 것입니다.

정리도 끝난 거 같으니 계속 진행하겠습니다.

Address	Hex dump	ASCII
004020F7	69 69 69 69 00 00 00 00 00 00 00 00 00 00 00 00	iiii.....

[그림1-24. loop가 끝난 후 버퍼의 데이터]

F8키로 계속 진행을 하면 버퍼의 4번째 데이터를 3번 XOR시킨 후 EAX 레지스터를 1증가 시켜서 EAX 레지스터가 00000004가 되면 버퍼의 5번째 값인 00을 가지고 Loop를 돌게 됩니다.

그러면 0040110E에서 버퍼의 5번째 값과 0을 비교하는데 이것은 조건이 만족되므로 00401115에서 0040112F로 분기하게 됩니다.

0040110C	, 33C0	xor eax, eax	EAX 초기화
0040110E	> 80B8 F7204000	cmp byte ptr ds:[eax+4020F7], 0	0과 비교해 입력된 값의 유무를 확인
00401115	, 74 18	je short due-cm1,0040112F	값이 없다면 0040112F로 분기
00401117	, 80B0 F7204000	xor byte ptr ds:[eax+4020F7], 43	[주소가 가리키는 곳의 값] xor 43
0040111E	, 80B0 F7204000	xor byte ptr ds:[eax+4020F7], 1E	[주소가 가리키는 곳의 값] xor 1E
00401125	, 80B0 F7204000	xor byte ptr ds:[eax+4020F7], 55	[주소가 가리키는 곳의 값] xor 55
0040112C	, 40	inc eax	EAX = EAX + 1
0040112D	, E2 DF	loopd short due-cm1,0040110E	0040110E로 돌아감
0040112F	> 83F8 00	cmp eax, 0	EAX와 0을 비교
00401132	, 75 18	jnz short due-cm1,0040114C	EAX가 0이 아니면 분기
00401134	, 68 00200000	push 2000	Style = MB_OK MB_TASKMODAL
00401139	, 68 01204000	push due-cm1,00402001	Title = "Duelist's Crackme #1"
0040113E	, 68 9D204000	push due-cm1,0040209D	Text = "Couldn't validate code, bec
00401143	, 6A 00	push 0	hOwner = NULL
00401145	, E8 A5010000	call <jmp,&USER32,MessageBoxA>	MessageBoxA

[그림1-25. 0040112F로 분기한 화면]

0040112F에서 EAX 레지스터의 값을 0과 비교하여 00401132에서 조건분기하고 있는걸 [그림1-24]에서 볼 수 있습니다. 00401132의 jnz명령은 0040112F의 cmp명령 왼쪽의 인자 값이 0이 아니라면 분기하라는 명령입니다. 그리고 jnz는 Jump Not Zero의 약자 인데 여기서 말하는 Zero는 Registers (FPU)의 Z를 말합니다.

```
Registers (FPU)
EAX 00000004
ECX 77D121B8 USER32,77D121B8
EDX 00140608
EBX 00000000
ESP 0012FBBC
EBP 0012FBC8
ESI 004010D1 due-cm1,004010D1
EDI 0012FC30
EIP 00401132 due-cm1,00401132
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
```

[그림1-26. Registers (FPU) 화면]

Registers (FPU)에 보이는 Z는 Zero FLAG를 말하는데 결과가 zero임을 가리킬 때 사용됩니다. (Register나 C, P, A, Z, S, T, D 같은 FLAG들에 대한 설명은 이 문서에 담기보다 다른 문서를 참고하시는 게 좋을 것 같아서 이것들에 대한 세부적인 내용은 담지 않겠습니다.)

0040112F cmp eax, 0은 EAX 레지스터가 0이 아니기 때문에 Zero FLAG에 0을 반환하며 분기하게 됩니다.

0040112F	>	83F8 00	cmp eax, 0	EAX와 0을 비교
00401132	>	75 18	jnz short due-cm1,0040114C	EAX가 0이 아니면 분기
00401134	.	68 00200000	push 2000	[Style = MB_OK MB_TASKMODAL Title = "Duelist's Crackme #1" Text = "Couldn't validate code, hOwner = NULL MessageBoxA Arg3 = 00000024 Arg2 = 004020D3 Arg1 = 004020F7 ASCII "iiii" due-cm1,004011C1]
00401139	.	68 01204000	push due-cm1,00402001	
0040113E	.	68 9D204000	push due-cm1,0040209D	
00401143	.	6A 00	push 0	
00401145	.	E8 A5010000	call <jmp,&USER32,MessageBoxA>	
0040114A	.	EB A8	jmp short due-cm1,004010F4	
0040114C	>	6A 24	push 24	
0040114E	.	68 D3204000	push due-cm1,004020D3	
00401153	.	68 F7204000	push due-cm1,004020F7	
00401158	.	E8 64000000	call due-cm1,004011C1	

[그림1-27. 00401132의 분기]

앞서 설명했듯이 분기조건이 성립하므로 빨간색으로 화살표가 출발지에서 목적지를 가리키고 있습니다. F8키를 눌러서 진행하면 00401134 코드로 진행되는 것이 아니라 0040114C 코드로 진행되게 됩니다. 만약 0040112F에서 EAX 레지스터의 값이 0이었다면 조건이 성립하지 않아 바로 00401134 코드로 진행되어 우리에게 입력된 값이 없다는 메시지박스를 보여 줬을 것입니다.

```

[
Style = MB_OK|MB_TASKMODAL
Title = "Duelist's Crackme #1"
Text = "Couldn't validate code, because it wasn't entered..."
hOwner = NULL
MessageBoxA
]

```

[그림1-28. 입력된 값이 없을 경우의 메시지박스]

그럼 F8키를 눌러서 계속 진행하겠습니다.

0040114C	>	6A 24	push 24	Arg3 = 00000024
0040114E	,	68 D3204000	push due-cm1,004020D3	Arg2 = 004020D3
00401153	,	68 F7204000	push due-cm1,004020F7	Arg1 = 004020F7 ASCII "iiii"
00401158	,	E8 64000000	call due-cm1,004011C1	due-cm1,004011C1

[그림1-29. 0040114C로 한 단계 진행된 화면]

00401158의 call 명령이 진행되기 전까지 3개의 값을 인자로 사용하기 위해서 Stack에 PUSH하는 것 같습니다. 첫 번째 값은 0x24, 두 번째 값은 004020D3가 가리키는 값, 세 번째 값은 004020F7이 가리키는 값입니다. 그러고 보니 세 번째 값이 우리가 입력했던 버퍼의 주소입니다. 그러면 0040114E가 가리키는 값은 무엇인지 궁금해집니다. 확인해보겠습니다.

Address	Hex dump	ASCII
004020D3	7B 61 65 78 64 6D 26 68 7A 69 6B 63 65 6D 26 3C	{aexdm&kzikcem&<
004020E3	26 66 6D 7F 6A 61 6D 7B 26 6A 71 26 6C 7D 6D 64	&fmdjam{&jq&l}md
004020F3	61 7B 7C 00 69 69 69 69 00 00 00 00 00 00 00 00	a{!.iiii.....
00402103	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

[그림1-30. 004020D3의 데이터와 버퍼에 입력하여 XOR로 변경된 데이터]

주소가 가리키는 곳의 데이터를 확인하였으니 다시 코드로 돌아가 진행하겠습니다.

0040114C	>	6A 24	push 24	Arg3 = 00000024
0040114E	,	68 D3204000	push due-cm1,004020D3	Arg2 = 004020D3
00401153	,	68 F7204000	push due-cm1,004020F7	Arg1 = 004020F7 ASCII "iiii"
00401158	,	E8 64000000	call due-cm1,004011C1	due-cm1,004011C1

[그림1-31. F7키를 사용해야 할 때]

코드를 F8키(Step-over)로 진행하다가 00401158과 같은 호출문을 만났을 때 호출문 안으로 들어가서 진행하려면 F7키(Step-into)를 이용해야 합니다. 그럼 F7키를 이용해서 호출문 안에 무엇이 있는지 확인해 보겠습니다.

004011C1	8B 000000	enter 0, 0	due-cm1,004011C1
004011C5	B8 01000000	mov eax, 1	
004011CA	8B7D 08	mov edi, [arg.1]	
004011CD	8B75 0C	mov esi, [arg.2]	
004011D0	8B4D 10	mov ecx, [arg.3]	
004011D3	F3:A6	repe cmps byte ptr es:[edi], byte ptr ds:[esi]	비교해야되는 값과 입력된 값을 비교
004011D5	67:E3 05	jcxz short due-cm1,004011D0	
004011D8	B8 00000000	mov eax, 0	
004011DD	C9	leave	
004011DE	C2 0C00	ret 0C	

[그림1-32. 00401158 코드에서 F7키를 타고 들어온 call 내부]

004011C5 mov eax, 1

//EAX Register에 1을 복사합니다. (기존의 내용은 삭제 됩니다.)

004011CA mov edi, [arg.1]

//EDI Register에 004020F7을 복사합니다.

004011CD mov esi, [arg.2]

//ESI Register에 004020D3을 복사합니다.

004011D0 mov ecx, [arg.3]

//ECX Register에 00000024를 복사합니다.

004011D3 repe cmps byte ptr es:[edi], byte ptr ds:[esi]

//REPE(Repeat until Equal)과 CMPS(Compare String)을 조합한 명령어

//일치하는 데이터가 얻어질 때까지 메모리상의 데이터를 탐색합니다.

//우리가 입력한 버퍼(004020F7)의 데이터와 004020D3의 데이터를 비교합니다.

//비교할 데이터가 같다면 004011D5에서 004011DD로 분기하며 그렇지 않으면 분기 없이

//진행하며, 호출문이 끝나면 0040115D로 돌아갑니다.

0040115D	83F8 00	cmp eax, 0	
00401160	74 1B	je short due-cm1,0040117D	
00401162	68 00200000	push 2000	Style = MB_OK MB_TASKMODAL
00401167	68 01204000	push due-cm1,00402001	Title = "Duelist's Crackme #1"
0040116C	68 17204000	push due-cm1,00402017	Text = "Congratulations! Please
00401171	6A 00	push 0	hOwner = NULL
00401173	E8 77010000	call <jmp.&USER32,MessageBoxA>	MessageBoxA
00401178	E9 77FFFFFF	jmp due-cm1,004010F4	
0040117D	68 00200000	push 2000	Style = MB_OK MB_TASKMODAL
00401182	68 01204000	push due-cm1,00402001	Title = "Duelist's Crackme #1"
00401187	68 63204000	push due-cm1,00402063	Text = "You've entered an invali
0040118C	6A 00	push 0	hOwner = NULL
0040118E	E8 5C010000	call <jmp.&USER32,MessageBoxA>	MessageBoxA

[그림1-33. 호출문이 끝난 후 0040115D로 돌아온 화면]

0040115D에서 EAX 레지스터가 0과 같다면 00401160에서 분기조건을 만족시키므로 0040117D로 분기하게 됩니다. 0040117D로 분기하여 진행이 되면 잘못된 Serial을 입력했다는 메시지박스를 보게 됩니다. 만약 분기하지 않고 00401162로 바로 진행을 하게 되면 축하한다는 메시지박스를 보게 됩니다. EAX 레지스터가 최근에 변한 곳을 찾아보면 004011D3에서 문자열을 비교한 후에 그 결과에 따라 004011D8에서 1이었던 EAX 레지스터를 0으로 만들 수 있습니다. 즉 [그림1-32] 004011D3에서 문자열을 비교할 때 틀리지 않고 비교를 마

친다면 성공메시지를 볼 수 있게 되는 것입니다.

0x0003 결론

분석과 풀이가 모두 끝났으니 결론을 맺어야 합니다. 004011D3에서 문자열을 비교하는데 우리가 입력한 데이터가 아닌 비교대상이 되는 데이터가 바로 CrackMe가 요구하는 데이터인 것을 알 수 있습니다. 004020D3이 가리키는 곳에 있는 데이터인데 이것을 추출하여 바로 텍스트박스에 넣고 체크할 수는 없습니다. 왜냐하면 [그림1-11]에 보이는 00401117~00401125 코드에서 입력된 데이터를 0x42, 0x1E, 0x55와 XOR시키기 때문입니다.

그렇다면 앞에서 언급했듯이 비교대상인 004020D3의 데이터를 0x42, 0x1E, 0x55와 거꾸로 XOR시키면 우리가 찾고자 하는 값이 나오지 않을까 추측 할 수 있습니다. [그림1-30]에서 0x7B~0x7C까지가 비교대상 문자열입니다. 프로그래밍을 통해서 그것을 0x42, 0x1E, 0x55와 거꾸로 XOR시켜서 출력해보겠습니다.

```
#include <stdio.h>
int main()
{
    int a[35] = {0x7B,0x61,0x65,0x78,0x64,0x6D,0x26,
                0x6B,0x7A,0x69,0x6B,0x63,0x65,0x6D,
                0x26,0x3C,0x26,0x66,0x6D,0x7F,0x6A,
                0x61,0x6D,0x7B,0x26,0x6A,0x71,0x26,
                0x6C,0x7D,0x6D,0x64,0x61,0x7B,0x7C};

    int b[35];

    for (int i = 0; i < 35; i++){
        b[i] = 0x43 ^ a[i];
        b[i] = 0x1e ^ b[i];
        b[i] = 0x55 ^ b[i];
    }
```



```
        printf("%c", b[i]);  
    }  
    return 0;  
}
```

Serial : simple.crackme.4.newbies.by.duelist

0x02 Duelist #2

Crackme: <http://beist.org/research/public/crackme10/duel-cm2.zip>

Keygen: -

Author: Duelist

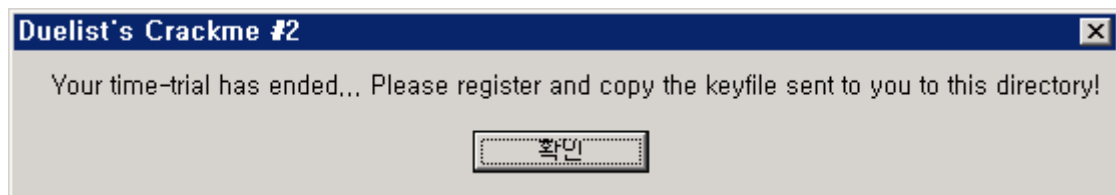
Level: ★

Protection: Time-Trial

0x0001 목표

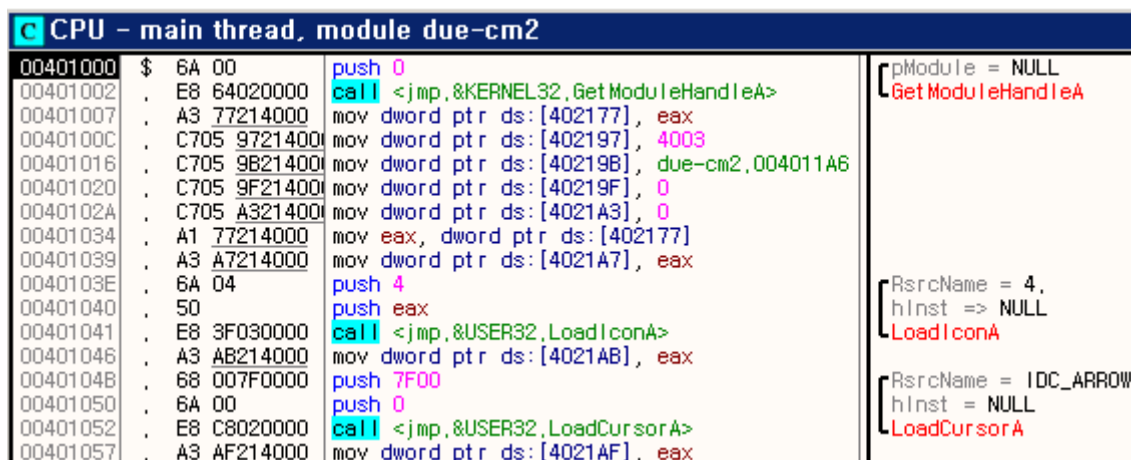
이번 CrackMe에서는 CreateFile같은 API와 복합적인 연산을 찾아 특정 조건을 만족시키는 Keyfile을 만들어야 합니다.

0x0002 분석 및 풀이



[그림2-1. Duelist's Crackme #2 실행화면]

처음에 Duelist's Crackme #2를 실행하면 Keyfile을 현재 디렉토리에 넣으라는 메시지를 볼 수 있습니다. 어떤 파일명의 어떤 내용을 가진 Keyfile을 요구하는지 알아야 할 것 같습니다. 그럼 OllyDbg를 이용해 Duelist's Crackme #2를 열어 보겠습니다.



[그림2-2. Ollydbg로 열었을 때 보이는 코드들]

조금씩 내려가면서 확인해보겠습니다.

0040105C	. 6A 00	push 0	hTemplateFile = NULL
0040105E	. 68 6F214000	push due-cm2,0040216F	Attributes = READONLY HIDDEN SYSTEM ARCHIVE TEMPORARY 402048
00401063	. 6A 03	push 3	Mode = OPEN_EXISTING
00401065	. 6A 00	push 0	pSecurity = NULL
00401067	. 6A 03	push 3	ShareMode = FILE_SHARE_READ FILE_SHARE_WRITE
00401069	. 68 000000C0	push C0000000	Access = GENERIC_READ GENERIC_WRITE
0040106E	. 68 79204000	push due-cm2,00402079	FileName = "due-cm2.dat"
00401073	. E8 0B020000	call <jmp,8KERNEL32,CreateFileA>	CreateFileA
00401078	. 83F8 FF	cmp eax, -1	due-cm2.dat 라는 keyfile 이 필요하다는것

[그림2-3. CreateFile API]

CreateFile API는 다음과 같습니다.

```

HANDLE CreateFile(
    LPCTSTR lpFileName, //열거나 만들고자 하는 파일의 완전경로를 문자열로 지정
    DWORD dwDesiredAccess, //파일에 대한 액세스 권한을 지정
    DWORD dwShareMode, //열려진 파일의 공유 모드를 지정
    LPSECURITY_ATTRIBUTES lpSecurityAttributes, //파일의 보안 속성을 지정하는
    SECURITY_ATTRIBUTES 구조체의 포인터
    DWORD dwCreationDisposition, //만들고자 하는 파일이 이미 있거나 또는 열고자 하는
    파일이 없을 경우의 처리를 지정
    DWORD dwFlagsAndAttributes, //파일의 속성과 여러 가지 옵션 설정
    HANDLE hTemplateFile//생성될 파일의 속성을 제공할 템플릿 파일
);
    
```

OlyDbg에서 자동으로 해석되는 Comment에서 확인한 FileName은 "due-cm2.dat"입니다. Access는 GENERIC_READ|GENERIC_WRITE입니다. 읽기와 쓰기가 가능합니다. ShareMode는 FILE_SHARE_READ+FILE_SHARE_WRITE입니다. 이것도 Access권한과 마찬가지로 읽기와 쓰기가 가능합니다. pSecurity는 NULL이므로 사용하지 않는다는 것을 알 수 있습니다. Mode는 OPEN_EXISTING입니다. due-cm2.dat파일을 열되 만약 파일이 없으면 에러 코드를 리턴합니다. 여기서 우리는 due-cm2.dat라는 이름을 가진 파일이 Keyfile임을 확인할 수 있습니다. Attributes는 READONLY|HIDDEN|SYSTEM|ARCHIVE|TEMPORARY입니다. 모든 속성을 or연산자로 묶어놓았습니다. hTemplateFile은 NULL이므로 사용하지 않는다는 것을 알 수 있습니다.

일단 due-cm2.dat라는 파일을 생성한 후 CrackMe를 실행하면 [그림2-4]처럼 잘못된 Keyfile이라는 메시지를 확인할 수 있습니다.



[그림2-4. due-cm2.dat파일 생성 후 CrackMe실행화면]

00401078	, 83F8 FF	cmp eax, -1	파일의 존재여부 확인
0040107B	, 75 1D	jnz short due-cm2,0040109A	파일이 존재한다면 분기
0040107D	, 6A 00	push 0	Style = MB_OK MB_APPLMODAL
0040107F	, 68 01204000	push due-cm2,00402001	Title = "Duelist's Crackme #2"
00401084	, 68 17204000	push due-cm2,00402017	Text = "Your time-trial has ended,
00401089	, 6A 00	push 0	hOwner = NULL
0040108B	, E8 D7020000	call <jmp,&USER32,MessageBoxA>	MessageBoxA
00401090	, E8 24020000	call <jmp,&KERNEL32,ExitProcess>	ExitProcess
00401095	, E9 28010000	jmp due-cm2,004011C2	
0040109A	> 6A 00	push 0	pOverlapped = NULL
0040109C	, 68 73214000	push due-cm2,00402173	pBytesRead = due-cm2,00402173
004010A1	, 6A 46	push 46	BytesToRead = 46 (70,)
004010A3	, 68 1A214000	push due-cm2,0040211A	Buffer = due-cm2,0040211A
004010A8	, 50	push eax	hFile
004010A9	, E8 2F020000	call <jmp,&KERNEL32,ReadFile>	ReadFile
004010AE	, 85C0	test eax, eax	그리고 파일을 읽는다는것

[그림2-5. ReadFile API]

00401078이전 부분은 CreateFile API부분입니다.

due-cm2.dat파일의 존재여부를 00401078~0040107B에서 확인하여 파일이 존재한다면 Read API로 분기합니다. 만약 due-cm2.dat 파일이 존재하지 않는다면 바로 밑의 메시지박스를 호출합니다. [그림2-1]과 같은 내용인걸 확인할 수 있습니다. 존재한다면 0040109A로 분기하여 ReadFile API를 호출합니다.

ReadFile API는 다음과 같습니다.

```

BOOL ReadFile(
    HANDLE hFile, //읽고자 하는 파일의 핸들
    LPVOID lpBuffer, //읽는 데이터를 저장할 버퍼의 포인터
    DWORD nNumberOfBytesToRead, //읽고자 하는 바이트 수
    LPDWORD lpNumberOfBytesRead, //실제로 읽은 바이트 수를 리턴 받기 위한 출력용 인
수
    LPOVERLAPPED lpOverlapped //비동기 입출력을 구조체의 포인터
);

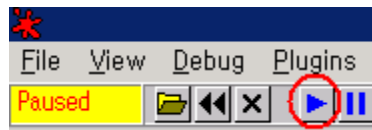
```

0040211A를 버퍼로 쓰는 것을 확인할 수 있고, 읽고자 하는 바이트 수는 0x46입니다. 과연 0x46바이트만큼 읽고 0040211A를 버퍼로 사용하는지 확인해보겠습니다.



[그림2-6. Keyfile에 값을 입력한 화면]

[그림2-6]처럼 값을 넣고 [그림2-7]의 실행버튼을 눌러보겠습니다.



[그림2-7. 왼쪽부터 열기, 다시 읽기, 중지, 실행, 일시 정지 아이콘]

Address	Hex dump	ASCII
0040211A	4F 73 69 72 69 73 20 69 73 20 48 61 6E 64 73 6F	Osiris is Handsome
0040212A	6D 65 00 00 00 00 00 00 00 00 00 00 00 00 00	me.....
0040213A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

[그림2-8. 버퍼에 들어간 값]

0040211A를 찾아가보니 값이 제대로 들어간 것을 확인했습니다. 그럼 이제 ReadFile API 이후의 코드들을 살펴보겠습니다.

004010B4	> 33DB	xor ebx, ebx	EBX 초기화
004010B6	. 33F6	xor esi, esi	ESI 초기화
004010B8	. 833D 73214001	cmp dword ptr ds:[402173], 12	KEY값의 문자의 개수를 0x12(18)와 비교
004010BF	. 7C 36	jl short due-cm2,004010F7	0x12(18)보다 작다면 분기

[그림2-9. ReadFile API이후의 코드들]

```
004010B4 xor ebx, ebx
//ebx를 초기화 시킵니다.
004010B6 xor esi, esi
//esi를 초기화 시킵니다.
004010B8 cmp dword ptr ds:[402173], 12
//입력된 버퍼의 크기를 0x12(18)와 비교합니다.
004010BF jl short due-cm2,004010F7
//0x12(18)보다 작다면 004010F7로 분기합니다.
```

Keyfile에 입력된 문자가 18개 미만일 경우 잘못된 메시지([그림2-4])를 보여주는 곳으로 바로 분기합니다. 확인해보겠습니다.

004010B8	. 833D 73214001	cmp dword ptr ds:[402173], 12	KEY값의 문자의 개수를 0x12(18)와 비교
		ds:[00402173]=00000011	

[그림2-10. 004010B8 코드와 그 내용]

Keyfile에 17개의 문자를 넣고 실행하였습니다. [그림2-10]에서 보이는 것처럼 0x11(17)과 0x12(18)를 비교하고 있습니다. 0x11(17)은 0x12(18)보다 작기 때문에 004010BF에서 분기조건을 만족시키므로 004010F7(Keyvalue가 틀렸다는 메시지)로 분기하게 됩니다. 그리고 Keyfile에 18개 이상의 문자가 들어가 004010BF에서 분기조건을 만족시키지 않는다면 분기

문은 실행되지 않고 그 다음인 004010C1 코드를 진행하게 됩니다.

004010B8	. 833D 73214000	cmp dword ptr ds:[402173], 12	KEY값의 문자의 개수를 0x12(18)와 비교
004010BF	. 7C 36	j1 short due-cm2.004010F7	0x12(18)보다 작다면 분기
004010C1	> 8A83 1A214000	mov al, byte ptr ds:[ebx+40211A]	1byte 단위로 버퍼의 값을 AL로 복사
004010C7	. 3C 00	cmp al, 0	AL과 0을 비교
004010C9	. 74 08	je short due-cm2.004010D3	AL이 0이라면 분기
004010CB	. 3C 01	cmp al, 1	AL과 1을 비교
004010CD	. 75 01	jnz short due-cm2.004010D0	AL이 1이면 분기
004010CF	. 46	inc esi	ESI = ESI + 1
004010D0	> 43	inc ebx	EBX = EBX + 1
004010D1	. ^ EB EE	jmp short due-cm2.004010C1	004010C1으로 분기
004010D3	> 83FE 02	cmp esi, 2	ESI와 2를 비교
004010D6	. 7C 1F	j1 short due-cm2.004010F7	ESI가 2보다 작다면 분기
004010D8	. 33F6	xor esi, esi	ESI를 초기화
004010DA	. 33DB	xor ebx, ebx	EBX를 초기화
004010DC	> 8A83 1A214000	mov al, byte ptr ds:[ebx+40211A]	1byte 단위로 버퍼의 값을 AL로 복사합니다
004010E2	. 3C 00	cmp al, 0	AL과 0을 비교합니다
004010E4	. 74 09	je short due-cm2.004010EF	AL이 0이라면 분기합니다
004010E6	. 3C 01	cmp al, 1	AL과 1을 비교합니다
004010E8	. 74 05	je short due-cm2.004010EF	AL이 1이라면 분기합니다
004010EA	. 03F0	add esi, eax	ESI = ESI + EAX
004010EC	. 43	inc ebx	EBX = EBX + 1
004010ED	. ^ EB ED	jmp short due-cm2.004010DC	004010DC로 분기합니다
004010EF	> 81FE 05010000	cmp esi, 105	ESI를 0x105(469)와 비교합니다.
004010F5	. 74 10	je short due-cm2.00401114	ESI와 0x105(469)가 같다면 분기

[그림2-11. 004010B8~004010F5 코드]

```

004010C1 mov al, byte ptr ds:[ebx+40211A]
//AL에 1byte 단위로 [ebx+40211A]의 내용을 복사합니다.
004010C7 cmp al, 0
//복사된 AL과 0을 비교합니다.
004010C9 je short due-cm2.004010D3
//AL이 0과 같다면 004010D3으로 분기합니다.
004010CB cmp al, 1
//AL과 1을 비교합니다.
004010CD jnz short due-cm2.004010D0
//AL이 1이 아니면 004010D0로 분기합니다.
004010CF inc esi
//ESI Register를 1증가 시킵니다.
004010D0 inc ebx
//EBX Register를 1증가 시킵니다.
004010D1 jmp short due-cm2.004010C1
//004010C1으로 분기합니다.
004010D3 cmp esi, 2
//ESI Register의 값과 2를 비교한다.
004010D6 jl short due-cm2.004010F7
//ESI Register의 값이 2보다 작다면 분기한다.
004010D8 xor esi, esi
//ESI Register를 초기화 시킨다.

```

```

004010DA xor ebx, ebx
//EBX Register를 초기화 시킨다.
004010DC mov al, byte ptr ds:[ebx+40211A]
//AL에 1byte 단위로 [ebx+40211A]의 내용을 복사합니다.
004010E2 cmp al, 0
//AL과 0을 비교합니다.
004010E4 je short due-cm2.004010EF
//AL과 0이 같다면 004010EF로 분기합니다.
004010E6 cmp al, 1
//AL과 1을 비교합니다.
004010E8 je short due-cm2.004010EF
//AL과 1이 같다면 004010EF로 분기합니다.
004010EA add esi, eax
//ESI Register에 EAX Register의 값을 더합니다.
004010EC inc ebx
//EBX Register를 1증가시킵니다.
004010ED jmp short due-cm2.004010DC
//004010DC로 분기합니다.
004010EF cmp esi, 1D5
//ESI Register와 0x1D5(469)를 비교합니다.
004010F5 je short due-cm2.00401114
//ESI Register값과 0x1D5(469)가 같다면 00401114로 분기합니다.

```

[그림2-9]에서 보면 004010B4~B6에 ESI와 EBX Register를 초기화 시키는 코드가 있습니다. 그리고 004010B8~BF를 보면 Keyfile의 내용이 0x12(18)이하일 경우 에러 메시지로 분기하는 코드가 있습니다. 일단 에러 메시지를 보지 않으려면 최소한 Keyfile안의 문자가 0x12(18)개 이상 되어야 된다는 것을 알 수 있습니다. 그리고 에러메시지를 보지 않기 위해서 조건을 만족하여 반드시 분기하여야 한다는 것을 알 수 있습니다. 조건을 만족시키지 않고 분기하지 않으면 바로 에러 메시지를 호출하게 됩니다. 일단 두 가지 조건을 생각하며 주석 달린 코드와 함께 진행하겠습니다.

[그림2-5]를 보면 0040211A를 버퍼로 쓰는 걸 확인할 수 있습니다. 그리고 004010DC를 보면 1byte단위로 [ebx+40211A]의 내용을 al로 복사하는 코드가 있습니다. 004010B4~B6에서 ESI와 EBX Register가 초기화 되었으므로 [ebx+40211A]는 [0+40211A]가 됩니다.

EBX Register는 1씩 증가하여 Keyfile의 내용이 저장된 버퍼의 내용을 불러와 al에 저장할 것입니다. 004010C7에서 al에 저장된 내용을 0과 비교합니다. 그리고 004010C9에서 조건분

기를 하게 되는데 004010C7에서 비교한 al이 0이라면 004010D3으로 분기하게 됩니다.

분기하여 온 004010D3을 보면 ESI Register와 0x2를 비교합니다. 그리고 004010D6에서 조건분기로 ESI Register가 2보다 작을 경우에 004010F7(에러 메시지)로 분기하게 됩니다. 에러 메시지로 분기하지 않기 위해서 ESI는 2보다 커야 될 것 입니다. 여기서 한가지 조건을 더 찾았습니다. ESI Register는 0x2보다 커야 된다는 것 입니다. EBX와 함께 초기화된 ESI Register가 0x2이상 증가 되려면 ESI Register를 증가시켜주는 코드가 있어야 됩니다. 004010CF가 바로 그 코드입니다. 그래서 최소한 Keyfile의 처음 두 문자는 0이 되어서는 안 된다는 걸 알 수 있습니다. 그럼 이제 004010C1~004010D1이후의 코드를 보도록 하겠습니다.

004010C9에서 분기해서 004010D3으로 오면 ESI Register가 2보다 작은지 확인하게 됩니다. 만약 작다면 004010D6에서 004010F7(에러 메시지)로 분기하게 되고, 작지 않다면 004010D8~004010DA에서 ESI와 EBX Register를 초기화 시킵니다. 그리고 나서 004010DC에서 또다시 1byte단위로 al에 [ebx+40211A]의 내용을 복사합니다. 004010E2에서 al과 0을 비교하여 004010E4에서 조건이 성립하면 004010EF로 분기합니다. 또 004010E6에서 al과 1을 비교하여 004010E8에서 조건이 성립하면 004010EF로 분기합니다. 004010EF에서는 ESI Register의 값을 0x1D5(469)와 비교를 합니다. 그리고 004010F5에서 ESI Register가 0x1D5(469)라면 00401114로 분기하게 됩니다. 만약 00401114로 분기하지 않고 그대로 진행된다면 에러 메시지를 보게 됩니다. 그러므로 어떻게든 004010EF 코드에서 ESI Register는 0x1D5(469)가 되어야 합니다.

<pre> 004010EF > 81FE D5010001 cmp esi, 1D5 004010F5 > 74 1D je short due-cm2,00401114 004010F7 > 6A 00 push 0 004010F9 > 68 01204000 push due-cm2,00402001 004010FE > 68 86204000 push due-cm2,00402086 00401103 > 6A 00 push 0 00401105 > E8 50020000 call <jmp.&USER32.MessageBoxA> 0040110A > E8 AA010000 call <jmp.&KERNEL32.ExitProcess> 0040110F > E9 AE000000 jmp due-cm2,004011C2 00401114 > 33F6 xor esi, esi </pre>	<pre> ESI를 0x1D5(469)와 비교합니다. ESI와 0x1D5(469)가 같다면 분기 Style = MB_OK MB_APPLMODAL Title = "Duelist's Crackme #2" Text = "Your current keyfile is hOwner = NULL MessageBoxA ExitProcess 4011C2로 분기 ESI를 초기화 </pre>
--	--

[그림2-12. 004010EF~00401114 코드]

ESI Register가 0x1D5(469)가 되기 위한 코드는 [그림2-11]에 004010DC~004010ED입니다. ESI Register에 EAX Register를 더하는데 ESI Register의 값이 몇 번의 루프(004010DC와 004010ED를 말함)를 진행하여 0x1D5(469)가 되면 al이 0(004010E2~004010E4) 또는 1(004010E6~004010E8)이 되면 루프를 빠져 나와야 합니다. 여기서 0또는 1은 Keyfile의 내용을 말합니다. 문자로서의 0(0x30)과 1(0x31)이 아닌 Hex로 0x00, 0x01값을 가지는 ASCII의 NULL(0x00), SOH(0x01)을 말합니다. 그런데 [그림2-11]의 004010DC~004010ED의 코드가 진행되기 위해서는 004010C1~004010D1에서 ESI Register의 값이 2이상 되는 조건을 만족해야 합니다. al에는 Keyfile의 내용이 차례대로 읽혀오므로 조건을 만족하기 전까지 0x00, 0x01이 아닌 더해서 0x1D5(469)가 되는 2개 이상의 값이 있어야 합니다. 그 값을 저는

0xEA(234)와 0xEB(235)로 정하겠습니다. 그리고 004010DC~004010ED의 루프를 빠져나가기 위해서 0xEA, 0xEB 다음에 0x01을 넣겠습니다. 그러면 004010EF~004010F5의 분기조건을 만족시키므로 00401114로 분기하게 됩니다. 그래서 다음 그림과 같은 코드를 진행하게 됩니다.

00401114	> 33F6	xor esi, esi	ESI를 초기화
00401116	> 43	inc ebx	EBX = EBX + 1
00401117	, 8A83 1A214000	mov al, byte ptr ds:[ebx+40211A]	1byte 단위로 버퍼의 값을 AL로 복사
0040111D	, 3C 00	cmp al, 0	AL과 0을 비교
0040111F	, 74 18	je short due-cm2,00401139	AL이 0이라면 분기
00401121	, 3C 01	cmp al, 1	AL과 1을 비교
00401123	, 74 14	je short due-cm2,00401139	AL이 1이라면 분기
00401125	, 83FE 0F	cmp esi, 0F	ESI와 0x0F(15)를 비교
00401128	, 73 0F	jnb short due-cm2,00401139	원쪽의 인자값이 오른쪽 값보다 작지 않으면(크거나 같으면) 분기
0040112A	, 3286 1A214000	xor al, byte ptr ds:[esi+40211A]	AL = AL ^ 버퍼
00401130	, 8986 60214000	mov dword ptr ds:[esi+402160], eax	402160에 XOR된 값을 복사
00401136	, 46	inc esi	ESI = ESI + 1
00401137	, EB D0	jmp short due-cm2,00401116	401116으로 분기
00401139	> 43	inc ebx	EBX = EBX + 1
0040113A	, 33F6	xor esi, esi	ESI초기화
0040113C	> 8A83 1A214000	mov al, byte ptr ds:[ebx+40211A]	1byte 단위로 버퍼의 값을 AL로 복사
00401142	, 3C 00	cmp al, 0	AL과 0을 비교
00401144	, 74 09	je short due-cm2,0040114F	AL이 0과 같다면 분기
00401146	, 3C 01	cmp al, 1	AL과 1을 비교
00401148	, 74 F2	je short due-cm2,0040113C	AL과 1이 같다면 분기
0040114A	, 03F0	add esi, eax	ESI = ESI + EAX
0040114C	, 43	inc ebx	EBX = EBX + 1
0040114D	, EB ED	jmp short due-cm2,0040113C	40113C로 분기
0040114F	> 81FE B2010000	cmp esi, 1B2	ESI와 0x1B2(434)를 비교
00401155	, 75 A0	jnz short due-cm2,004010F7	ESI가 0x1B2(434)가 아니면 분기

[그림2-13. 00401114~00401155 코드]

00401114에서 ESI Register가 초기화 됩니다. 그리고 00401116에서 EBX Register의 값을 1증가시킵니다. 00401117의 코드는 버퍼에 저장된 Keyfile의 값을 al로 읽어오는 것입니다. 그리고 EBX Register의 값이 현재코드에 오기까지 변경된 것이 없으므로 EBX Register의 값은 2입니다. 그래서 그 값에 1을 더 증가시키므로 EBX Register가 3이 되고 00401117의 코드에서 al에 복사하려고 버퍼에서 읽어오는 것은 Keyfile의 4번째 값입니다. 아직 우리는 4번째 값을 정하지 않았습니다. 우리가 정하지 않은 4번째 값을 "4번째 값"이라고 정하고 진행하겠습니다. 0040111D~00401128의 코드를 보면 3개의 분기문이 존재하는데 3개 모두 00401139로 분기하게 되어 있습니다.

0040111D	, 3C 00	cmp al, 0	AL과 0을 비교
0040111F	, 74 18	je short due-cm2,00401139	AL이 0이라면 분기
00401121	, 3C 01	cmp al, 1	AL과 1을 비교
00401123	, 74 14	je short due-cm2,00401139	AL이 1이라면 분기
00401125	, 83FE 0F	cmp esi, 0F	ESI와 0x0F(15)를 비교
00401128	, 73 0F	jnb short due-cm2,00401139	원쪽의 인자값이 오른쪽

[그림2-14. 3개의 분기]

Register의 값을 변경한다거나 하는 코드가 없고, 단순히 비교를 통해 분기하는 내용만 있으므로 일단 넘어갑시다. 하지만 분기를 하기 때문에 (일부러 꼬아놓은 것이 아니라면) 나중에 분명 쓸모가 있는 부분인건 확실합니다.

[그림2-13]의 0040112A부터 계속 보겠습니다. "4번째 값" 이 들어 있는 al을 [esi+40211A]와 XOR연산을 한 후 al에 연산된 값을 넣습니다. 그리고 00401130에서 [esi+402160]에 0040112A에서 연산한 값을 가지고 있는 EAX Register의 값을 복사합니다. [esi+402160]은

XOR된 al값을 따로 저장하는 버퍼인 것 같습니다. 그리고 나서 00401136에서 ESI Register를 1증가시킵니다. 그리고 00401137에서 00401116으로 분기합니다. 즉 00401116~00401137을 또 하나의 루프로 보면 되겠습니다.

[그림2-14]처럼 분기하기 위해선 al이 0x00, 0x01이던지 아니면 ESI Register의 값이 0x0F여야 한다는 걸 알 수 있습니다. 일단 "4번째 값"부터 값을 모르기 때문에 모두 정상적으로 진행되었다고 가정하고 00401139로 분기해서 진행하겠습니다.

00401139는 EBX Register를 1증가 시킵니다. 그리고 0040113A에서 ESI Register를 초기화 시킵니다. 0040113C에서 [ebx+40112A]의 값을 al에 복사합니다. 그리고 00401142~0040114D의 루프를 진행하게 됩니다. 0040113C~0040114D의 루프를 빠져 나오는 분기문은 2개 있습니다.

0040113C	> 8A83 1A214000	mov al, byte ptr ds:[ebx+40211A]	1byte 단위로 버퍼의 값을 AL로 복사
00401142	. 3C 00	cmp al, 0	AL과 0을 비교
00401144	. 74 09	je short due-cm2,0040114F	AL이 0과 같다면 분기
00401146	. 3C 01	cmp al, 1	AL과 1을 비교
00401148	. 74 F2	je short due-cm2,0040113C	AL과 1이 같다면 분기
0040114A	. 03F0	add esi, eax	ESI = ESI + EAX
0040114C	. 43	inc ebx	EBX = EBX + 1
0040114D	. EB ED	jmp short due-cm2,0040113C	40113C로 분기

[그림2-15. 또 다른 확인]

00401142~00401144와 00401146~00401148 2개의 분기문이 존재하는데, 첫 번째 것은 0040114F로 분기하고 두 번째 것은 0040113C로 분기하게 됩니다. 0040113C로 분기하는 것은 단지 버퍼의 값을 읽어오기 위한 것 이고, 0040114F로 분기하는 것은 ESI Register의 값을 마지막으로 비교해서 성공 또는 실패를 구분 짓습니다.

00401144	. 74 09	je short due-cm2,0040114F	AL이 0과 같다면 분기
00401146	. 3C 01	cmp al, 1	AL과 1을 비교
00401148	. 74 F2	je short due-cm2,0040113C	AL과 1이 같다면 분기
0040114A	. 03F0	add esi, eax	ESI = ESI + EAX
0040114C	. 43	inc ebx	EBX = EBX + 1
0040114D	. EB ED	jmp short due-cm2,0040113C	40113C로 분기
0040114F	> 81FE B2010000	cmp esi, 1B2	ESI와 0x1B2(434)를 비교
00401155	. 75 A0	jnz short due-cm2,004010F7	ESI가 0x1B2(434)가 아니면 분기

[그림2-16. 마지막 검증]

[그림2-16]에 0040114F~00401155를 만족하면 코드를 해결하게 되는 것입니다. 성공의 조건은 ESI Register의 값이 0x1B2(434)가 되어야 하는 것인데, 가장 가까운 곳에서 ESI Register의 값을 변경시키는 코드는 0040114A인걸 알 수 있습니다. 즉 ESI Register에 EAX Register의 값을 더해서 0x1B2(434)가 만들어져야 합니다.

요약해보면 0040113C에서 버퍼의 값을 읽어 al에 복사하고 0040114A에서 ESI Register의 값이 0x1D5(434)가 된 후에 0040114C에서 EBX Register를 1증가 시키고, 0040114D에서 0040113C로 분기한 후에 0040113C에서 al에 복사하는 버퍼의 값이 0x00이어야 합니다. 그 래야만 0040113C~0040114D 루프를 빠져 나와 0040114F로 분기하고 계속 진행하여 성공

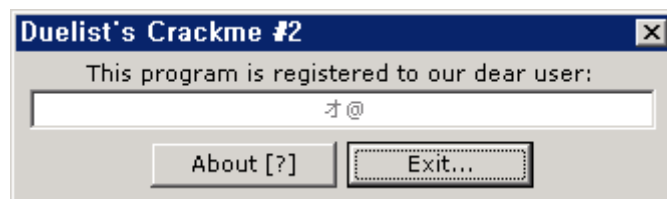
하게 됩니다. 즉 0040114F로 분기하기 위해선 버퍼의 마지막 값은 0x00이 되어야 한다는 것을 알 수 있습니다.

이 모든 것을 종합해서 Keyfile의 조건을 정리해보면 Keyfile의 크기는 최소한 18byte 이상이어야 하고, Keyfile의 값 중에는 0x01과 0x00이 필요한데 0x01은 2개 이상이어야 하고 0x00은 0x01을 2개 이상 사용하기 전에는 사용해서는 안됩니다. 그리고 0x00은 마지막에 들어가야 합니다. Keyfile의 값 중 첫 번째 0x01이 읽히기 전까지 값들의 합이 0x1D5(469)가 되어야 하고, 첫 번째 0x01 이후의 값과 두 번째 0x01 이전의 값은 XOR연산하여 그 결과가 다른 버퍼에 저장되게 됩니다. 마지막으로 Keyfile의 값 중 두 번째 0x01값 이후의 값들 중 0x00이전까지의 값들의 합이 0x1B2(434)가 되어야 합니다.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00000000h:	EA	EB	01	41	41	41	41	41	41	41	41	41	41	41	01	D9 ; 00.AAAAAAAAAAAAAA.?
00000010h:	D9	00														; ?

[그림2-17. Keyfile]

첫 번째 0x01이 나오기 전의 값들의 합을 0x1D5(469)로 맞추고 두 번째 0x01이 나오기 전까지 값은 0x00과 0x01을 제외한 나머지 값으로 채웁니다. 그리고 두 번째 0x01 이후와 0x00이 나오기 전까지의 값들의 합이 0x1B2(434)가 되면 Keyfile은 정상 작동합니다. [그림2-17]의 값을 가진 Keyfile을 만들어 CrackMe를 실행해보겠습니다.



[그림2-18. 정상작동 된 화면]

정상적으로 작동하는걸 확인하였는데 등록된 User이름이 나와야 할 곳에 깨진 문자가 나오는걸 봐선 뭔가 문제가 있는 것 같습니다. 풀이한 코드중에 Keyfile의 값을 가지고 하는 연산은 ADD와 XOR가 있었는데 ADD는 Keyfile의 처음과 끝에서 사용해서 특정 값을 만들고 있고, XOR연산은 XOR후에 새로운 버퍼에 저장시키는걸 알고 있으므로 그 부분을 확인해보면 될 것 같습니다. 새로운 버퍼의 주소는 [그림2-13]에 00301130에서 확인할 수 있습니다. 바로 00402160입니다.

Address	Hex dump	ASCII
00402160	AB AA 40 00 00 00 00 00 00 00 00 00 00 00 00 00	才@.....
00402170	00 00 00 12 00 00 00 00 00 40 00 00 00 00 00 00	...I.....

[그림2-19. 새로운 버퍼 00402160]

예상했던 대로 텍스트박스의 내용이 새로운 버퍼에 저장되어 있는걸 확인할 수 있었습니다. 따라서 저 값들이 어떻게 생성되는지 알면 우리의 닉네임을 혹은 원하는 문자를넣을 수

있을 것 같습니다. XOR연산을 하는 코드로 가보겠습니다.

0040112A	,	3286	1A214001	xor al, byte ptr ds:[esi+40211A]	AL = AL ^ 버퍼
00401130	,	8986	60214001	mov dword ptr ds:[esi+402160], eax	402160에 XOR된 값을 복사

[그림2-20. XOR연산 후 새로운 버퍼에 저장하는 코드]

[그림2-17]의 Keyfile을 이용해 어떻게 값이 변화되고 진행되는지 확인해보면 되겠습니다. Keyfile의 첫 번째 값과 두 번째 값의 합이 0x1D5(469)가 되어서 004010F5에서 분기하여 00401114로 오게 되었습니다. [그림2-21]은 그때의 Registers화면입니다. EBX Register와 ESI Register를 유심히 볼 필요가 있습니다.

```
Registers (FPU)
EAX 00000001
ECX 7C801898 kernel32,7C801898
EDX 7C93EB94 ntdll,KiFastSystemCallRet
EBX 00000002
ESP 0012FFC4
EBP 0012FFF0
ESI 000001D5
EDI 7C940738 ntdll,7C940738
EIP 00401114 due-cm2,00401114
```

[그림2-21. 00401114 코드의 Registers]

00401114	>	33F6	xor esi, esi	ESI를 초기화	
00401116	>	43	inc ebx	EBX = EBX + 1	
00401117	,	8A83	1A214001	mov al, byte ptr ds:[ebx+40211A]	1byte 단위로 버퍼의 값을 AL로 복사

```
ds:[0040211D]=41 ('A')
al=01
```

```
Registers (FPU)
EAX 00000001
ECX 7C801898 kernel32,7C801898
EDX 7C93EB94 ntdll,KiFastSystemCallRet
EBX 00000003
ESP 0012FFC4
EBP 0012FFF0
ESI 00000000
EDI 7C940738 ntdll,7C940738
EIP 00401117 due-cm2,00401117
```

[그림2-22. 코드와 코드내용 그리고 Registers화면]

[그림2-22]의 00401117 코드를 보면 al에 [ebx+40211A]값을 복사하는걸 볼 수 있습니다. 이 때 EBX Register의 값은 3입니다. Keyfile의 내용을 가지고 있는 버퍼의 주소에 3을 더해 주면 40211D가 됩니다. 40211D으로 가서 그 값을 확인해 보겠습니다. 0xEA, 0xEB, 0x01 다음의 0x41인 것을 확인할 수 있습니다. 그 뒤에 이어지는 [그림2-14]의 조건분기는 만족하지 않으므로 무시되고 진행되어 0040112A에서 al과 [esi=40211A]의 값이 XOR연산 후 al에 저장되어 00401130에서 [esi+402160]에 복사됩니다. 이 반복은 [그림14]의 조건분기중 하나를 만족시켜야 빠져나가게 됩니다. 즉 두 번째 0x01이 나오기 전까지 반복되게 됩니다. 두 번

째 0x01이 나오기 전까지 진행시켜 새로운 버퍼([esi+402160])의 값을 확인해 보겠습니다. 그런데 같은 값을 XOR시키면 0이 되기 때문에 알아보기가 귀찮으니 0xEA, 0xEB, 0x01, 0x41, 0x42, ... 0x01, 0xD9, 0xD9, 0x00으로 새로운 Keyfile을 만들어 확인해 보겠습니다.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
00000000h:	EA	EB	01	41	42	43	44	45	46	47	48	49	50	51	01	D9	; .ABCDEFGHIQ.?
00000010h:	D9	00															; ?

[그림2-23. 새로운 Keyfile]

XOR연산은 al에 일단 [ebx+40211A]의 값을 넣고, al을 [esi+40211A]와 XOR시킨 후 그 값을 [esi+402160]에 넣습니다. EBX Register는 3 ESI Register는 0에서부터 증가한다고 가정하고 연산해보겠습니다.

[00402160] = 0x41 ^ 0xEA

[00402161] = 0x42 ^ 0xEB

[00402162] = 0x43 ^ 0x01

[00402163] = 0x44 ^ 0x41

...

...

[00402168] = 0x49 ^ 0x46

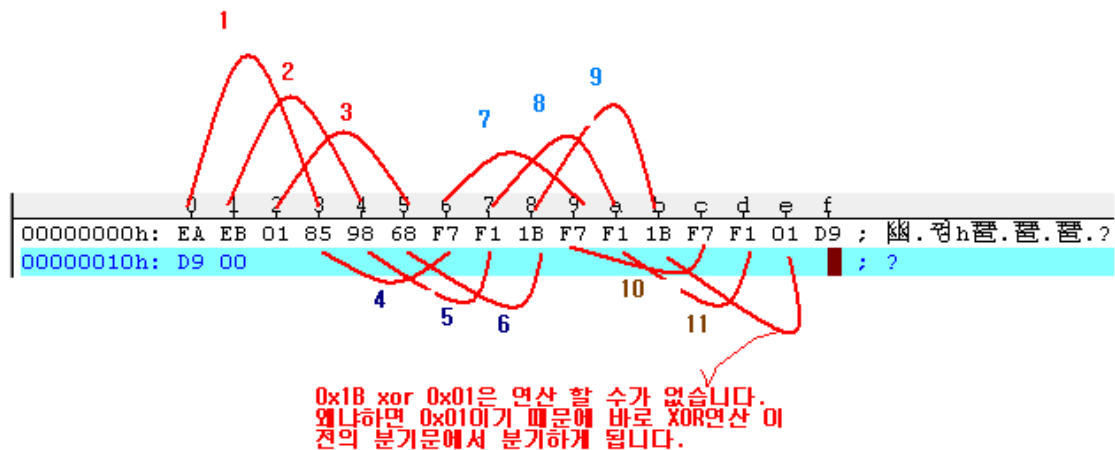
[00402169] = 0x50 ^ 0x47

연산된 결과를 [그림2-24]에서 보시는 것처럼 00402160에서 확인할 수 있었습니다.

Address	Hex dump	ASCII
00402160	AB A9 42 05 07 05 03 00 0F 17 19 00 00 00 00 80	⌘B • L,4t,...I

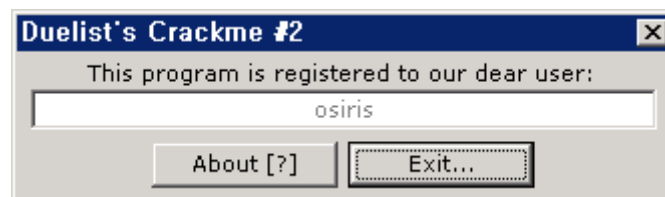
[그림2-24. 새로운 Keyfile로 생성된 새로운 버퍼에 저장된 값들]

0x0003 결론



[그림2-25. osiris 닉네임 만들기]

첫 번째 0x01이 나오기 전의 값들의 합이 0x1D5(469)가 되어야 합니다. 두 번째 0x01이 나온 후의 값과 마지막 0x00이 나오기 이전 값들의 합이 0x1B2(434)가 되어야 합니다. 첫 번째 0x01과 두 번째 0x01 사이의 값은 XOR연산에 이용되며 이를 이용해서 Username을 텍스트박스에 표시할 수 있습니다. Username을 만드는 것을 [그림2-25]를 가지고 설명하자면 두 번째 0x01이 나오기 전까지의 값들을 이용해 XOR연산을 시키는데 0xEA를 0번 값이라고 한다면 0번 값과 3번 값을 XOR시키고, 1번 값과 4번 값을 XOR시키고 ... 이런 식으로 진행합니다. 그렇게 11번째 XOR연산을 한 후에 12번째 XOR연산을 하려고 하지만 두 번째 0x01이 나왔기 때문에 XOR연산을 중단하고 분기하게 됩니다. 분기한 곳에서 두 번째 0x01 이후의 값과 마지막 0x00 이전의 값을 더해서 0x1B2(434)인지 확인 후 에러메시지를 보게 되거나 User등록을 할 수 있게 됩니다.



[그림2-26. osiris user등록 완료]

0x03 Duelist #3

Crackme: <http://beist.org/research/public/crackme10/duel-cm3.zip>

Keygen: <http://beist.org/research/public/crackme10/duel-cm3-keygen.zip>

Author: Duelist

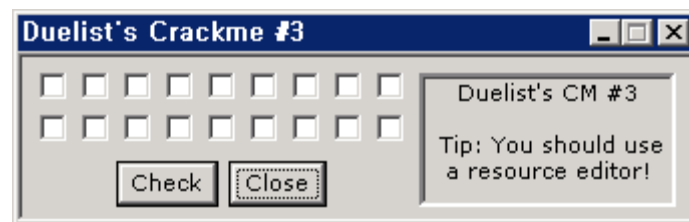
Level: ★

Protection: Matrix

0x0001 목표

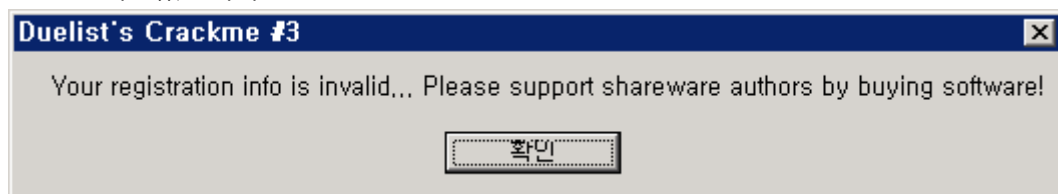
퍼즐 같은 CrackMe입니다. 총 18개의 체크박스 중 정해진 몇 개만 선택해야 성공메시지를 볼 수 있습니다. 분석을 통해서 어떻게 체크박스를 이용해서 CrackMe를 만들었고, 어떻게 하면 262144개의 경우를 테스트 해 볼 것인지 공부합니다.

0x0002 분석 및 풀이



[그림3-1. Duelist's Crackme #3 실행화면]

총 18개의 체크박스와 Resource Editor를 사용하라는 Tip이 나와있습니다. 일단 체크버튼을 한번 눌러보겠습니다.



[그림3-2. 에러 메시지]

당연히 [그림3-2]와 같은 메시지가 뜰 줄 알았습니다. 체크박스는 총18개 입니다. 그리고 체크박스는 Check를 하거나 Check하지 않는 2가지의 모양새를 가질 수 있습니다.

즉, 2^{18} (262144)개를 표현 할 수 있습니다.

그럼 이제 OllyDbg를 이용해서 Duelist's Crackme #3를 열어보도록 하겠습니다.

00401101	> 68 C3204000	push due-cm3,004020C3	Text = "Duelist's CM #3,NO Tip: You sho
00401106	, 6A 03	push 3	ControlID = 3
00401108	, FF75 08	push dword ptr ss:[ebp+8]	hWnd
0040110B	, E8 3E020000	call <jmp,&USER32.SetDlgItemTextA>	SetDlgItemTextA
00401110	, B8 01000000	mov eax, 1	
00401115	, EB E3	jmp short due-cm3,004010FA	
00401117	> 33F6	xor esi, esi	ESI 초기화
00401119	, 33D2	xor edx, edx	EDX 초기화
0040111B	, 8935 5E214000	mov dword ptr ds:[40215E], esi	
00401121	, 8935 62214000	mov dword ptr ds:[402162], esi	
00401127	> 0FB8E8 FE2040	movsx ecx, byte ptr ds:[esi+4020FE]	
0040112E	, 83F9 4D	cmp ecx, 4D	ECX와 0x4D(77)을 비교
00401131	, 74 2F	je short due-cm3,00401162	같다면 401162로 분기
00401133	, 890D 5E214000	mov dword ptr ds:[40215E], ecx	
00401139	, 51	push ecx	ButtonID
0040113A	, FF75 08	push dword ptr ss:[ebp+8]	hWnd
0040113D	, E8 00010000	call <jmp,&USER32.IsDlgButtonChecked>	IsDlgButtonChecked
00401142	, 46	inc esi	ESI = ESI + 1
00401143	, 83F8 00	cmp eax, 0	EAX를 0과 비교
00401145	, 74 DF	je short due-cm3,00401127	Checkbox가 Check되지 않았다면 분기
00401148	, A1 5E214000	mov eax, dword ptr ds:[40215E]	40215E가 가리키는곳의 값을 EAX에 복사
0040114D	, 0FB8E8 FE2040	movsx ecx, byte ptr ds:[esi+4020FE]	61 다음 값인 52를 받아서 ecx에 저장
00401154	, 0FAFC1	imul eax, ecx	eax와 ecx를 곱해서 eax에 넣는다
00401157	, 0FAFC6	imul eax, esi	eax와 esi를 곱해서 eax에 넣는다
0040115A	, 0105 62214000	add dword ptr ds:[402162], eax	곱셈되어진 최종 eax의 값을 402162에 저장
00401160	, EB C5	jmp short due-cm3,00401127	401127로 분기
00401162	> A1 62214000	mov eax, dword ptr ds:[402162]	402162의 값을 EAX에 복사
00401167	, 68C0 4D	imul eax, eax, 4D	EAX = EAX * 0x4D(77)
0040116A	, 3D 6654F300	cmp eax, 0F35466	EAX와 0x0F35466(15946854)을 비교
0040116F	, 75 20	jnz short due-cm3,00401191	다르면 분기 같으면 밑으로
00401171	, 68 00200000	push 2000	Style = MB_OK MB_TASKMODAL
00401176	, 68 01204000	push due-cm3,00402001	Title = "Duelist's Crackme #3"
0040117B	, 68 17204000	push due-cm3,00402017	Text = "Congratulations! Please send a s
00401180	, 6A 00	push 0	hOwner = NULL
00401182	, E8 55010000	call <jmp,&USER32.MessageBoxA>	MessageBoxA
00401187	, B8 01000000	mov eax, 1	
0040118C	, E9 69FFFFFF	jmp due-cm3,004010FA	
00401191	> 68 00200000	push 2000	Style = MB_OK MB_TASKMODAL
00401196	, 68 01204000	push due-cm3,00402001	Title = "Duelist's Crackme #3"
0040119B	, 68 68204000	push due-cm3,00402068	Text = "Your registration info is invali
004011A0	, 6A 00	push 0	hOwner = NULL
004011A2	, E8 35010000	call <jmp,&USER32.MessageBoxA>	MessageBoxA

[그림3-3. Duelist's Crackme #3의 코드들]

이전 CrackMe와 달리 코드가 상당히 짧아 보입니다. 0040116A~0040116F에서 조건분기를 통해서 성공 메시지와 실패 메시지로의 분기를 하게 됩니다. 조건은 EAX Register의 값이 0x0F35466(15946854)일 때 입니다. 0040116A 바로 위의 코드를 보면 EAX Register의 값이 어떻게 만들어지는지 알 수 있습니다. 즉 00401167에서 EAX Register가 가져야 되는 값은 $0xF35466 / 0x4D = 0x328FE$ 입니다. 우리는 체크박스를 이용해 EAX Register의 값을 0x328FE로 만들어야 하고 그 값이 어떻게 만들어지는지 알아야 합니다.

00401117 xor esi, esi

//ESI Register를 초기화 시킵니다.

00401119 xor edx, edx

//EDX Register를 초기화 시킵니다.

00401127 movsx ecx, byte ptr ds:[esi+4020FE]

//ECX Register에 [esi+4020FE]가 가지고 있는 값을 읽어옵니다.

Address	Hex dump	ASCII
004020FE	16 49 5E 15 27 26 21 25 1D 59 53 37 31 48 5D 00	^I^L'&!%YS71H).
0040210E	61 52 4D 00 00 00 00 00 00 00 00 00 00 00 00 00	aRM,.....

[그림3-4. ECX Register에 들어갈 값들]

[그림3-4]를 보면 0x16부터 0x4D까지의 값을 확인할 수 있습니다. 00401127에서 ECX Register로 읽혀질 값들입니다.

00401117	> 33F6	xor esi, esi	ESI 초기화
00401119	. 33D2	xor edx, edx	EDX 초기화
0040111B	. 8935 5E214000	mov dword ptr ds:[40215E], esi	
00401121	. 8935 62214000	mov dword ptr ds:[402162], esi	
00401127	> 0FBF8E FE2040	movsx ecx, byte ptr ds:[esi+4020FE]	
0040112E	. 83F9 4D	cmp ecx, 4D	ECX와 0x4D(77)을 비교
00401131	. 74 2F	je short due-cm3,00401162	같다면 401162로 분기
00401133	. 890D 5E214000	mov dword ptr ds:[40215E], ecx	
00401139	. 51	push ecx	
0040113A	. FF75 08	push dword ptr ss:[ebp+8]	
0040113D	. E8 D0010000	call <jmp.&USER32.IsDlgButtonChecked>	IsDlgButtonChecked
00401142	. 46	inc esi	ESI = ESI + 1
00401143	. 83F8 00	cmp eax, 0	EAX를 0과 비교
00401146	. 74 DF	je short due-cm3,00401127	Checkbox가 Check되지 않았다면 분기

[그림3-5. 00401117~00401146]

ECX Register에 읽은 값을 0040112E에서 0x4D(77)과 비교합니다. 그 결과가 참이라면 00401162로 분기하게 됩니다.

00401162	> A1 62214000	mov eax, dword ptr ds:[402162]	402162의 값을 EAX에 복사
00401167	. 6BC0 4D	imul eax, eax, 4D	EAX = EAX * 0x4D(77)
0040116A	. 3D 6654F300	cmp eax, 0F35466	EAX와 0x0F35466(15946854)을 비교
0040116F	. 75 20	jnz short due-cm3,00401191	다르면 분기 같으면 밑으로

[그림3-6. 00401162~0040116F]

하지만 그 결과가 참이 아니라면 ECX Register의 값을 0x40215E에 넣습니다. 그리고 ESI Register를 1증가시키고 나서 체크박스가 선택되어 있는지 안되어 있는지 확인을 하게 되고, 선택되어 있다면 EAX Register를 1로 만들어 [그림3-5]의 00401143~00401146에서 00401127로 분기하지 않게 합니다.

00401148	. A1 5E214000	mov eax, dword ptr ds:[40215E]	40215E가 가리키는곳의 값을 EAX에 복사
0040114D	. 0FBF8E FE2040	movsx ecx, byte ptr ds:[esi+4020FE]	61 다음 값인 52를 받아서 ecx에 저장
00401154	. 0FAFC1	imul eax, ecx	eax와 ecx를 곱해서 eax에 넣는다
00401157	. 0FAFC6	imul eax, esi	eax와 esi를 곱해서 eax에 넣는다
0040115A	. 0105 62214000	add dword ptr ds:[402162], eax	곱셈되어진 최종 eax의 값을 402162에 저장
00401160	. EB C5	jmp short due-cm3,00401127	401127로 분기

[그림3-7. 00401148~00401160]

위의 내용들을 종합해서 생각하면 체크박스가 하나라도 Check되어 있다면 00401146에서 00401127로 분기하지 않고 00401148로 진행하여 [그림3-7]과 같은 연산을 하게 됩니다.

```

00401148 mov eax, dword ptr ds:[40215E]
//EAX Register에 40215E의 값을 복사합니다.
0040114D movsx ecx, byte ptr ds:[esi+4020FE]
//ECX Register에 [esi+4020FE]가 가지고 있는 값을 읽어옵니다.
00401154 imul eax, ecx
00401157 imul eax, esi
//EAX = EAX * ECX * ESI
0040115A add dword ptr ds:[402162], eax
//402162에 EAX Register의 값을 더합니다.
00401160 jmp short due-cm2.00401127
//00401127로 분기합니다.

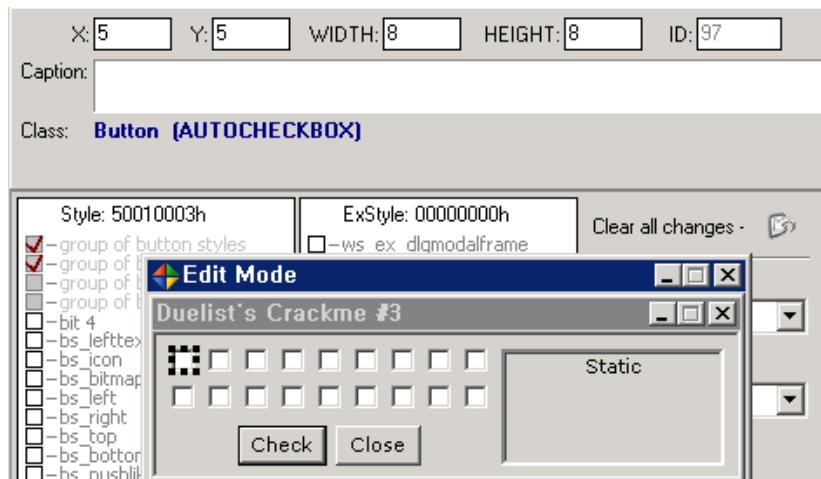
```

00401127	>	OFBE8E FE2041	movsx ecx, byte ptr ds:[esi+4020FE]	
0040112E	.	83F9 4D	cmp ecx, 4D	ECX와 0x4D(77)을 비교
00401131	.	74 2F	je short due-cm3.00401162	같다면 401162로 분기
00401133	.	890D 5E214000	mov dword ptr ds:[40215E], ecx	
00401139	.	51	push ecx	
0040113A	.	FF75 08	push dword ptr ss:[ebp+8]	
0040113D	.	E8 00010000	call <jmp.&USER32.IsDlgButtonChecked>	IsDlgButtonChecked
00401142	.	46	inc esi	ESI = ESI + 1
00401143	.	83F8 00	cmp eax, 0	EAX을 0과 비교
00401145	^	74 DF	je short due-cm3.00401127	Checkbox가 Check되지 않았다면 분기
00401148	.	A1 5E214000	mov eax, dword ptr ds:[40215E]	40215E가 가리키는곳의 값을 EAX에 복사
0040114D	.	OFBE8E FE2041	movsx ecx, byte ptr ds:[esi+4020FE]	61 다음 값인 52를 받아서 ecx에 저장
00401154	.	0FAFC1	imul eax, ecx	eax와 ecx을 곱해서 eax에 넣는다
00401157	.	0FAFC6	imul eax, esi	eax와 esi을 곱해서 eax에 넣는다
0040115A	.	0105 62214000	add dword ptr ds:[402162], eax	곱셈되어진 최종 eax의 값을 402162에 저장
00401160	.	EB C5	jmp short due-cm3.00401127	401127로 분기

[그림3-8. 00401127~00401160]

00401127~00401160을 Loop로 볼 수 있습니다. 중간에 2개의 조건 분기문이 있는데 이것은 Loop를 빠져나가기 위한 코드로 보면 될 것입니다.

Resource Editor툴을 이용해서 Duelist's Crackme #3를 열어보겠습니다. 그리고 체크박스가 가지고 있는 각각의 ID를 알아보겠습니다.



[그림3-9. Resource Editor를 이용한 ID확인]

[그림3-9]처럼 해서 각각의 체크박스가 가지는 ID값을 알아내었습니다.



[그림3-10. 체크박스의 ID값들]

[그림3-10]의 체크박스가 가지는 ID값들과 똑 같은 값들을 [그림3-4]에서 찾을 수 있었습니다. [그림3-4]의 값들은 총 19개인데 비해 체크박스의 개수는 총 18개입니다. [그림3-4]의 마지막 값이 19번째 0x4D는 004011E~00401131에서 Loop를 탈출 하기 위한 0040112E의 조건 분기문에 쓰이는 임의의 값이라고 짐작할 수 있습니다.

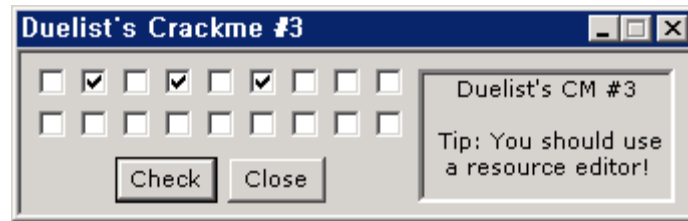
4020FE에 저장된 값 (0x4D제외)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
16	49	5E	15	27	26	21	25	1D	59	53	37	31	48	5D	0C	61	52

체크박스의 ID값

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
61	49	5E	16	25	26	21	59	53	15	37	31	48	50	0C	52	27	10

이제 실행해서 임의로 몇 개의 체크박스를 Check한 후에 진행해 보겠습니다.



[그림3-11. Check된 체크박스]

[그림3-11]처럼 2, 4, 8번째 체크박스를 선택하고 Check 버튼을 눌렀습니다. F8키를 이용해서 Step-over로 한 라인씩 진행하겠습니다. 00401117~00401119에서 ESI Register와 EDX Register를 초기화 시킵니다. 그리고 40215E에 ESI Register값을 넣습니다. 초기화된 값이므로 0이 들어갈 것입니다. 그리고 00401121에서 402162에도 ESI Register값을 넣습니다.

이것 또한 초기화된 값이므로 0이 들어갈 것입니다. 00401127에서 ECX Register에 [esi+4020FE]의 값을 복사합니다. [0+4020FE]의 값은 [그림11]위의 표에서 확인할 수 있습니다. 0x16이 ECX Register로 복사 될 것입니다. Registers (FPU)에서 확인해보겠습니다.

```
Registers (FPU)
EAX 7FFDE000
ECX 00000016
EDX 00000000
EBX 00000000
ESP 0012FBBC
EBP 0012FBC8
ESI 00000000
EDI 0012FC30
EIP 0040112E due-cm3,0040112E
```

[그림3-12. Registers (FPU) ECX의 변화]

[그림3-12]에서 보시는 것과 같이 ECX Register의 값이 0x16으로 변경되었습니다. 다음 코드인 0040112E~00401131에선 조건분기를 하게 되는데 ECX Register의 값이 0x4D 일 때만 분기하게 됩니다. 지금 ECX Register의 값은 [그림3-12]에서 보이는 것처럼 0x16입니다. 그러므로 절대 분기하고 다음 코드를 진행하게 됩니다. 다음 코드인 00401133에서 [40215E]에 ECX Register값을 복사합니다. 그러면 [40215E]에는 0x16이 들어갈 것입니다. 확인해보겠습니다.

Address	Hex dump	ASCII
0040215E	16 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

[그림3-13. 0040215E에 처음 입력된 값]

[40215E]에 0x16이 들어간 것을 [그림3-13]에서 확인할 수 있었습니다. 그리고 난 후 ESI Register를 1증가 시키고 나서 IsDlgButtonChecked API를 호출 하게 됩니다.

<pre>00401139 . 51 push ecx 0040113A . FF75 08 push dword ptr ss:[ebp+8] 0040113D . E8 D0010000 call <jmp,&USER32.IsDlgButtonChecked></pre>	<pre>ButtonID = 16 (22.) hWnd IsDlgButtonChecked</pre>
--	--

[그림3-14. IsDlgButtonChecked API]

IsDlgButtonChecked API는 다음과 같습니다.

```
UINT IsDlgButtonChecked(
```

```
    HWND hDlg, //대화상자의 핸들
```

```
    int nIDButton //Check상태를 조사할 버튼의 ID
```

```
);
```

설명

라디오 버튼이나 체크 버튼의 체크 상태를 조사한다. 컨트롤에게 BM_GETCHECK 메시지를 보내 체크 상태를 조사해 준다.

[그림3-14]를 보면 ButtonID에 0x16값이 들어가 있는 것을 확인할 수 있습니다. ECX Register가 가지고 있는 값을 ButtonID로 가지게 되는데 [4020FE]에 저장된 값을 차례로 읽어와 체크박스의 Check상태를 확인하게 됩니다. Check가 되어 있다면 EAX Register에 1을 반환하여 00401127로 분기하지 않게 됩니다.

우리는 2, 4, 8번째 체크박스를 Check하였으므로 0x49, 0x16, 0x26값에 대해서 IsDlgButtonChecked를 호출하면 EAX Register에 1을 반환할 것입니다. 계속 진행하면서 지켜보겠습니다. 0x16를 ID값으로 가지는 체크박스를 Check하였으므로 EAX Register에 1을 반환하고 00401127로 분기하지 않고 진행하게 됩니다. 00401148에서 EAX Register에 [40215E]의 값을 복사합니다. [그림13]에서 보듯이 [40215E]의 값은 0x16입니다. 0040114D에서 ECX Register에 [esi+4020FE]값을 넣습니다. 이때 ESI는 00401142에서 1증가하였기 때문에 1이 되어서 ECX Register에는 [1+4020FE]의 값(0x49)이 들어가게 됩니다. 그럼 EAX Register에는 0x16, ECX Register에는 0x49가 들어있게 됩니다. 그 다음 코드인 00401154~57에서 EAX Register와 ECX Register, ESI Register의 값을 곱해 EAX Register에 넣습니다. 계산을 하면 $0x16 * 0x49 * 1 = 0x646(1606)$ 이 됩니다. 그 결과인 EAX Register값을 0040115A에서 [402162]에 더해서 넣습니다. [402162]의 초기값은 0이므로 현재 [402162]의 값은 0x646(1606)이 됩니다. 그리고 나서 00401160에서 00401127로 분기하게 됩니다.

00401127에서 ECX Register에 [esi+4020FE]의 값을 넣는데 ESI는 이전에 1로 증가하였으므로 ECX Register에 들어가는 값은 [1+4020FE]의 값(0x49)이 됩니다. 00401142에서 ESI Register를 1증가 시키고 그 다음 조건 분기문을 만족시키지 못하므로 또 아래로 진행하게 됩니다. 00401148에서 EAX Register에 [40215E]의 값(0x49)을 넣습니다. 그리고 0040114D에서 ECX Register에 [esi+4020FE]의 값(0x5E)을 넣습니다. 00401154~00401157에서 EAX Register에 $EAX * ECX * ESI$ 값을 넣게 됩니다. 계산을 하면 $0x49 * 0x5E * 2 = 0x1ACE(6862)$ 가 됩니다. 0x1ACE(6862)값을 [402162]에 더합니다. 그럼 [402162]의 값은 0x3BE2(15330)가 됩니다. 그리고 나서 또 00401127로 분기합니다. 그렇게 계속 진행하여 선택되지 않은 체크박스의 ID값은 무시하고 선택된 체크박스의 ID값을 이용해 다음과 같은 계산을 하게 됩니다.

$[402162] = ([\text{선택된 체크박스의 ID값}] * [\text{esi}+4020\text{FE}] * [\text{esi}]) + (...) + ... + (...)$

$[402162] = (0x16 * 0x49 * 1) + (0x49 * 0x5E * 2) + (0x26 * 0x21 * 6)$

여기서 esi값은 다음 표를 참고하시면 됩니다.

4020FE에 저장된 값 (0x4D제외)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
16	49	5E	15	27	26	21	25	1D	59	53	37	31	48	5D	0C	61	52

$[402162] = (0x16 * 0x49 * 1) + (0x49 * 0x5E * 2) + (0x26 * 0x21 * 6)$

즉, 선택된 체크박스의 ID값과 [esi+4020FE]값 그리고 esi값을 곱한 값들의 합이 0x328FE(207102)인 조건을 찾는 것이며, 우리는 간단한 프로그래밍을 통해서 값을 구할 수 있습니다.

0x0003 결론

체크박스는 선택과 선택하지 않음의 2가지 조건만을 가질 수 있습니다. 총 18개의 체크박스의 선택과 선택하지 않음의 2가지 조건을 경우의 수로 따지면 2의 18승이 됩니다. 262144라는 엄청난 숫자가 나오는데 이걸 수작업으로 하자면 시간이 엄청 오래 걸릴 것입니다. 그래서 프로그래밍의 도움을 받아야 합니다.

```
// duelist3_keygen.cpp : 콘솔응용프로그램에대한진입점을정의합니다.
// 코드 By Mins4416 @ naver . com

#include "stdafx.h"
char checked[18];
void flag_checked(int i) {
    for(int j=0; j < 18; j++) {
        if (i & (1<<j))
            checked[j] = 0x31;
        else
            checked[j] = 0x30;
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    // 미리연산된값
    int keyvalue[]={0x21032, 0x359C, 0x1722, 0x646, 0x2188, 0x1D64, 0x2163, 0x1208E,
        0xC427, 0xCCC, 0x7E54, 0xB328, 0x16E30, 0x4164, 0x48C0, 0x1BBF4, 0x1CF2, 0x5ABD};

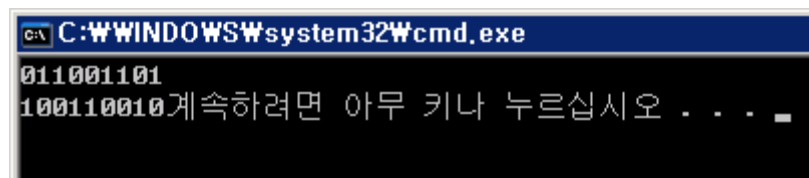
    /* int keyvalue[]={0x646, 0x359C, 0x1722, 0xCCC, 0x1CF2, 0x1D64, 0x2163, 0x2188, 0x5ABD,
        0x1208E, 0xC427, 0x7E54, 0xB328, 0x16E30, 0x4164, 0x48C0, 0x21032, 0x1BBF4}; */

    //각각의경우의수의연산값이저장되는곳
    int sum = 0;
    //정답
    int answer = 0x328FE;
}
```

```

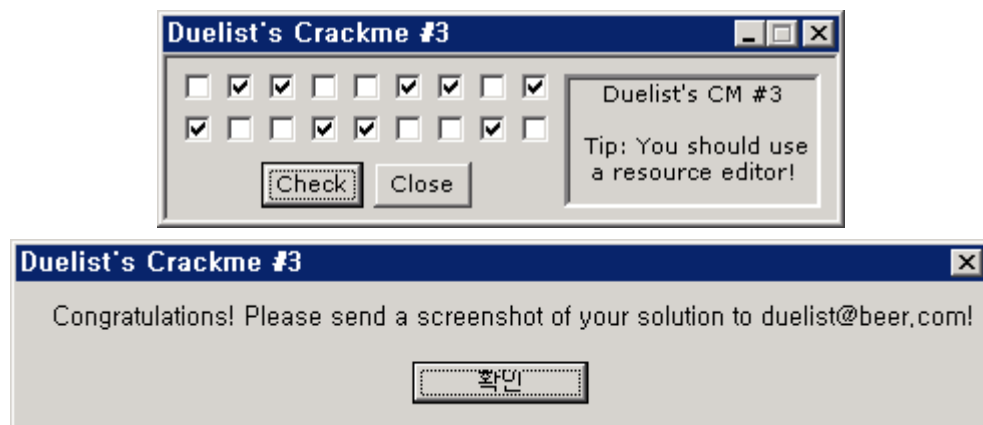
//모든경우의수를계산한다.
for(int i = 0; i < 262143; i++)
{
    sum=0;
    flag_checked(i);
    for(int j=0;j<18; j++) {
        if(checked[j]=='1') {
            sum += keyvalue[j];
        }
    }
    if (sum == answer)
        break;
}
//정답출력
for(int i=0; i < 18; i++)
{
    printf("%c", checked[i]);
    if(i == 8)
        printf("\n");
}
system("pause");
return 0;
}

```



[그림3-15. 정답]

[그림3-15]에 보이는 것처럼 체크박스를 선택한 후 체크버튼을 눌러보면 ...



[그림3-16. 성공 메시지]

성공하였습니다!!

0x04 Duelist #4

Crackme: <http://beist.org/research/public/crackme10/duel-cm4.zip>

Keygen: <http://beist.org/research/public/crackme10/duel-cm4-keygen.zip>

Author: Duelist

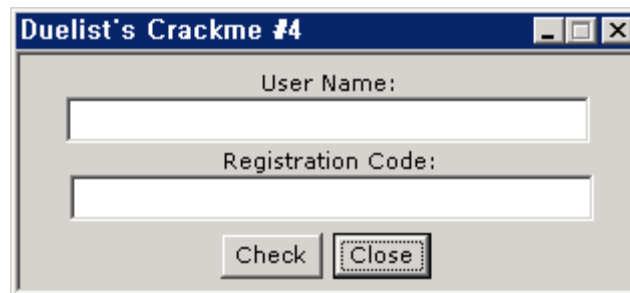
Level: ★

Protection: Name / Serial

0x0001 목표

User Name과 그에 맞는 Registration 코드를 찾아야 하며, 각각의 User Name에 맞는 Registration 코드를 생성하는 Keygen을 만드는 것이 이번의 목표입니다.

0x0002 분석 및 풀이



[그림4-1. Duelist's Crackme #4 실행화면]

[그림4-1]에서 보는 바와 같이 이번 CrackMe에선 User Name과 Registration 코드를 요구하고 있습니다. 제 닉네임 Osiris를 이용해서 진행하겠습니다. 일단 OllyDbg를 이용해서 CrackMe를 열어보겠습니다.

00401127	> 6A 00	push 0	IPParam = 0 wParam = 0 Message = WM_GETTEXTLENGTH ControlID = 3 hWnd
00401129	. 6A 00	push 0	
0040112B	. 6A 0E	push 0E	SendDlgItemMessageA UserName의 길이인 EAX값을 [4021AF]에 복사 EAX를 0과 비교 (입력 여부 확인) EAX와 0과 같다면 40121A로 분기 (입력값이 없다면 분기) EAX와 8을 비교 (최대 길이 확인) EAX가 8보다 크다면 40121A(실패)로 분기 ESI = EAX;
0040112D	. 6A 03	push 3	
0040112F	. FF75 08	push dword ptr ss:[ebp+8]	IPParam = 0 wParam = 0 Message = WM_GETTEXTLENGTH ControlID = 4 hWnd
00401132	. E8 41020000	call <jmp.&USER32.SendDlgItemMessageA>	
00401137	. A3 AF214000	mov dword ptr ds:[4021AF], eax	SendDlgItemMessageA Registration Code 입력 안했으면 :p UserName 길이와 Reg Code 길이를 비교 같지 않다면 40121A(실패)로 분기
0040113C	. 83F8 00	cmp eax, 0	
0040113F	. 0F84 D5000000	je due-cm4,0040121A	
00401145	. 83F8 08	cmp eax, 8	
00401148	. 0F8F CC000000	jg due-cm4,0040121A	
0040114E	. 8BF0	mov esi, eax	
00401150	. 6A 00	push 0	
00401152	. 6A 00	push 0	
00401154	. 6A 0E	push 0E	
00401156	. 6A 04	push 4	
00401158	. FF75 08	push dword ptr ss:[ebp+8]	
0040115B	. E8 18020000	call <jmp.&USER32.SendDlgItemMessageA>	
00401160	. 83F8 00	cmp eax, 0	
00401163	. 0F84 B1000000	je due-cm4,0040121A	
00401169	. 3BF0	cmp esi, eax	
0040116B	. 0F85 A9000000	jnz due-cm4,0040121A	

[그림4-2. 입력 받는 조건]

[그림4-2]에서 보면 Username과 Registration 코드의 길이의 조건이 나타나있습니다. Username은 1~8개의 길이를 가져야 하고, Registration 코드는 Username과 같은 길이를 가져야 합니다. 그리고 Username의 길이는 EAX Register에 저장되며 만약 조건이 성립하지 않는다면 0040121A로 분기하여 실패 메시지를 보여주게 됩니다.

00401215	. E9 FFEFFFFF	jmp due-cm4,00401119	Style = MB_OK MB_TASKMODAL; Default case Title = "Duelist's Crackme #4" Text = "Your registration info is invalid" hOwner = NULL MessageBoxA
0040121A	> 68 00200000	push 2000	
0040121F	. 68 01204000	push due-cm4,00402001	
00401224	. 68 AE204000	push due-cm4,004020AE	
00401229	. 6A 00	push 0	
0040122B	. E8 36010000	call <jmp.&USER32.MessageBoxA>	

[그림4-3. 실패 메시지 0040121A]

그럼 OllyDbg에서 CrackMe를 Open한 상태에서 실행시킨 후 Username에는 앞서도 예고했듯이 Osiris를 사용하고, Registration 코드는 Osiris와 자릿수를 맞춰서 aaaaaa를 넣은 후 Check버튼을 눌러보겠습니다. [그림4-2]에서 보면 브레이크포인트가 여러 군데 잡혀 있지만 입력부분부터 진행 할 것이기 때문에 [그림4-2]에 나온 것처럼 모두 Breakpoint를 지정할 필요는 없습니다.

00401127에 브레이크포인트를 하나 걸어주고 한 F7(Step-in), F8키(Step-over)를 적절히 이용해 한 라인씩 진행하면 되니까요. 아무튼 그렇게 해서 00401127에서 프로그램 진행이 멈추었기 때문에 천천히 F8키를 눌러서 내려옵니다. 그럼 Username이 가져야 되는 길이의 조건을 무사히 통과하는걸 확인할 수 있습니다. 그리고 나서 0040114E에서 EAX Register의 값을 ESI Register에 복사합니다. 또 천천히 F8키(Step-over)를 누르면서 내려오다 보면 Registration 코드의 길이를 EAX Register에 넣는걸 확인할 수 있습니다. 그렇다면 0040114E에서 EAX Register의 값을 ESI Register에 왜 복사하는지 이유를 알 것 같습니다.

00401169에서 ESI Register의 값과 EAX Register의 값을 비교해서 0040121A로 분기여부를 결정하게 됩니다. 하지만 우리가 입력한 값은 분기조건과는 맞지 않기 때문에 분기하지 않게 됩니다.

```

00401171 . 68 60214000 push due-cm4,00402160
00401176 . 6A 08 push 8
00401178 . 6A 00 push 00
0040117A . 6A 03 push 3
0040117C . FF75 08 push dword ptr ss:[ebp+8]
0040117F . E8 F4010000 call <jmp.&USER32.SendDlgItemMessageA>
00401184 . 68 79214000 push due-cm4,00402179
00401189 . 6A 10 push 10
0040118B . 6A 00 push 00
0040118D . 6A 04 push 4
0040118F . FF75 08 push dword ptr ss:[ebp+8]
00401192 . E8 E1010000 call <jmp.&USER32.SendDlgItemMessageA>
00401197 . B9 FFFFFFFF mov ecx, -1
0040119C . 41 inc ecx
0040119D . 0FBE81 602140 movsx eax, byte ptr ds:[ecx+402160]

```

IParam = 402160
 wParam = 8
 Message = WM_GETTEXT
 ControlID = 3
 hWnd
 SendDlgItemMessageA
 IParam = 402179
 wParam = 10
 Message = WM_GETTEXT
 ControlID = 4
 hWnd
 SendDlgItemMessageA
 ECX를 FFFFFFFF으로 만든다
 ECX를 1증가 시킨다 (Full일 땐 0이 됨)
 EAX에 [ecx+402160]값을 복사

[그림4-4. Username과 Registration 코드 조건 확인 이후]

계속 진행을 하게 되면 00401197에서 ECX Register에 -1을 넣게 되는데 그러면 ECX Register는 FFFFFFFF이 되게 됩니다. 그리고 0040119C에서 ECX Register를 1증가 시키게 되는데 FFFFFFFF였던 ECX Register가 00000000으로 변하게 됩니다. 그리고 나서 EAX Register에 [ecx+402160]의 값을 복사합니다. 그 다음 코드들을 보도록 하겠습니다.

```

00401197 . B9 FFFFFFFF mov ecx, -1
0040119C . 41 inc ecx
0040119D . 0FBE81 602140 movsx eax, byte ptr ds:[ecx+402160]
004011A4 . 83F8 00 cmp eax, 0
004011A7 . 74 32 je short due-cm4,004011DB
004011A9 . BE FFFFFFFF mov esi, -1
004011AE . 83F8 41 cmp eax, 41
004011B1 . 7C 67 jl short due-cm4,0040121A
004011B3 . 83F8 7A cmp eax, 7A
004011B6 . 77 62 ja short due-cm4,0040121A
004011B8 . 83F8 5A cmp eax, 5A
004011BB . 7C 03 jl short due-cm4,004011C0
004011BD . 83E8 20 sub eax, 20
004011C0 . 46 inc esi
004011C1 . 0FBE96 172040 movsx edx, byte ptr ds:[esi+402017]
004011C8 . 3BC2 cmp eax, edx
004011CA . 75 F4 jnz short due-cm4,004011C0
004011CC . 0FBE86 3C2040 movsx eax, byte ptr ds:[esi+40203C]
004011D3 . 8981 94214000 mov dword ptr ds:[ecx+402194], eax
004011D9 . EB C1 jmp short due-cm4,0040119C

```

[그림4-5. Key를 만드는 코드]

0040119D에서 EAX Register에 값을 읽어 들이는 [ecx+402160]의 값이 무엇인지 우선 확인을 해야 합니다. Username값인지 Registration 코드값인지 아니면 다른 임의의 값인지 확인해야 됩니다.

Address	Hex dump	ASCII
00402160	4F 73 69 72 69 73 00 00 00 00 00 00 00 00 00 00	Osiris.....
00402170	00 00 00 00 00 00 00 00 00 61 61 61 61 61 61 00aaaaaa

[그림4-6. 402160의 값 그리고 aaaaaa]

00402160에 들어있는 값은 입력했던 Username인 "Osiris"인 것을 확인할 수 있었습니다. 그리고 가까운 곳에 Registration 코드로 입력했던 "aaaaaa"가 있는걸 확인하였습니다. 그리고 [그림4-5]에 나와있는 코드는 Username을 가지고 Registration 코드를 만듭니다. [그림4-5]의 코드들을 살펴보겠습니다.

0040119D movsx eax, byte ptr ds:[ecx+402160]
//[ecx+402160]의 값을 EAX Register에 넣습니다.
004011A4 cmp eax, 0
//EAX Register의 값을 0과 비교합니다.
004011A7 je short due-cm4.004011DB
//EAX Register의 값이 0과 같다면 004011DB로 분기합니다.
004011A9 mov esi, -1
//ESI Register를 FFFFFFFF으로 만듭니다.

004011AE cmp eax, 41 ①
//EAX Register의 값을 0x41과 비교합니다.

004011B1 jl short due-cm4.0040121A
//EAX Register의 값이 0x41보다 작다면 0040121A(에러 메시지)로 분기합니다.

004011B3 cmp eax, 7A ②
//EAX Register의 값을 0x7A와 비교합니다.

004011B6 ja short due-cm4.0040121A
//EAX Register의 값이 0x7A보다 크다면 0040121A(에러 메시지)로 분기합니다.

004011B8 cmp eax, 5A ③
//EAX Register의 값을 0x5A와 비교합니다.

004011BB jl short due-cm4.004011C0
//EAX Register의 값이 0x5A보다 작다면 004011C0로 분기합니다.
004011BD sub eax, 20
//EAX Register의 값에서 0x20을 뺍니다.

004011C0 inc esi
//ESI Register를 1증가시킵니다. 004011A9에서 FFFFFFFF였던 ESI Register의 값은 00000000이 됩니다.

제가 입력한 Username인 "Osiris"의 ASCII 코드값은 ['O = 0x4F'], ['s = 0x73'], ['i = 0x69'], ['r = 72'], ['i = 0x69'], ['s = 0x73'] 입니다. 'O = 0x4F'는 ①, ② 두 조건에는 해당하지 않지만 ③ 조건에는 해당되게 됩니다. 따라서 004011BB에서 004011BD를 진행하지 않고 004011C0로 바로 분기하게 됩니다.

Registers (FPU)			
EAX	0000004F		
ECX	00000000		
EDX	00140608		
EBX	00000000		
ESP	0012FBBC		
EBP	0012FBC8		
ESI	FFFFFFFF		
EDI	0012FC30		
EIP	004011BB	due-cm4.004011BB	

004011BB	7C 03	short due-cm4.004011C0
004011BD	83E8 20	sub eax, 20
004011C0	46	inc esi

[그림4-7. 004011BB에서 004011C0으로 분기와 그 때의 EAX Register의 값]

004011BB에서 004011C0으로 분기하게 되면서 ESI Register는 1증가하여 FFFFFFFF였던 ESI Register의 값이 00000000이 됩니다. 그리고 [그림4-5]의 004011C1에서 [esi(0x00)+402017]의 값이 EDX Register로 복사됩니다.

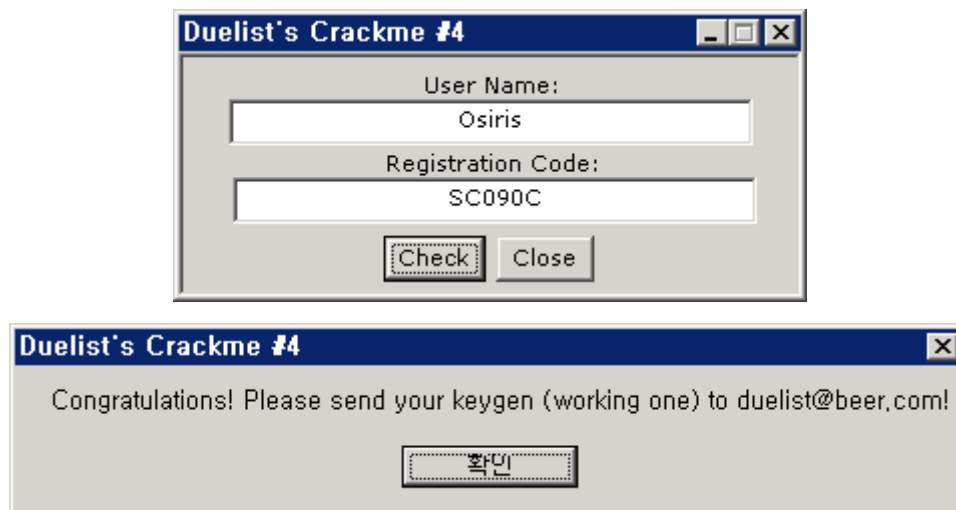
Address	Hex dump	ASCII
00402017	41 31 4C 53 4B 32 44 4A 46 34 48 47 50 33 51 57	A1LSK2DJF4HGP3QW
00402027	4F 35 45 49 52 36 55 54 59 5A 38 4D 58 4E 37 43	05EIR6UTYZ8MXN7C
00402037	42 56 39 00 20 53 55 37 43 53 4A 4B 46 30 39 4E	BV9, SU7CSJKF09N
00402047	43 53 44 4F 39 53 44 46 30 39 53 44 52 4C 56 4B	CSD09SDF09SDRLVK
00402057	37 38 30 39 53 34 4E 46 00 20 43 6F 6E 67 72 61	7809S4NF Congra

[그림4-8. 402017과 그 근처에 저장된 알 수 없는 값들]

[그림4-8]을 보니 004011C1에서 EDX Register로 복사되는 값은 [esi(0x00)+402017]의 '0x41 = A'인 것을 확인할 수 있습니다. 그리고 004011C8에서 EAX Register와 EDX Register의 값을 비교합니다. 두 Register의 값이 같지 않으므로 004011C0으로 분기하게 됩니다.

그러면 또 ESI Register를 1증가시키고 EDX Register에 [esi(0x01)+402017]의 값을 복사하며 EAX Register와 EDX Register가 같을 때까지 비교 후 분기를 하게 됩니다. 이렇게 ESI Register가 계속 증가하다 보면 00402027의 '0x4F = O'가 EDX Register에 들어가며 004011C8에서 EAX Register와 EDX Register를 비교했을 때, 두 Register의 값이 같으므로 004011C0으로 분기하지 않게 됩니다. (이 때의 ESI Register의 값은 10 입니다.) 그럼 004011CC에서 EAX Register에 [esi(0x10)+40203C]의 값인 'S = 0x53'을 복사합니다.

그리고 004011D3에서 EAX Register의 값을 [ecx(0x00)+402194]에 복사한 후 004011D9에서 0040119C로 분기하게 됩니다. 이런 식으로 그 다음 Username의 값인 's = 0x73'을 가지고 Registration 코드를 만들고 또 다음 Username의 값으로 그 다음 Registration 코드를 만들게 됩니다. 따라서 Username이 "Osiris"일 때 Registration 코드는 "SC090C"가 됩니다.



[그림4-9. Username과 Registration 코드 그리고 성공 메시지]

조건을 기준으로 해서 Keygen을 만들어 보았는데 도중에 이상한 점을 하나 발견했습니다. Username과 Registration 코드는 1:1 매칭이 되어야 하는데 알파벳이 아닌 " ` "같은 경우는 sub eax, 0x20 의 결과로 EAX Register의 값이 "0x40 = @"이 되어야 하지만 [그림4-8]에서 Username과 Registration 코드값으로 참조하는 값들 중에는 " @ "는 없었습니다.

```
00402087 65 6E 20 28 77 6F 72 6B 69 6E 67 20 6F 6E 65 29 en (working one)
00402097 20 74 6F 20 64 75 65 6C 69 73 74 40 62 65 65 72 to duelist@beer
```

[그림4-10. @ 를 찾았다]

다행스럽게도 성공 메시지박스가 사용하는 문자들 중에 메일주소가 있어서 " @ "를 찾았지만, CrackMe 제작자가 의도적으로 만들었다고 생각되지 않았기 때문에 Keygen은 알파벳만 가능하게끔 만들었습니다.

0x0003 결론

Username과 Registration 코드는 1:1로 매칭되는 값이라는걸 확인하였고 그 값들이 무엇인지 확인할 수 있었기 때문에 우리는 "Osiris"가 아닌 다른 Username의 Registration 코드를 만들 수 있게 Keygen을 만들어야 합니다.

```
// keygen_test.cpp : 콘솔응용프로그램에대한진입점을정의합니다.
//
#include "stdafx.h"
```

```

int _tmain(int argc, _TCHAR* argv[])
{
    char username[8];
    int registration_코드[8];
    int registration_table1[35] = {0x41, 0x31, 0x4C, 0x53, 0x4B, 0x32,
                                   0x44, 0x4A, 0x46, 0x34, 0x48, 0x47,
                                   0x50, 0x33, 0x51, 0x57, 0x4F, 0x35,
                                   0x45, 0x49, 0x52, 0x36, 0x55, 0x54,
                                   0x59, 0x5A, 0x38, 0x4D, 0x58, 0x4E,
                                   0x37, 0x43, 0x42, 0x56, 0x39};
    int registration_table2[35] = {0x53, 0x55, 0x37, 0x43, 0x53, 0x4A,
                                   0x4B, 0x46, 0x30, 0x39, 0x4E, 0x43,
                                   0x53, 0x44, 0x4F, 0x39, 0x53, 0x44,
                                   0x46, 0x30, 0x39, 0x53, 0x44, 0x52,
                                   0x4C, 0x56, 0x4B, 0x37, 0x38, 0x30,
                                   0x39, 0x53, 0x34, 0x4E, 0x46};

    int username_length;
    int not_char = 0;
    int counter = 0;
    int i, j;

usernameinput:
    printf("plz input username : ");
    gets(username);
    if(strlen(username) > 8 || strlen(username) == 0)
    {
        printf("username은1글자이상8글자이하로입력하십시오.\n");
        goto usernameinput;
    }

    username_length = strlen(username);

    //0x41 과0x7A 를검사하는부분
    for(i = 0; i < username_length; i++)
    {
        if (username[i] < 0x41 || username[i] > 0x7A)
        {
            not_char++;
        }
    }
    if(not_char >= 1)
    {
        printf("허용되지않는문자가입력되었습니다.\n");
        not_char = 0;
        goto usernameinput;
    }
    //0x41 과0x7A 를검사하는부분

    //0x5A 를검사및registration 생성
    for(i = 0; i < username_length; i++)
    {
        if(username[i] > 0x5A)
        {
            username[i] = username[i] - 0x20;
        }
        for (j = 0; j < 35; j++)
        {
            if (username[i] == registration_table1[j])
            {
                registration_코드[i] = registration_table2[j];
                break;
            }
        }
    }
    //registration_코드 출력
    for(i = 0; i < username_length; i++)
    {
        printf("%c", registration_코드[i]);
    }
    printf("\n");
    system("pause");
    return 0;
}

```

}

```
C:\WINDOWS\system32\cmd.exe
plz input username : Answer
S0C9F9
계속하려면 아무 키나 누르십시오 . . .
```

Duelist's Crackme #4

User Name:

Registration Code:

Duelist's Crackme #4

Congratulations! Please send your keygen (working one) to duelist@beer.com!

[그림4-11. Keygen을 이용한 정답확인]

0x05 Duelist #5

Crackme: <http://beist.org/research/public/crackme10/duel-cm5.zip>

Keygen: <http://beist.org/research/public/crackme10/duel-cm5-keygen.zip>

Author: Duelist

Level: ★

Protection: License 코드

0x0001 목표

Duelist's Crackme의 마지막인 #5는 3개의 목표가 있습니다.

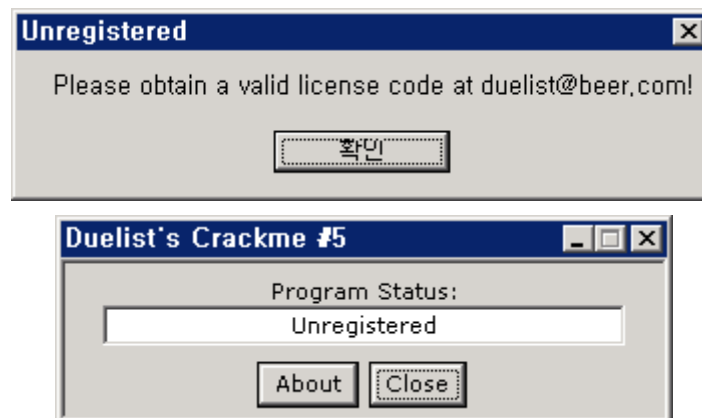
A. Unpacking

B. [Please obtain a ... duelist@beer.com] 메시지박스 안 뜨게 하기

C. Unregistered를 Registered로 바꾸기

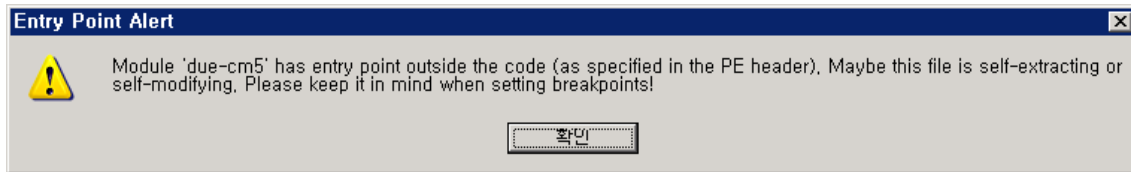
이렇게 총 3가지 입니다.

0x0002 분석 및 풀이



[그림5-1. Duelist's Crackme #5 실행 시 뜨는 메시지와 화면]

Duelist's Crackme#5를 실행하면 [그림5-1]처럼 메시지박스가 뜨고 난 후 Unregistered라는 텍스트가 보이며 실행이 완료됩니다. OllyDbg를 이용하여 열어보도록 하겠습니다.



[그림5-2. Entry Point Alert]

OllyDbg로 파일을 열려고 하자 [그림5-2]처럼 경고문구가 담긴 메시지박스가 화면에 보입니다. 내용을 보니 코드밖에 Entry Point를 가지고 있어서 Self-extracting이나 Self-modifying을 해야 할 것 같다고 합니다. 그냥 확인버튼을 눌러봅니다. 그럼 Duelist's Crackme #1~4까지 보던 코드들과는 비슷해 보이지도 않는 코드들만 잔뜩 있는 것을 볼 수 있습니다.

00406600	53	push ebx
00406601	51	push ecx
00406602	52	push edx
00406603	56	push esi
00406604	57	push edi
00406605	55	push ebp
00406606	E8 00000000	call due-cm5,0040660B
0040660B	5D	pop ebp
0040660C	81E0 34304000	sub ebp, due-cm5,00403034
00406612	FF95 E8344000	call near dword ptr ss:[ebp+4034E8]
00406618	B8 29304000	mov eax, due-cm5,00403029
0040661D	03C5	add eax, ebp
0040661F	2B85 00344000	sub eax, dword ptr ss:[ebp+403400]
00406625	8985 19344000	mov dword ptr ss:[ebp+403419], eax
0040662B	83BD 01344000	cmp dword ptr ss:[ebp+403401], 0
00406632	75 1D	jnz short due-cm5,00406651
00406634	90	nop
00406635	90	nop
00406636	90	nop
00406637	90	nop
00406638	C785 01344000	mov dword ptr ss:[ebp+403401], 1
00406642	E8 1E000000	call due-cm5,00406665
00406647	E8 25020000	call due-cm5,00406871
0040664C	E8 DA020000	call due-cm5,0040692B
00406651	8B85 05344000	mov eax, dword ptr ss:[ebp+403405]
00406657	0385 19344000	add eax, dword ptr ss:[ebp+403419]

[그림5-3. 좀 생소해 보이는 코드들]

천천히 F8을 누르면서 진행해 봅니다.

00406642	E8 1E000000	call due-cm5,00406665	
00406647	E8 25020000	call due-cm5,00406871	
0040664C	E8 DA020000	call due-cm5,0040692B	
00406651	8B85 05344000	mov eax, dword ptr ss:[ebp+403405]	
00406657	0385 19344000	add eax, dword ptr ss:[ebp+403419]	
0040665D	5D	pop ebp	
0040665E	5F	pop edi	
0040665F	5E	pop esi	
00406660	5A	pop edx	
00406661	59	pop ecx	
00406662	5B	pop ebx	
00406663	- FFE0	jmp near eax	due-cm5,00401000
00406665	8D85 7A344000	lea eax, dword ptr ss:[ebp+40347A]	
0040666B	50	push eax	

Registers (FPU)		
EAX	00401000	due-cm5,00401000
ECX	7C816FF7	kernel32,7C816FF7
EDX	7FFDF000	
EBX	7C940738	ntdll,7C940738
ESP	0012FFCC	
EBP	FFFFFFFF	
ESI	0012FFB0	
EDI	7C93EB94	ntdll,KiFastSystemCallRet
EIP	00406663	due-cm5,00406663

[그림5-4. jmp near eax와 Registers (FPU)]

F8로 천천히 내려오다가 보면 00406663의 코드에서 EAX Register가 가지고 있는 값으로 분기하게 됩니다. 이곳은 실제 코드가 존재하는 곳입니다.

00401000	6A	db 6A	CHAR 'j'
00401001	00	db 00	
00401002	E8	db E8	
00401003	8C	db 8C	
00401004	01	db 01	
00401005	00	db 00	
00401006	00	db 00	
00401007	A3	db A3	
00401008	0F	db 0F	
00401009	21	db 21	CHAR 'I'
0040100A	40	db 40	CHAR '@'
0040100B	00	db 00	
0040100C	C7	db C7	
0040100D	05	db 05	
0040100E	E3	db E3	
0040100F	20	db 20	CHAR ' '
00401010	40	db 40	CHAR '@'
00401011	00	db 00	
00401012	03	db 03	
00401013	40	db 40	CHAR '@'

[그림5-5. 알 수 없는 코드들의 집합]

그런데 00406663에서 분기하여 00401000으로 왔는데 코드를 보니 실제 코드는 보이지 않고 이상한 코드들만 보입니다. 이 때 Ctrl + A를 누르게 되면 코드들을 분석해주게 됩니다.

00401000	6A 00	push 0	pModule = NULL
00401002	E8 8C010000	call due-cm5,00401193	GetModuleHandleA
00401007	A3 0F214000	mov dword ptr ds:[40210F], eax	
0040100C	C705 E3204000 03400000	mov dword ptr ds:[4020E3], 4003	
00401016	C705 E7204000 95104000	mov dword ptr ds:[4020E7], due-cm5,00401000	
00401020	C705 EB204000 00000000	mov dword ptr ds:[4020EB], 0	
0040102A	C705 EF204000 00000000	mov dword ptr ds:[4020EF], 0	
00401034	A1 0F214000	mov eax, dword ptr ds:[40210F]	
00401039	A3 F3204000	mov dword ptr ds:[4020F3], eax	
0040103E	6A 01	push 1	RsrcName = 1,
00401040	50	push eax	hInst => NULL
00401041	E8 3D020000	call due-cm5,00401283	LoadIconA
00401046	A3 F7204000	mov dword ptr ds:[4020F7], eax	
0040104B	68 007F0000	push 7F00	RsrcName = IDC_ARROW
00401050	6A 00	push 0	hInst = NULL
00401052	E8 38020000	call due-cm5,0040128F	LoadCursorA
00401057	A3 FB204000	mov dword ptr ds:[4020FB], eax	
0040105C	EB 63	jmp short due-cm5,004010C1	
0040105E	6A 00	push 0	MsgFilterMax = 0
00401060	6A 00	push 0	MsgFilterMin = 0
00401062	6A 00	push 0	hWnd = NULL
00401064	68 C7204000	push due-cm5,004020C7	pMsg = due-cm5,004020C7
00401069	E8 57020000	call due-cm5,004012C5	GetMessageA

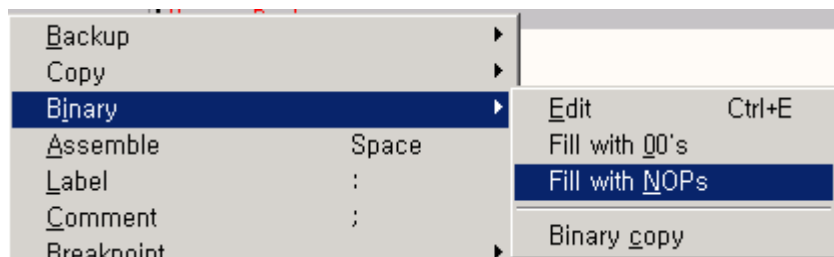
[그림5-6. 알 수 없는 코드들이 분석되어 알 수 있는 코드로 변함]

보기 좋게 분석이 완료 된 것을 확인할 수 있습니다. 이로써 첫 번째 목표인 Unpack이 완료 되었습니다.

004010C1	>	68 00200000	push 2000	[Style = MB_OK MB_TASKMODAL Title = "Unregistered" Text = "Please obtain a valid hOwner = NULL MessageBoxA
004010C6	,	68 5C204000	push due-cm5,0040205C	
004010CB	,	68 17204000	push due-cm5,00402017	
004010D0	,	6A 00	push 0	
004010D2	,	E8 94010000	call due-cm5,0040126B	

[그림5-7. 메시지박스호출]

두 번째 목표인 [Please obtain a ... duelist@beer.com] 메시지박스 제거를 하도록 하겠습니다. [그림5-7]에 보면 004010D2에서 메시지박스를 호출하게 됩니다. 이 코드를 호출되지 않게끔 수정해주면 제거가 완료 됩니다.



[그림5-8. NOP으로 가득채우기]

[그림5-8]처럼 해당 코드 line에서 우 클릭 후 Binary => Fill with NOPs를 선택하면 자동으로 해당 코드 line을 NOP 코드로 채워줍니다.

004010C1	>	68 00200000	push 2000	[Style = MB_OK MB_TASKMODAL Title = "Unregistered" Text = "Please obtain a valid hOwner = NULL MessageBoxA
004010C6	,	68 5C204000	push due-cm5,0040205C	
004010CB	,	68 17204000	push due-cm5,00402017	
004010D0	,	6A 00	push 0	
004010D2		90	nop	
004010D3		90	nop	
004010D4		90	nop	
004010D5		90	nop	
004010D6		90	nop	
004010D7	,	6A 00	push 0	[lParam = NULL DlgProc = due-cm5,004010B8 hOwner = NULL pTemplate = 1 hInst = NULL DialogBoxParamA
004010D9	,	68 B8104000	push due-cm5,004010B8	
004010DE	,	6A 00	push 0	
004010E0	,	6A 01	push 1	
004010E2	,	FF35 0F214000	push dword ptr ds:[40210F]	
004010E8	,	E8 12010000	call due-cm5,004011FF	

[그림5-9. NOP 한 가독]

이로써 메시지박스를 제거하는 두 번째 목표도 완료하였습니다. 이제 남은 건 세 번째 목표인 Unregistered문구를 Registered로 수정하는 것입니다.

00401113	>	68 00200000	push 2000	{ Style = MB_OK MB_TASKMODAL Title = "Duelist's Crackme" Text = "Please send your pe hOwner = NULL MessageBoxA IParam = 40205C wParam = 0 Message = WM_SETTEXT ControlID = 3 hWnd SendDlgItemMessageA }
00401118	.	68 01204000	push due-cm5,00402001	
0040111D	.	68 6A204000	push due-cm5,0040206A	
00401122	.	6A 00	push 0	
00401124	.	E8 42010000	call due-cm5,0040126B	
00401129	.	B8 01000000	mov eax, 1	
0040112E	^	EB DC	jmp short due-cm5,0040110C	
00401130	>	68 5C204000	push due-cm5,0040205C	
00401135	.	6A 00	push 0	
00401137	.	6A 0C	push 0C	
00401139	.	6A 03	push 3	
0040113B	.	FF75 08	push dword ptr ss:[ebp+8]	
0040113E	.	E8 3A010000	call due-cm5,0040127D	

[그림5-10. 00401130]

[그림5-9]에서 NOP작업을 끝낸 후 조금 더 내려오다 보면 [그림5-10]에 나오는 코드들을 만날 수 있습니다. 00401130의 코드를 보면 40205C의 값을 사용하는 것을 확인할 수 있는데 그곳을 찾아가보면 다음과 같습니다.

Address	Hex dump	ASCII
0040205C	55 6E 72 65 67 69 73 74 65 72 65 64 00 20 50 6C	Unregistered, Pl
0040206C	65 61 73 65 20 73 65 6E 64 20 79 6F 75 72 20 70	ease send your p

[그림5-11. 0040205C Unregistered]

Unregistered를 Registered로 수정하는 방법은 2가지가 정도가 있습니다. 하나는 0040205C의 값 자체를 Registered로 수정하는 것과 또 다른 하나는 00401130에서 push하는 0040205C를 다른 주소(Registered를 가진)로 바꿔주는 것입니다. 전 후자를 선택하겠습니다.

Address	Hex dump	ASCII
00402050	52 65 67 69 73 74 65 72 65 64 00 20 55 6E 72 65	Registered, Unre
00402060	67 69 73 74 65 72 65 64 00 20 50 6C 65 61 73 65	gistered, Please

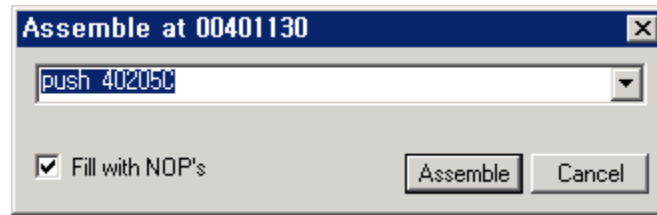
[그림5-12. Registered와 Unregistered]

Unregistered 근처에 Registered가 저장되어 있는데 그것을 활용하는 것입니다. 아마 제작자가 사용하라고 만들어 놓은 것 같습니다. 간단한 작업으로 변경할 수 있는데요, 00401130 코드 line에서 Assembly가 있는 곳에서 더블클릭을 하게 되면 [그림5-14]와 같은 창이 뜹니다.

이 곳을 더블클릭합니다.

00401130	>	68 5C204000	push due-cm5,0040205C	{ IParam = 40205C wParam = 0 Message = WM_SETTEXT ControlID = 3 hWnd SendDlgItemMessageA }
00401135	.	6A 00	push 0	
00401137	.	6A 0C	push 0C	
00401139	.	6A 03	push 3	
0040113B	.	FF75 08	push dword ptr ss:[ebp+8]	
0040113E	.	E8 3A010000	call due-cm5,0040127D	

[그림5-13. 이곳을 더블클릭 합니다]



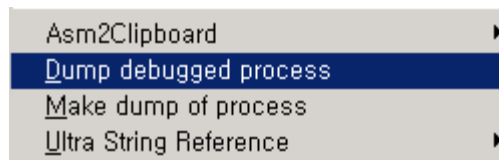
[그림5-14. Assemble at xxxxxxxx]

[그림5-14]에서 push 40205C를 push 402050으로 변경해주고 Assemble버튼을 클릭합니다. 그러면 수정이 완료됩니다.

00401130	68 50204000	push due-cm5,00402050	ASCII "Registered" wParam = 0 Message = WM_SETTEXT ControlID = 3 hWnd SendMessageA
00401135	, 6A 00	push 0	
00401137	, 6A 0C	push 0C	
00401139	, 6A 03	push 3	
0040113B	, FF75 08	push dword ptr ss:[ebp+8]	
0040113E	, E8 3A010000	call due-cm5,00401270	

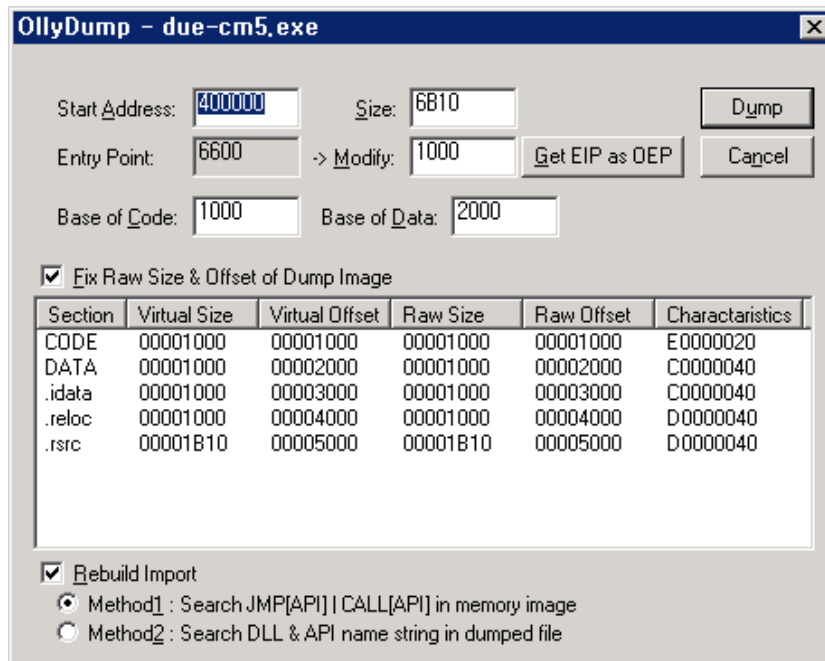
[그림5-15. 변경된 00401130]

이제 Unpack과 수정작업이 완료된 코드를 실행파일로 저장해야 합니다. 코드가 있는 곳 아무데서나 우 클릭 후 Dump debugged process 메뉴를 선택합니다. (그런데 Dump debugged process는 OllyDbg의 Plug-in입니다. 따라서 순수 OllyDbg에서는 메뉴를 찾을 수 없으므로 해당 Plug-in을 첨부파일로 올리도록 하겠습니다.)



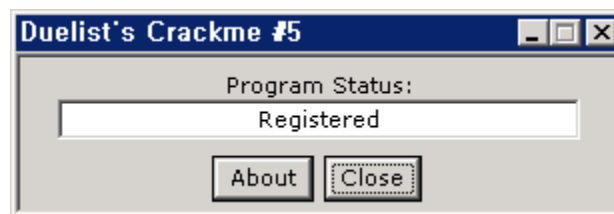
[그림5-16. Dump debugged process]

Dump debugged process 메뉴를 선택하게 되면 다음과 같은 창이 뜹니다.



[그림5-17. Dump창]

그럼 Cancel버튼 왼쪽에 있는 Get EIP as OEP버튼을 누른 후 Dump버튼을 누릅니다. 그럼 파일을 저장 할 수 있게끔 되는데 저는 due-cm5_clear.exe라는 이름으로 저장하였습니다. 그리고 저장된 파일을 찾아가 실행해 보았습니다.



[그림5-18. Registered]

처음에 뜨는 메시지박스는 온데간데 없이 사라졌고 Unregistered문구도 Registered로 변경된 것을 확인할 수 있었습니다.

//추가 Memory Loader

Unpack만 된 Duelist's Crackme #5를 이용해서 Memory Loader를 만들어 보도록 하겠습니다.

1. Loader의 동작

1-1. 프로세서를 생산하고 타겟이 되는 프로그램을 실행하여야 합니다. 이를 위해 사용되는 API가 CreateProcess입니다. MSDN에 나와있는 CreateProcess API를 살펴보면 다음과 같

습니다.

```
BOOL CreateProcess(
    LPCTSTR lpApplicationName,
    // 타겟프로그램의 경로 및 파일명
    LPTSTR lpCommandLine,
    // 타겟이 실행할 때 주어지는 파라미터를 설정하기 위해 사용(로더만들 때는 NULL로 설정)
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    BOOL bInheritHandles,
    DWORD dwCreationFlags,
    // 로더로 타겟을 메모리에 로드시켰을 때, 해당 프로세스를 정지시켜주기 위한 설정에 사용(CREATE_SUSPENDED 상수 사용)
    LPVOID lpEnvironment,
    LPCTSTR lpCurrentDirectory,
    LPSTARTUPINFO lpStartupInfo,
    // CreateWindow함수나 ShowWindow함수와 같이 main window의 속성을 명시하기 위해 CreateProcess와 같이 사용되는 구조체
    LPPROCESS_INFORMATION lpProcessInformation
    // 타겟이 메모리에 로드되었을 때, 프로세스정보가 채워지는 구조체 (process handle, thread handle, process/thread ID를 가지고 있음)
);
```

1-2. 타겟이 메모리에 로드 된 후 해당 프로세스를 중지시켜야 합니다.

이를 위한 API는 다음과 같습니다.

```
DWORD ResumeThread(
    HANDLE hThread // 정지된 Thread를 다시 실행시키는 일을 합니다.
);

DWORD SuspendThread(
    HANDLE hThread // 실행중인 Thread를 정지시키는 일을 합니다.
);

위에서 사용되는 hThread handle정보는 LPPROCESS_INFORMATION 구조체가 가지고 있습니다.
```

1-3. 이제 우리가 원하는 작업인 패치를 할 수 있게 됩니다.

패치를 하기 위해서는 메모리를 읽고 쓰는 API를 사용합니다.

Write API:

```

BOOL WriteProcessMemory(
    HANDLE hProcess, // Handle을 적는 곳입니다.
    LPVOID lpBaseAddress, // Address를 적는 곳입니다.
    LPCVOID lpBuffer, // 쓸 값을 적는 곳입니다.
    SIZE_T nSize, // 쓸 값의 크기를 적는 곳입니다. (byte)
    SIZE_T* lpNumberOfBytesWritten // 데이터를 몇 개 썼는지 출력시켜줍니다. (보통 사용하지 않으므로 NULL값을 줍니다.)
);

```

Read API:

```

BOOL ReadProcessMemory(
    HANDLE hProcess, // Handle을 적는 곳입니다.
    LPCVOID lpBaseAddress, // Address를 적는 곳입니다.
    LPVOID lpBuffer, // 읽어온 값을 저장할 변수를 적는 곳입니다.
    SIZE_T nSize, // 읽어올 값의 크기를 적는 곳입니다. (byte)
    SIZE_T* lpNumberOfBytesRead // 데이터를 몇 개 읽었는지 출력시켜줍니다. (보통 사용하지 않으므로 NULL값을 줍니다.)
);

```

2. Loader만들기

우선 언팩된 CrackMe를 OllyDbg로 열고 패치 할 곳의 주소를 찾아야 합니다. 메시지박스를 호출하는 부분과 Unregistered가 있는 부분입니다.

004010C1	>	68 00200000	push 2000	<div style="border: 1px solid black; padding: 5px;"> Style = MB_OKIMB_TASKMODAL Title = "Unregistered" Text = "Please obtain a val hOwner = NULL MessageBoxA </div>
004010C6	,	68 5C204000	push due-cm5_.0040205C	
004010CB	,	68 17204000	push due-cm5_.00402017	
004010D0	,	6A 00	push 0	
004010D2	,	E8 94010000	call <jmp,&USER32,MessageBoxA>	MessageBoxA

[그림5-19. 004010D2의 메시지박스 호출]

Address	Hex dump	ASCII
00402050	52 65 67 69 73 74 65 72 65 64 00 20 55 6E 72 65	Registered, Unre
00402060	67 69 73 74 65 72 65 64 00 20 50 6C 65 61 73 65	gistered, Please

[그림5-20. 00402050의 Registered와 0040205C의 Unregistered]

일단 Loader의 소스를 보면서 설명을 드리겠습니다.

```

#include <stdio.h>
#include <windows.h>

char Filename[] = ".\\due-cm5_test.exe";
//due-cm5_test.exe는언팩만된상태의Crackme입니다.

```



```

char notloaded[] = "due-cm5_test.exe가실행이안되었습니다~";
char Letsgo[] = "패치가완료되었습니다.";
char Registered[11];
//402050의 값을 복사하기 위한 곳.
char Noperation[] = "9090909090";
//메시지박스를 NOP시키기 위한 것.

STARTUPINFO startupinfo;
PROCESS_INFORMATION processinfo;

unsigned long bytewritten;
int uExit코드;

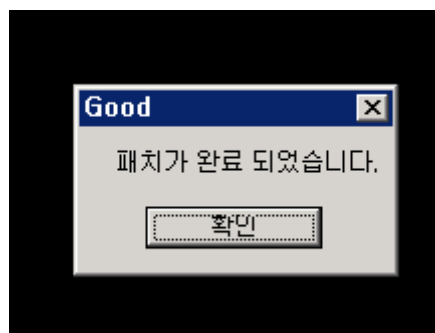
void main()
{
    //processinfo, startupinfo 초기화
    memset(&processinfo, 0, sizeof(PROCESS_INFORMATION));
    memset(&startupinfo, 0, sizeof(STARTUPINFO));
    startupinfo.cb = sizeof(STARTUPINFO);
    //프로세스를생성시키고, due-cm5_test를 로드후정지(CREATE_SUSPENDED)
    BOOL bRes = CreateProcess(Filename, NULL, NULL, NULL, FALSE,
    CREATE_SUSPENDED, NULL, NULL, &startupinfo, &processinfo);

    if(bRes == NULL) //프로세스가 생성되었는지 확인
        메시지박스(NULL, notloaded, NULL, MB_ICONEXCLAMATION);
    else{
        메시지박스(NULL, Letsgo, "Good", MB_OK);
        WriteProcessMemory(processinfo.hProcess, (LPVOID)0x004010D2,
        Noperation, 5, NULL); //0x004010D2에 9090909090를 쓴다. ①

        ReadProcessMemory(processinfo.hProcess, (LPVOID)0x00402050,
        Registered, 11, NULL); //0x00402050의 값을 Registered에 저장. ②
        WriteProcessMemory(processinfo.hProcess, (LPVOID)0x0040205C,
        Registered, 11, NULL); //0x0040205C에 Registered에 저장된 값을 쓴다. ③
        ResumeThread(processinfo.hThread); //정지된 프로세스를 다시 실행
    }
    ExitProcess(1);
}

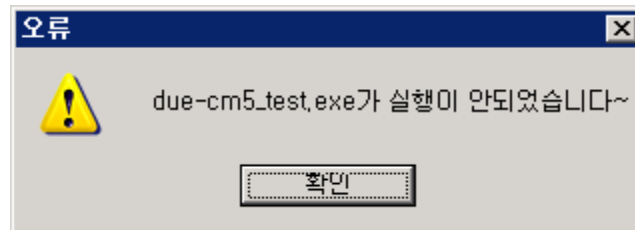
```

소스를 컴파일 한 후 loader.exe를 만들어서 실행시켜 보겠습니다.



[그림5-21. 패치가 완료 되었습니다]

생성된 파일인 loader.exe를 due-cm5_test.exe가 있는 곳에서 실행시키면 [그림5-21]과 같은 메시지박스가 뜨게 되는데 확인을 누르면 패치가 적용되어 메시지박스가 제거되고 Unregistered가 Registered로 변하게 됩니다. 그리고 만약 due-cm5_test.exe파일이 없는 상태에서 loader.exe를 실행시키게 되면 다음과 같은 메시지박스가 뜨게 됩니다.



[그림5-22. 실행이 안되었습니다]

생성된 loader.exe를 실행시켰을 때 결과를 먼저 얘기하였는데, 이제 그 과정을 설명하도록 하겠습니다.

WriteProcessMemory(processinfo.hProcess, (LPVOID)0x004010D2, Noperation, 5, NULL);
//0x004010D2에 909090909를 쓴다. ①

[그림19]에서 0x004010D2를 보면 E8 94010000 의 Op코드를 확인할 수 있습니다. 총 5byte 인데, 이것들을 NOP로 바꿔야지 메시지박스가 호출되지 않게 됩니다. E8 94010000를 90으로 다 바꿔야되는데 총 5byte이니까 90 90909090으로 채우면 되겠습니다. 그래서 Noperation변수를 보면 값이 909090909인 것 입니다.

004010C1	>	68 00200000	push 2000	[Style = MB_OK MB_TASKMODAL Title = "Unregistered" Text = "Please obtain a val hOwner = NULL MessageBoxA
004010C6	,	68 5C204000	push due-cm5_..0040205C	
004010CB	,	68 17204000	push due-cm5_..00402017	
004010D0	,	6A 00	push 0	
004010D2		90	nop	
004010D3		90	nop	
004010D4		90	nop	
004010D5		90	nop	
004010D6		90	nop	

이 그림과 같은 의미입니다. 그래서 메시지박스는 호출되지 않게 됩니다.

ReadProcessMemory(processinfo.hProcess, (LPVOID)0x00402050, Registered, 11, NULL);

//0x00402050의 값을 Registered에 저장. ②

0x00402050의 값을 11byte만큼 읽어서 Registered에 저장합니다.

Address	Hex dump	ASCII
00402050	52 65 67 69 73 74 65 72 65 64 00 20 55 6E 72 65	Registered, Unre
00402060	67 69 73 74 65 72 65 64 00 20 50 6C 65 61 73 65	gistered, Please

위의 그림에서 보면 알 수 있듯이 0x00402050에서 11byte를 읽게 되면 "52 65 67 69 73 74 65 72 65 64"에 NULL Byte인 00까지 읽어오게 됩니다. 그리고 Registered변수에 저장하게 됩니다.

```
WriteProcessMemory(processinfo.hProcess, (LPVOID)0x0040205C, Registered, 11, NULL);
```

//0x0040205C에 Registered에 저장된 값을 쓴다. ③

그리고나서 0x0040205C에 Registered변수가 가지고 있는 값을 11byte만큼 쓰게 됩니다. "52 65 ... 65 64 00"까지 0x0040205C에 쓰게 되는겁니다. Crackme에서 출력되는 String은 NULL Byte이전 까지 이므로 Registered변수에 저장된 11byte의 값이 0x0040205C가 가지고 있는 값의 길이보다 작다고 할지라도 상관이 없습니다. Unregistered니까 2byte 길군요.

이렇게 해서 Duelist's Crackme #5의 Memory loader가 완료 되었습니다.

0x0003 결론

언팩을 하는 방법 중 처음에 설명했던 것처럼 코드를 한 줄 한 줄 실행하다 보면 실제 코드가 있는 곳으로 분기하는 경우도 있다는 경우를 어디선가 들은 적이 있었습니다. 그 어디선가 주어들은 것을 가지고 이번 CrackMe를 해결하게 되었는데, 모든 Pack이 이런 식으로 언팩이 된다면 말이 안되겠죠. 실행압축이란 것에 대해서 많은걸 공부해야 될 것 입니다.

그리고 Memory Loader를 짧게나마 공부하면서 느낀 것이 있다면 뭐든지 응용한다면 재미있는게 무궁무진 하다는 겁니다.

0x06 keygenning4newbies #1

Crackme: <http://beist.org/research/public/crackme10/k4n.zip>

Keygen: <http://beist.org/research/public/crackme10/k4n-keygen.zip>

Author: analys

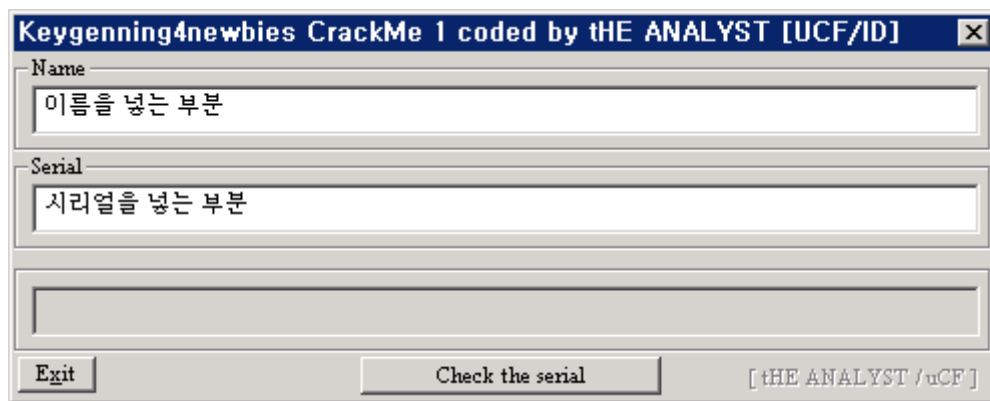
Level: ★

Protection: Name / Serial

0x0001 목표

이번 CrackMe는 분석을 통해 입력된 Name에 맞는 Serial을 찾아야합니다. 그리고 입력된 Name을 가지고 어떻게 Serial을 생성하는지 확인한 다음 Serial을 생성하는 Keygen을 만들어 봅니다.

0x0002 분석 및 풀이



[그림6-1. Keygenning4newbies #1 실행화면]

Name과 Serial을 넣고 Check the serial버튼을 누르면 시리얼 아래 비활성화된 컨트롤에 그 상황에 맞는 결과를 알려줍니다. 입력되지 않았거나 입력되었지만 틀렸거나 축하한다는 등의 문구를 볼 수 있습니다.

OllyDbg로 Breakpoint를 설정하고, Name에는 Osiris를 넣고 Serial에는 aaaaaa를 넣어서 Check the serial버튼을 누르겠습니다.

0040110C	> 0FBE840D 48F1	movsx eax, byte ptr ss:[ebp+ecx-B8]	
00401114	. 41	inc ecx	
00401115	. 33C1	xor eax, ecx	
00401117	. 03D8	add ebx, eax	
00401119	. 3B4D D8	cmp ecx, [local.10]	
0040111C	. ^ 75 EE	jnz short k4n.0040110C	
0040111E	. 6BC0 06	imul eax, eax, 6	
00401121	. C1E3 07	shl ebx, 7	
00401124	. 03C3	add eax, ebx	
00401126	. 8945 C8	mov [local.14], eax	
00401129	. FF75 C8	push [local.14]	
0040112C	. 68 38B44000	push k4n.0040B438	Arg3
00401131	. 8D8D 80FEFF	lea ecx, [local.96]	Arg2 = 0040B438 ASCII "%IX"
00401137	. 51	push ecx	Arg1
00401138	. E8 873D0000	call k4n.00404EC4	k4n.00404EC4
0040113D	. 83C4 0C	add esp, 0C	
00401140	. 8D85 80FEFF	lea eax, [local.96]	
00401146	. 50	push eax	String2
00401147	. 8D95 E4FEFF	lea edx, [local.71]	String1
0040114D	. 52	push edx	lstrcmpA
0040114E	. E8 339C0000	call <jmp.&KERNEL32.lstrcmpA>	
00401153	. 85C0	test eax, eax	
00401155	. ^ 75 00	jnz short k4n.00401164	
00401157	. 68 3CB44000	push k4n.0040B43C	Text = "Congratulations! IF
0040115C	. 56	push esi	hWnd
0040115D	. E8 289B0000	call <jmp.&USER32.SetWindowTextA>	SetWindowTextA
00401162	. ^ EB 18	jmp short k4n.0040117C	
00401164	. > 68 90B44000	push k4n.0040B490	Text = "This serial is *NOT:
00401169	. 56	push esi	hWnd
0040116A	. E8 1B9B0000	call <jmp.&USER32.SetWindowTextA>	SetWindowTextA
0040116F	. ^ EB 08	jmp short k4n.0040117C	
00401171	. > 68 C9B44000	push k4n.0040B4C9	Text = "Name must contain m
00401176	. 56	push esi	hWnd
00401177	. E8 0E9B0000	call <jmp.&USER32.SetWindowTextA>	SetWindowTextA

[그림6-2. 주 코드들]

0040110C movsx eax, byte ptr ss:[ebp+ecx-B8]

//[ebp+ecx-B8]의 값을 EAX Register에 넣습니다.

00401114 inc ecx

//ECX Register를 1증가 시킵니다.

00401115 xor eax, ecx

//EAX Register와 ECX Register를 XOR시킵니다.

00401117 add ebx, eax

//EBX = EBX + EAX

00401119 cmp ecx, [local.10]

//ECX Register와 입력한 문자의 길이를 비교한다.

0040111C jnz short k4n.0040110C

//00401119의 비교결과에 따라 0040110C로 분기하거나 다음 코드를 진행한다.

0040110C~0040111C까지의 코드는 입력한 Name의 길이만큼 반복됩니다. Name에 "Osiris"를 입력했으므로 입력한 문자의 길이는 6이 됩니다. 총 6번만큼 0040110C~0040111C까지의 코드가 반복됩니다.

40110C 'O'를 읽어와 EAX Register 에 0x4F를 넣습니다.
 401114 ECX Register를 1증가 시킵니다.(입력한 문자의 개수만큼 루프를 돌기 위함입니다)
 401115 EAX = EAX ^ ECX를 합니다. (ECX는 1 EAX는 4F 이므로 결과는 4E가 됩니다)
 401117 EBX = EBX + EAX를 합니다. (EBX는 0 EAX는 4E 이므로 결과는 4E가 됩니다)
 401119 ECX Register와 입력한 문자의 개수인 6을 비교합니다.
 40111C 비교결과에 따라 분기합니다.

40110C 's'를 읽어와 EAX Register 에 0x73 을 넣습니다.
 401114 ECX Register를 1증가 시킵니다. (입력한 문자의 개수만큼 루프를 돌기 위함입니다)
 401115 EAX = EAX ^ ECX를 합니다. (ECX는 2 EAX는 73 이므로 결과는 71이 됩니다)
 401117 EBX = EBX + EAX를 합니다. (EBX는 4E EAX는 71 이므로 결과는 BF가 됩니다)
 401119 ECX Register와 입력한 문자의 개수인 6을 비교합니다.
 40111C 비교결과에 따라 분기합니다.

이렇게 0040110C~0040111C의 루프가 끝날 때까지 입력한 Name값의 문자 하나 하나를 읽어와 연산하며 그 결과는 EBX Register에 저장되게 됩니다. 그리고 EAX Register에는 입력한 Name의 마지막 문자 값인 's'와 ECX Register값인 6이 XOR연산되어 저장되게 됩니다. (0x73 XOR 0x06 = 0x75)

0040111E EAX = EAX(0x75) * 0x06을 하여 EAX Register는 0x2BE라는 값을 가지게 됩니다.
 00401121 EBX Register를 왼쪽으로 7칸 Shift연산을 하게 됩니다.

Registers (FPU)		
EAX	000002BE	
ECX	00000006	
EDX	00000000	
EBX	00000280	
ESP	0012F588	
EBP	0012FA68	
ESI	000C0282	
EDI	0012F5F8	
EIP	00401121	k4n.00401121

00401121		C1E3 07		shl	ebx,	7
----------	--	---------	--	-----	------	---

[그림6-3. 00401121 Shift 연산과 그 상황의 Registers (FPU)]

0x280이던 EBX Register가 왼쪽으로 7칸 Shift연산을 하게 되면, EBX Register는 0x00014000이 됩니다.
 00401124 EAX = EAX(0x2BE) + EBX(0x00014000)를 하여 EAX Register는 0x000142BE가 됩니다. 그리고 나서 EAX Register의 값을 스택에 넣습니다. (이 값이 입력된 Name으로 만들어진 Serial값 입니다.)

00401146	, 50	push eax	String2 String1 IstrcmpA
00401147	, 8D95 E4FEFF	lea edx, [local,71]	
0040114D	, 52	push edx	
0040114E	, E8 339C0000	call <jmp,&KERNEL32,IstrcmpA>	
00401153	, 85C0	test eax, eax	
00401155	, 75 0D	jmp short k4n,00401164	

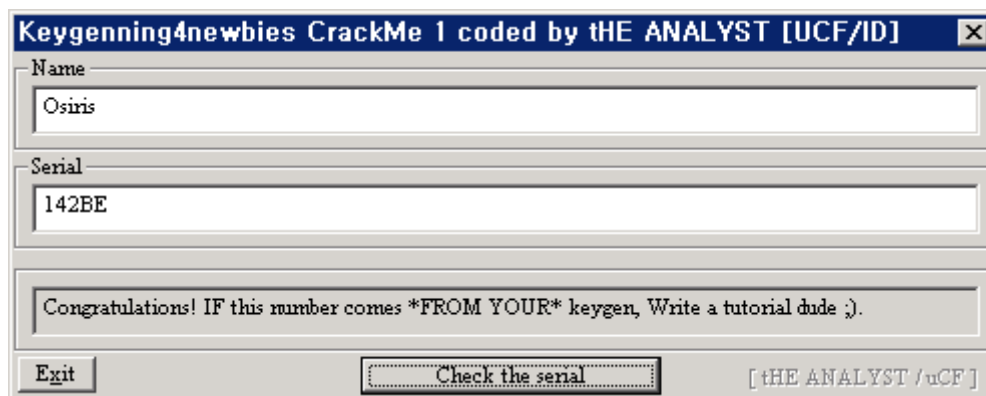
[그림6-4. 생성된 Serial과 입력된 Serial을 비교하여 알맞은 메시지로 분기]

[그림6-4]에 보이는 String2에는 입력된 Name으로 생성된 Serial(142BE)이 들어가며, String1에는 처음에 입력한 Serial(aaaaaa)값이 들어가게 됩니다. 생성된 Serial과 입력한 Serial이 다르므로 0040114E의 호출문이 끝난 후 EAX Register는 1이 되고 00401155에서 00401164로 분기하게 됩니다. 만약 생성된 Serial과 입력한 Serial이 같다면 0040114E의 호출문이 끝난 후 EAX Register는 0이 되고, 00401155에서 00401164로 분기하지 않고 00401157로 진행하여 성공메시지를 볼 수 있게 됩니다.

00401157	, 68 3CB44000	push k4n,0040B43C	Text = "Congratulations! IF this hWnd SetWindowTextA
0040115C	, 56	push esi	
0040115D	, E8 289B0000	call <jmp,&USER32,SetWindowTextA>	

[그림6-5. 성공메시지]

그럼 Name에 'Osiris'를 넣고 Serial에 '142BE'를 넣고 Check the serial버튼을 눌러보겠습니다.



[그림6-6. Congratulations!]

성공하였습니다 !!!

0x0003 결론

어떻게 입력된 Name으로 Serial이 생성되는지 알았으니 Keygen을 만들어야 합니다. Serial 생성 코드가 복잡하지 않으므로 쉽게 할 수 있습니다.

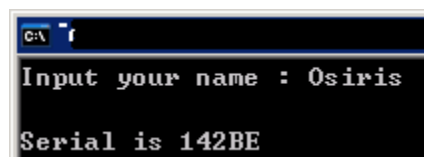
```
#include "stdafx.h"
#include <windows.h>

int _tmain(int argc, _TCHAR* argv[])
{
    char name[10];
    int count = 0;
    int temp;
    int i = 0;
    int answer = 0;

    printf("Input your name : ");
    gets(name);
    count = strlen(name);

    for (i = 0; i < count; i++)
    {
        temp = name[i] ^ i+1;
        answer = answer + temp;
    }
    temp = temp * 0x6;
    answer = (answer << 7) + temp;

    printf("\nSerial is %X\n", answer);
    system("pause");
    return 0;
}
```



[그림6-7. 생성된 Serial]

0x07 CaD's Crack Me #1

Crackme: <http://beist.org/research/public/crackme10/cad-crackme1.zip>

Keygen: -

Author: CaD

Level: ★

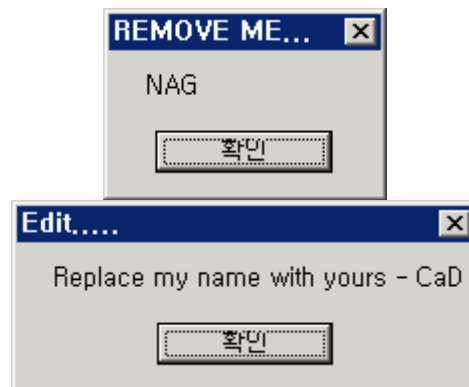
Protection: None

0x0001 목표

실행하면 처음에 뜨는 메시지박스를 제거합니다. 그리고 그 다음에 뜨는 메시지박스를 수정합니다. 마지막으로 앞의 두 가지를 한방에 해결 주는 패치를 만들어봅니다.

0x0002 분석 및 풀이

이번 CrackMe는 이전에 풀이하던 CrackMe와 특별히 다른 건 없지만 패치를 하는 내용을 다루게 됩니다. 패치는 원본 코드를 목적에 맞게 수정 해주는 프로그램을 말합니다. 일단 Crackme를 실행해보겠습니다.



[그림7-1. 제거해야 되는 메시지박스와 수정해야 되는 메시지박스]

일단 OllyDbg를 이용해서 열어보겠습니다.

00401000	\$ 6A 00	push 0	Style = MB_OK MB_APPLMODAL
00401002	, 68 00304000	push Crackme#,00403000	Title = "REMOVE ME..."
00401007	, 68 00304000	push Crackme#,00403000	Text = "NAG"
0040100C	, 6A 00	push 0	hOwner = NULL
0040100E	, E8 18000000	call <jmp,&USER32.MessageBoxA>	MessageBoxA
00401013	, 6A 00	push 0	Style = MB_OK MB_APPLMODAL
00401015	, 68 11304000	push Crackme#,00403011	Title = "Edit,...."
0040101A	, 68 18304000	push Crackme#,0040301B	Text = "Replace my name with yours - CaD"
0040101F	, 6A 00	push 0	hOwner = NULL
00401021	, E8 08000000	call <jmp,&USER32.MessageBoxA>	MessageBoxA
00401026	, 6A 00	push 0	ExitCode = 0
00401028	, E8 07000000	call <jmp,&KERNEL32.ExitProcess>	ExitProcess
0040102D	CC	int 3	
0040102E	\$- FF25 08204000	jmp near dword ptr ds:[<&USER32.MessageBoxA	USER32.MessageBoxA
00401034	,- FF25 00204000	jmp near dword ptr ds:[<&KERNEL32.ExitProcess	kernel32.ExitProcess

[그림7-2. 매우 짧고 간단한 코드]

[그림7-2]에서 보시는 것처럼 코드가 매우 짧고 간결합니다. 제거를 원하는 메시지박스는 메시지박스 호출문을 NOP시키면 간단히 해결될 것입니다.

00401000	\$ 6A 00	push 0	ASCII "REMOVE ME..."
00401002	, 68 00304000	push Crackme#,00403000	ASCII "NAG"
00401007	, 68 00304000	push Crackme#,00403000	
0040100C	, 6A 00	push 0	
0040100E	, 90	nop	
0040100F	, 90	nop	
00401010	, 90	nop	
00401011	, 90	nop	
00401012	, 90	nop	
00401013	, 6A 00	push 0	Style = MB_OK MB_APPLMODAL
00401015	, 68 11304000	push Crackme#,00403011	Title = "Edit,...."
0040101A	, 68 18304000	push Crackme#,0040301B	Text = "Replace my name with yours - CaD"
0040101F	, 6A 00	push 0	hOwner = NULL
00401021	, E8 08000000	call <jmp,&USER32.MessageBoxA>	MessageBoxA
00401026	, 6A 00	push 0	ExitCode = 0
00401028	, E8 07000000	call <jmp,&KERNEL32.ExitProcess>	ExitProcess
0040102D	CC	int 3	
0040102E	\$- FF25 08204000	jmp near dword ptr ds:[<&USER32.MessageBoxA	USER32.MessageBoxA
00401034	,- FF25 00204000	jmp near dword ptr ds:[<&KERNEL32.ExitProcess	kernel32.ExitProcess

[그림7-3. 메시지박스 호출문 NOP시키기]

간단하게 메시지박스 호출문을 없애버렸습니다. 그리고 [그림7-3]의 0040101A의 Text에 ... yours - CaD부분의 CaD를 제 닉네임인 Osiris로 변경하도록 하겠습니다.

Address	Hex dump	ASCII
0040301B	52 65 70 6C 61 63 65 20 6D 79 20 6E 61 6D 65 20	Replace my name
0040302B	77 69 74 68 20 79 6F 75 72 73 20 2D 20 43 61 44	with yours - CaD

[그림7-4. 0040301B에 저장되어 있는 문자들]

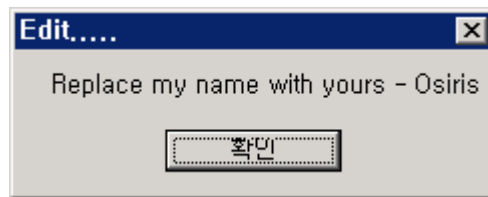
우 클릭 후 Binary -> Edit로 간단히 수정하였습니다.

Address	Hex dump	ASCII
0040301B	52 65 70 6C 61 63 65 20 6D 79 20 6E 61 6D 65 20	Replace my name
0040302B	77 69 74 68 20 79 6F 75 72 73 20 2D 20 4F 73 69	with yours - Osi
0040303B	72 69 73 00 00 00 00 00 00 00 00 00 00 00 00	ris.....

[그림7-5. 수정된 닉네임]

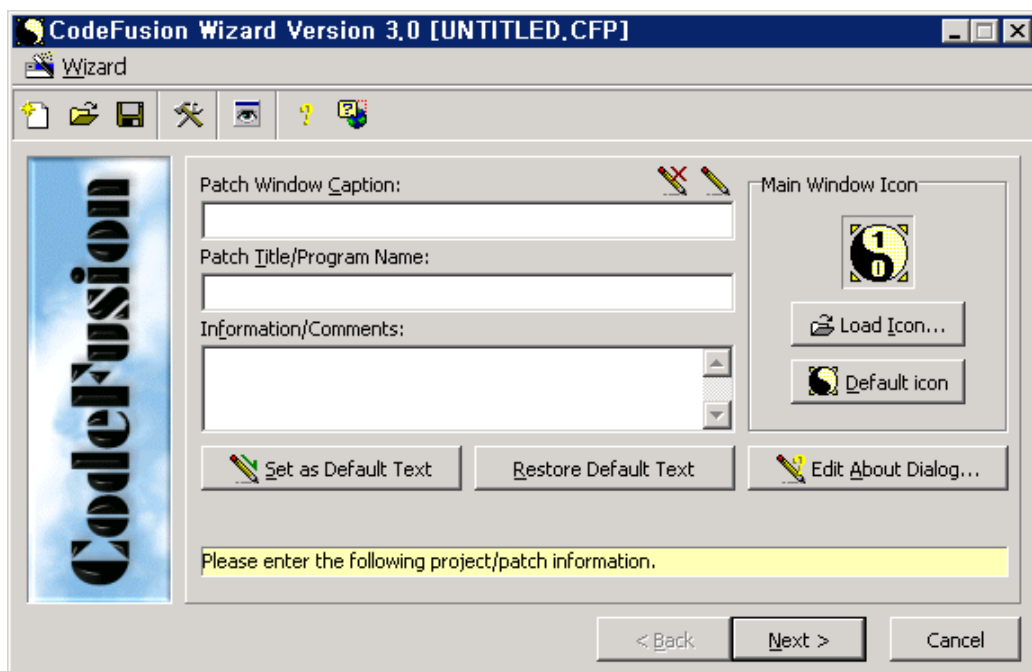
수정이 완료되었으므로 이제 저장을 하여야 합니다. [그림7-5]가 있는 부분에서 마우스 우 클릭을 하게 되면 여러 가지 메뉴가 뜨는 거기서 Copy to executable file이라는 메뉴를 선택합니다. 그러면 새로운 창이 하나 뜨게 됩니다. 거기서 다시 마우스 우 클릭을 하면 Save

file이라는 메뉴를 볼 수 있습니다. Save file을 클릭하신 후 파일명을 정하고 저장을 하면 저장이 완료됩니다. [그림7-6]은 수정 후 실행 했을 때 뜨는 메시지박스 입니다.



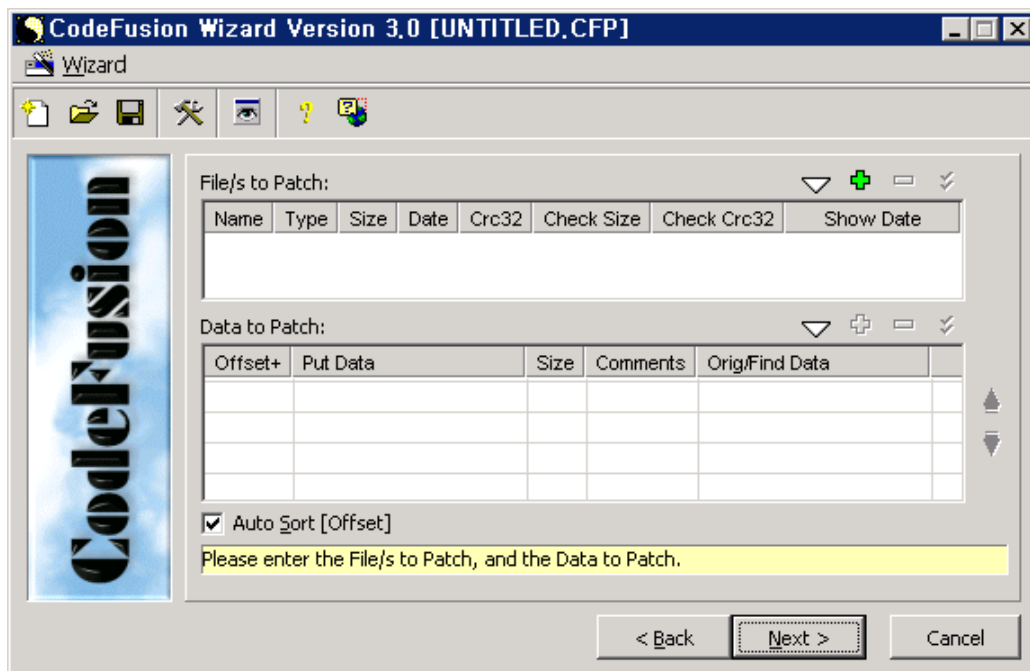
[그림7-6. Replace my name with yours - Osiris]

수정 작업이 모두 완료 되었습니다. 이제 패치파일을 만들어야 합니다. 코드 Fusion이라는 Tool을 이용해서 만들어 보겠습니다.



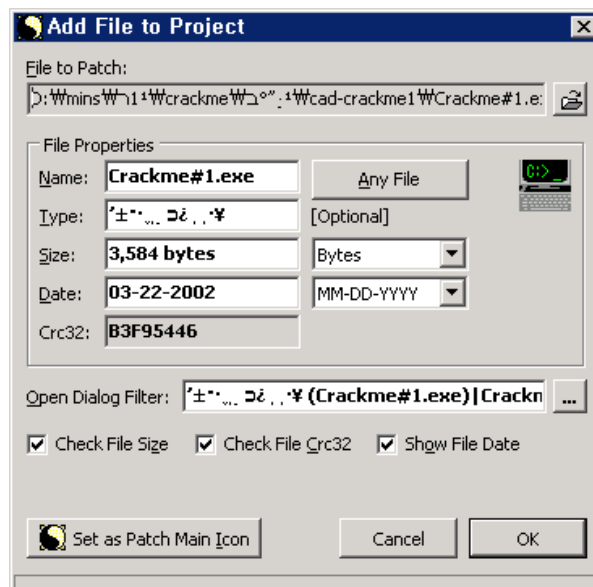
[그림7-7. 코드 Fusion의 실행화면]

사용방법은 매우 간단합니다. 우선 [그림7-7]을 보면 만들 패치파일에 대해서 간단한 정보를 입력하게 됩니다. 간단히 적고 Next버튼을 누르겠습니다.



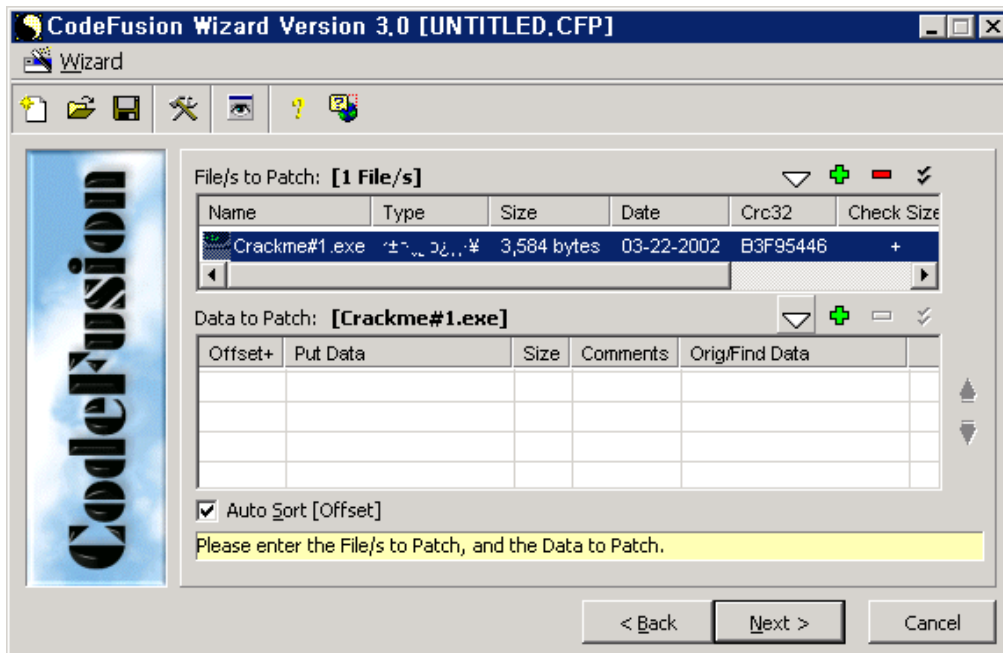
[그림7-8. 파일 추가하는 부분]

[그림7-8]에서 File/s to Patch 에는 패치를 할 원본파일을 불러오면 됩니다. Show Date 위의 하얀색 화살표를 클릭합니다. 그리고 ADD file 을 선택합니다.



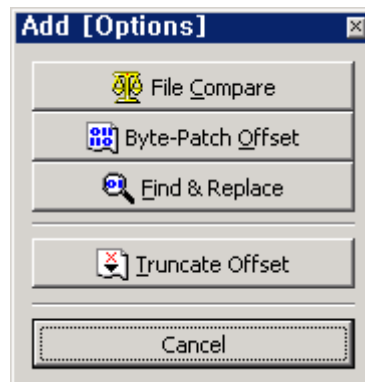
[그림7-9. Add File to Project]

원본파일을 선택합니다. 한글이 다 깨지네요. 그래도 만드는데 문제 없으므로 무시합니다. OK를 눌러 줍니다.



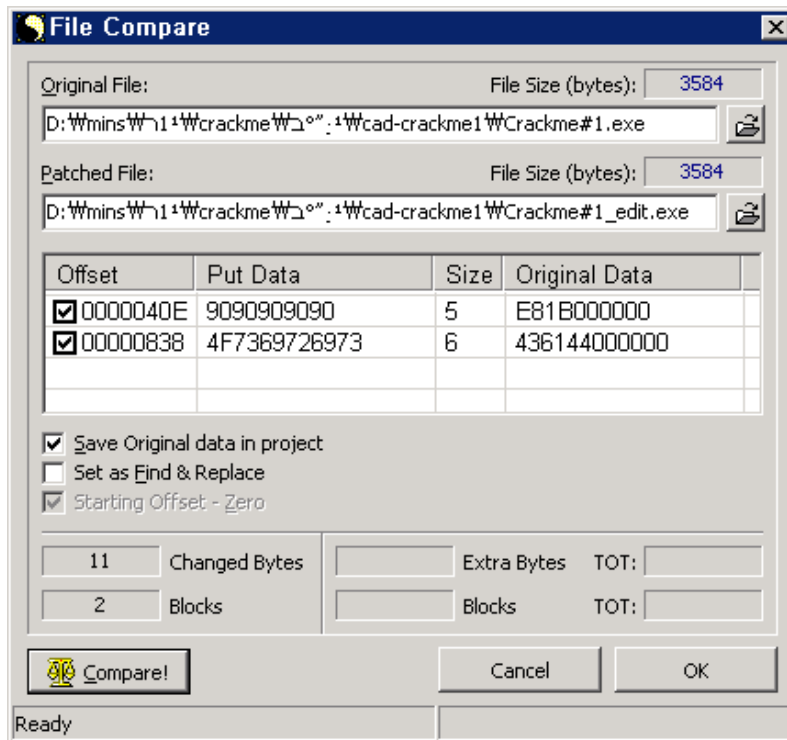
[그림7-10. 파일이 추가된 화면]

이제 Orig/Find Data 위의 하얀색 화살표를 클릭합니다. 그리고 ADD DATA를 선택합니다.



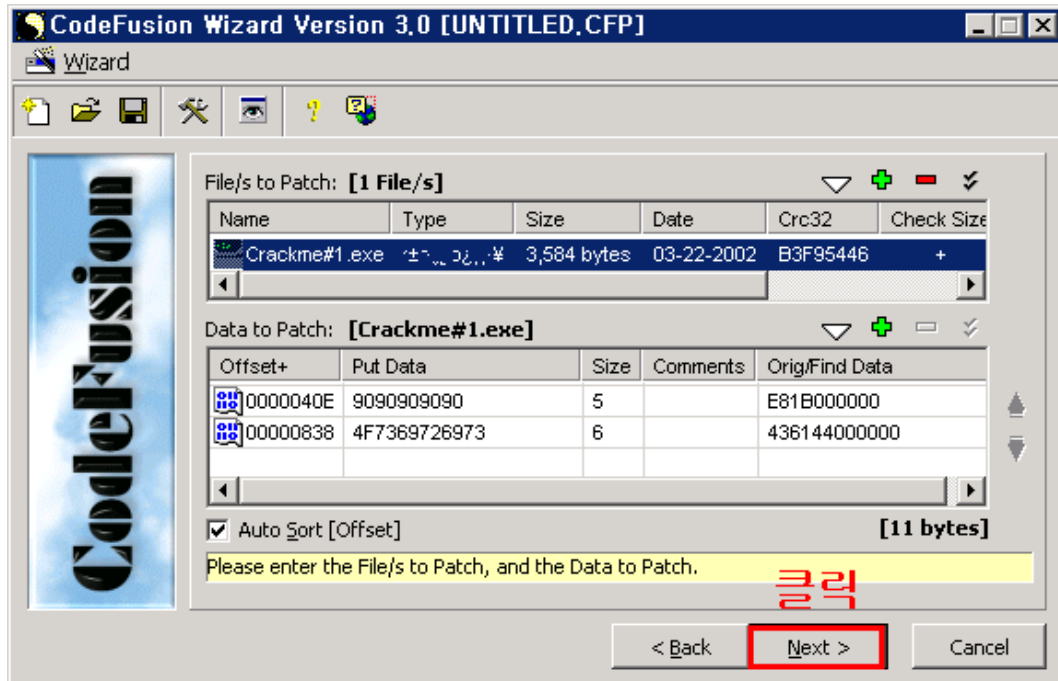
[그림7-11. Add [Options]]

File Compare를 선택합니다. File Compare 외에 다른 메뉴도 있는데 영어로 적힌 그대로의 기능을 가지고 있습니다.



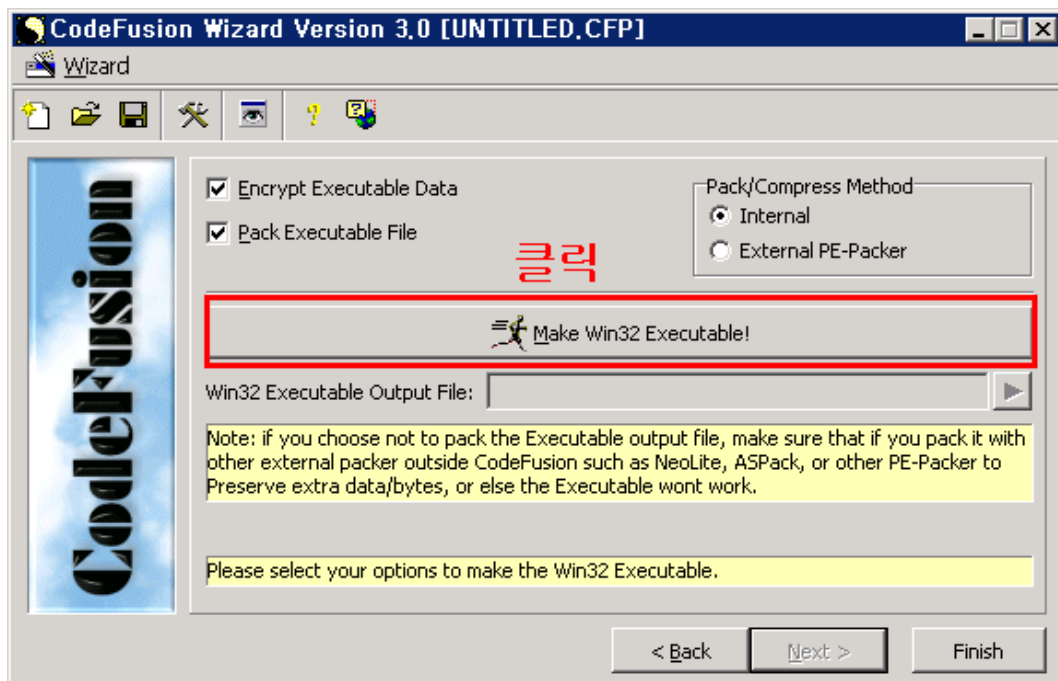
[그림7-12. File Compare]

Patched File 을 찾아서 지정해주고 Compare! 버튼을 누르면 위 [그림7-12]처럼 됩니다. 두 파일을 비교해서 틀린 부분을 찾아내서 보여주게 되는겁니다. 909090 .. NOP 된 부분과 Osiris(0x4F 0x73 ...0x69 0x73)를 확인할 수 있습니다. 이제 OK를 누릅니다.



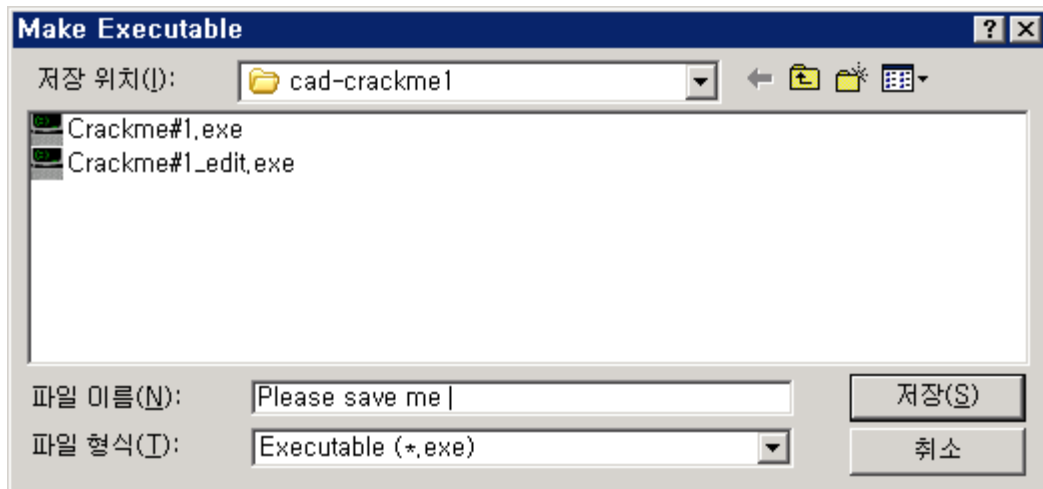
[그림7-13. File Compare후 화면]

Next버튼을 누릅니다.



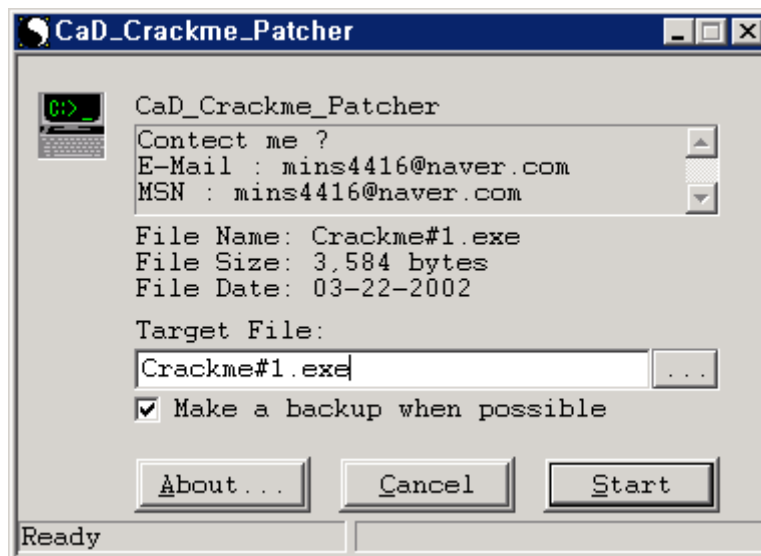
[그림7-14. 패치파일 생성하기]

Make Win32 Executable!을 클릭합니다. 그럼 아래 [그림15]와 같은 창이 뜹니다.



[그림7-15. 파일이름 정하기]

파일이름을 지정해주고 저장하면 Patch 파일이 생성되게 됩니다.



[그림7-16. 생성된 패치파일 실행화면]

완성된 Patch파일을 실행해 보았습니다. Target File을 정해서 Start를 누르면 Patch가 됩니다. Make a backup when possible 옵션을 선택하면 Target File을 Backup하고 Patch를 합니다. 이로써 3가지 목표를 모두 달성하였습니다.

0x0003 결론

툴을 이용해서 간단하게 패치파일을 만들어보았습니다. 코드 Fusion의 Compare기능을 이용해서 패치파일이 어떤 코드를 수정하는지 확인도 할 수 있다는 걸 알 수 있습니다. 주위에서 쉽게 패치파일(원본파일을 덮어 쓰기 하는 식의 패치파일)을 구할 수 있는데, 과연 어떤 코드들을 수정하는지 원본파일과의 Compare를 통해 알아보고 어떻게 만들어진 건지 공부 할 수 있을 것 같습니다.

0x08 Orion Crackme #1

Crackme: http://beist.org/research/public/crackme10/orion_crackme1.zip

Keygen: -

Author: diablo

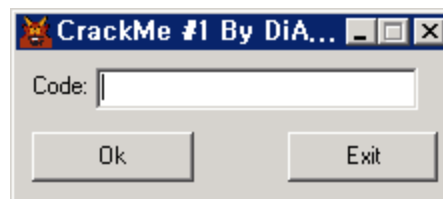
Level: ★

Protection: Serial

0x0001 목표

문자열 검색을 통해 CrackMe 내에 숨겨진 Serial을 찾아야 합니다. 숨겨진 Serial을 입력하고 OK버튼을 누르면 나체의 여자사진이 나타납니다.

0x0002 분석 및 풀이



[그림8-1. Crackme 실행화면]

Visual-Basic으로 만들어진 Crackme입니다. OllyDbg를 이용해서 열어보겠습니다.

R Text strings referenced in CrackMe1;.text		
Address	Disassembly	Text string
00401000	mov eax, dword ptr ds:[4602B4]	(Initial CPU selection)
004012EC	mov edx, CrackMe1.0046D2E8	ASCII "CrackMe #1 By DiABLO"
00401384	ascii "Exception &",0	
004013C0	ascii "AnsiString",0	
00401408	ascii "Exception",0	
004014D8	ascii "TObject",0	
004014F8	ascii "Exception *",0	
004015C0	ascii "TForm1 *",0	
00401600	mov esi, CrackMe1.0046D374	ASCII "****vErYeAsY****"
004016D8	mov ecx, CrackMe1.0046D3A0	ASCII "CrackMe #1 By DiABLO"
004016E0	mov edx, CrackMe1.0046D389	ASCII "Wrong Code! Try Again!"
004016F8	mov ecx, CrackMe1.0046D3D4	ASCII "CrackMe #1 By DiABLO"
004016FD	mov edx, CrackMe1.0046D3B5	ASCII "Right Code! Good Work Cracker!"
00401768	mov ecx, CrackMe1.0046D401	ASCII "CrackMe #1 By DiABLO"
0040176D	mov edx, CrackMe1.0046D3E9	ASCII "U Must Write Something!"

[그림8-2. Text Strings referenced]

OllyDbg로 CrackMe를 열고나서 마우스 우 클릭 후 Search for -> All referenced text string을 선택합니다. 그러면 [그림2]와 같은 창이 새로 생성됩니다. CrackMe의 입력란에 아무 글자나 넣고 OK버튼을 눌러봅니다. 그러면 아래와 같은 메시지박스가 뜹니다.



[그림8-3. Wrong 코드]

잘못된 코드를 넣었으니 다시 입력하라고 합니다. 그런데 [그림8-2]를 보면 [그림3]의 메시지박스가 가지는 문자들이 있는 것을 확인할 수 있습니다. [그림8-2]에서 ASCII "Wrong 코드! Try Again!"라고 적힌 라인을 더블클릭 해봅니다.

004016D7	74 1D	je short CrackMe1.004016F6	같다면 정답으로 분기
004016D9	6A 00	push 0	
004016DB	B9 A0D34600	mov ecx, CrackMe1.0046D3A0	ASCII "CrackMe #1 By DiABLO"
004016E0	BA 89D34600	mov edx, CrackMe1.0046D3B9	ASCII "Wrong Code! Try Again!"
004016E5	A1 501D4700	mov eax, dword ptr ds:[471D50]	
004016EA	8B00	mov eax, dword ptr ds:[eax]	
004016EC	E8 6FA00600	call CrackMe1.0046B760	
004016F1	E9 88000000	jmp CrackMe1.0040177E	
004016F6	6A 00	push 0	
004016F8	B9 D4D34600	mov ecx, CrackMe1.0046D3D4	ASCII "CrackMe #1 By DiABLO"
004016FD	BA B5D34600	mov edx, CrackMe1.0046D3B5	ASCII "Right Code! Good Work Cracker!"

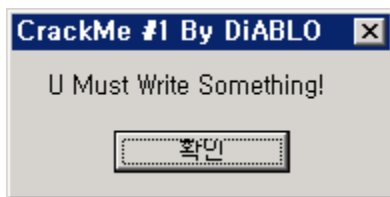
[그림8-4. 진실 혹은 거짓]

더블클릭을 하게 되면 [그림8-4]와 같은 코드가 있는 곳으로 바로 이동하게 됩니다. 더블클릭했던 내용은 004016E0에 있는 것을 확인할 수 있습니다. 그 바로 위인 004016D7을 보면 조건 분기문이 있습니다. 같다면 004016F6으로 분기하게 되는데 아마도 그 분기는 올바른 Serial을 넣었을 때 볼 수 있는 내용인 것 같습니다. 그리고 분기하지 않고 아래로 진행하게 되면 좀 전에 보았던 메시지박스를 보게 됩니다. 즉 004016D7위를 찾아보면 무엇을 비교해서 조건분기를 하는지 알 수 있을 것이고, 어떤 식으로 비교를 하는지 알 수 있을 것입니다.

00401671	0F84 EF000000	je CrackMe1.00401766	아무것도 입력안했으면 분기
00401677	66:C745 04 1400	mov word ptr ss:[ebp-2C], 14	
0040167D	8D45 F4	lea eax, [local.3]	
00401680	E8 0B010000	call CrackMe1.00401790	
00401685	8BD0	mov edx, eax	
00401687	FF45 E0	inc [local.8]	
0040168A	8B4D C0	mov ecx, [local.16]	
0040168D	8B81 C8020000	mov eax, dword ptr ds:[ecx+2C8]	
00401693	E8 80EE0300	call CrackMe1.00440518	
00401698	8D55 F4	lea edx, [local.3]	
0040169B	52	push edx	
0040169C	8D55 A8	lea edx, [local.22]	실제 CODE 값을 가져온다
0040169F	8D45 F0	lea eax, [local.4]	CODE 값과 무언가를 하는듯
004016A2	E8 C9A00600	call CrackMe1.0046B770	
004016A7	FF45 E0	inc [local.8]	
004016AA	8D55 F0	lea edx, [local.4]	
004016AD	58	pop eax	
004016AE	E8 75A10600	call CrackMe1.0046B828	
004016B3	50	push eax	
004016B4	FF4D E0	dec [local.8]	
004016B7	8D45 F0	lea eax, [local.4]	
004016BA	BA 02000000	mov edx, 2	
004016BF	E8 20A10600	call CrackMe1.0046B7E4	
004016C4	FF4D E0	dec [local.8]	
004016C7	8D45 F4	lea eax, [local.3]	
004016CA	BA 02000000	mov edx, 2	
004016CF	E8 10A10600	call CrackMe1.0046B7E4	
004016D4	59	pop ecx	
004016D5	84C9	test cl, cl	CL이 0이면 실제 CODE와 입력한 CODE가 동일
004016D7	74 1D	je short CrackMe1.004016F6	같다면 정답으로 분기

[그림8-5. 이 곳 어딘가에 Serial이 있다]

입력된 내용이 없다면 00401671에서 00401766으로 분기하여 아래와 같은 메시지박스를 보여줍니다.



[그림8-6. 뭐 좀 넣어요]

00401671에서 입력 값의 유무를 확인하고 004016D7에서 정답의 여부를 확인하는 걸로 봐서 00401671과 004016D7사이에 Serial이 있거나 Serial과 관련한 무엇이 있을 것으로 추측이 됩니다. 이 사이의 코드들 중에 값을 가져오고 복사하고 하는 코드에 넉넉히 Breakpoint를 설정하고 진행을 하게 되면 쉽게 Serial같지 않은 Serial을 발견 할 수 있습니다.

0040169C	8D55 A8	lea edx, [local.22]	실제 CODE 값을 가져온다
Stack address=0012F2D4, (ASCII "****vErYeAsY****") edx=0012F320, (ASCII "??")			

[그림8-7. Serial 발견]

스택에 저장되어 있는 문자열을 불러와 EDX Register에 넣는 코드입니다. 그런데 이 문자열이 정말 Serial인지 아니면 낚시인지 확인할 방법은 입력해보는 수밖에 없습니다. 하지만 조금이라도 더 과학적(?)으로 접근하기 위해서 입력한 Serial과 저장된 Serial을 비교하는 부분을 찾겠습니다. 바로 004016AE의 호출문입니다. 004016AE의 호출문만으로 들어가면 다음

과 같은 코드가 나옵니다.

0046B828	\$ 55	push ebp	
0046B829	. 8BEC	mov ebp, esp	
0046B82B	. 53	push ebx	
0046B82C	. 8B00	mov eax, dword ptr ds:[eax]	
0046B82E	. 8B12	mov edx, dword ptr ds:[edx]	
0046B830	. E8 1354FFFF	call CrackMe1.00460C48	<<<<<<<<
0046B835	. 0F95C0	setne al	
0046B838	. 83E0 01	and eax, 1	
0046B83B	. 5B	pop ebx	
0046B83C	. 5D	pop ebp	
0046B83D	. C3	retn	

[그림8-8. 004016AE 호출문]

0046B82C에서 EAX Register에 입력한 Serial값을 가지고 0046B82E에서 EDX Register에 올
바른 Serial이 들어가게 됩니다. [그림8-9]에서 보는 것처럼 말이죠.

0046B828	\$ 55	push ebp	
0046B829	. 8BEC	mov ebp, esp	
0046B82B	. 53	push ebx	
0046B82C	. 8B00	mov eax, dword ptr ds:[eax]	입력한 Serial
0046B82E	. 8B12	mov edx, dword ptr ds:[edx]	
0046B830	. E8 1354FFFF	call CrackMe1.00460C48	<<<<<<<<
0046B835	. 0F95C0	setne al	
0046B838	. 83E0 01	and eax, 1	
0046B83B	. 5B	pop ebx	
0046B83C	. 5D	pop ebp	
0046B83D	. C3	retn	

Registers (FPU)	
EAX	00ED5B9C ASCII "Osiris"
ECX	00000000
EDX	00ED5CA4 ASCII "****vErYeAsY****"
EBX	00ED2BDC ASCII "뽕 A"
ESP	0012F2A4
EBP	0012F2A8
ESI	0046D388 CrackMe1.0046D388
EDI	0012F2CC
EIP	0046B830 CrackMe1.0046B830

[그림8-9. 입력한 Serial과 올바른 Serial]

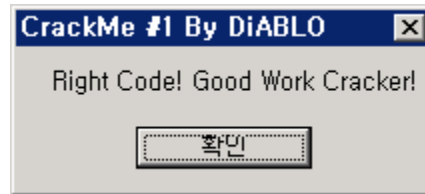
그리고 0046B830의 호출문을 따라 들어가게 되면 [그림10]과 같은 코드를 보게 됩니다.

00460C48	\$ 53	push ebx	
00460C49	. 56	push esi	
00460C4A	. 57	push edi	
00460C4B	. 89C6	mov esi, eax	입력한 CODE를 ESI로 복사
00460C4D	. 89D7	mov edi, edx	실제 CODE를 EDI로 복사
00460C4F	. 39D0	cmp eax, edx	실제 CODE와 입력한 CODE 비교
00460C51	. 0F84 8F000000	je CrackMe1.00460CE6	
00460C57	. 85F6	test esi, esi	
00460C59	. 74 68	je short CrackMe1.00460CC3	
00460C5B	. 85FF	test edi, edi	
00460C5D	. 74 6B	je short CrackMe1.00460CCA	
00460C5F	. 8B46 FC	mov eax, dword ptr ds:[esi-4]	
00460C62	. 8B57 FC	mov edx, dword ptr ds:[edi-4]	
00460C65	. 29D0	sub eax, edx	
00460C67	. 77 02	ja short CrackMe1.00460C6B	
00460C69	. 01C2	add edx, eax	
00460C6B	. 52	push edx	
00460C6C	. C1EA 02	shr edx, 2	

[그림8-10. 00460C4B~00460C4F의 비교부분]

이곳에서 우리는 입력한 Serial과 올바른 Serial을 비교하는 것을 확인할 수 있습니다. 중간중간 생략한 부분이 조금씩 있는데 이번 Crackme같은 경우는 Name을 입력 받아서 어떤 루틴을 거쳐 Serial을 생성하는 것이 아니라 고정되고 숨겨진 Serial을 찾는 것이 목적이기 때문에 적당히 생략하였습니다.

[그림8-7]에서 찾아낸 Serial을 넣고 OK버튼을 눌러보겠습니다.



[그림8-11. Right 코드]

0x0003 결론

워낙 간단한 CrackMe였기 때문에 Search -> All referenced string에서 Serial로 추측되는 문자를 찾아 임의로 대입해 볼 수 있었습니다. 나체의 여성사진이 목적이라면 ResourceHack과 같은 Tool을 이용하면 좋겠습니다.

0x09 CTM-CM #1

Crackme: <http://beist.org/research/public/crackme10/ctm-cm1.zip>

Keygen: <http://beist.org/research/public/crackme10/ctm-cm1-keygen.zip>

Author: cytomic

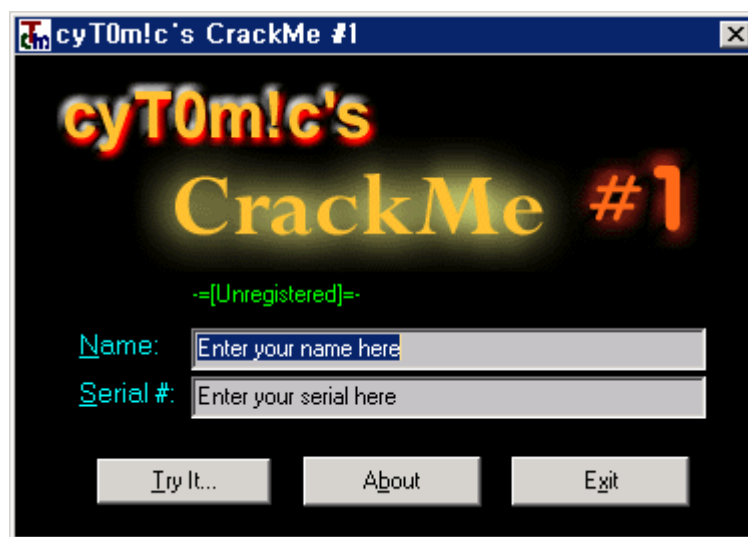
Level: ★

Protection: Name / Serial

0x0001 목표

다른 CrackMe처럼 Name에 맞는 Serial을 구해야 합니다. 하지만 이번 CrackMe는 어셈코드를 직접사용하여 Keygen을 만듭니다.

0x0002 분석 및 풀이



[그림9-1. Crackme실행화면]

Name과 Serial을 입력 할 수 있지만 Serial을 입력하는 부분에는 숫자만 입력할 수 있습니다. 그럼 제 닉네임인 Osiris에 맞는 Serial을 찾아보겠습니다. 일단 OllyDbg를 이용해서 Crackme를 열어보겠습니다.

004251A0	\$	53	push ebx
004251A1	.	89C3	mov ebx, eax
004251A3	.	83FB 00	cmp ebx, 0
004251A6	✓	74 13	je short CrackMe,004251BB
004251A8	.	B8 01000000	mov eax, 1
004251AD	.	31C9	xor ecx, ecx
004251AF	>	8A0B	mov cl, byte ptr ds:[ebx]
004251B1	.	80F9 00	cmp cl, 0
004251B4	✓	74 05	je short CrackMe,004251BB
004251B6	.	F7E1	mul ecx
004251B8	.	43	inc ebx
004251B9	^	EB F4	jmp short CrackMe,004251AF
004251BB	>	25 FFFFFFFF	and eax, 0FFFFFFF
004251C0	.	5B	pop ebx
004251C1	.	C3	retn

[그림9-2. Name을 가지고 Serial을 만드는 부분]

우선 004251A0~004251C1까지 Breakpoint를 설정합니다. 그리고 제 닉네임 Osiris를 Name에 넣고 Serial에는 123456을 넣은 후 Try it버튼을 눌러보겠습니다.

Registers (FPU)	
EAX	00E85D80 ASCII "Osiris"
ECX	00E8620C ASCII "123456"
EDX	0012F9B0
EBX	00E81C34 ASCII "뽀 B"
ESP	0012F98C ASCII "췌 B"
EBP	0012F9B8
ESI	0001E240
EDI	00E8589C ASCII "H1B"
EIP	004251A0 CrackMe,004251A0

[그림9-3. Try it을 누른 후 Registers (FPU)]

[그림9-3]에서 EAX Register에 Name이 그리고 ECX Register에 Serial이 들어간 것을 확인할 수 있습니다.

004251A1 EAX Register의 값을 EBX Register에 복사합니다.

004251A3 EBX Register를 0과 비교합니다.

004251A6 EBX Register와 0이 같다면 004251BB로 분기합니다.

004251A8 EAX Register에 1을 넣습니다.

004251AD ECX Register를 자기자신과 XOR시켜 초기화 시킵니다.

004251AF CL Register에 DS:[EBX]의 값을 복사합니다.

[그림9-4]를 보면 Osiris의 첫 번째 글자인 'O'를 ECX Register에 복사하는걸 알 수 있습니다.

ds:[00E86FC0]=4F ('O')
cl=00
Jump from 004251B9

[그림9-4. CL Register에 DS:[EBX]의 값을 복사]

004251B1 CL Register를 0과 비교합니다.

004251B4 CL Register와 0이 같다면 004251BB로 분기합니다.

004251B6 EAX = EAX * ECX를 합니다.

EAX Register의 값이 004251A8에서 1이 된 이후로 변화가 없었으므로 연산을 하게 되면 EAX Register에 들어가는 값은 4F가 됩니다.

004251B8 EBX Register를 1증가시킵니다.

004251B9 004251AF로 분기합니다.

이렇게 순차적으로 진행이 됩니다. 입력한 Name인 Osiris가 위의 연산을 모두 마치게 되면 EAX Register에는 BC60633E라는 값이 들어가게 됩니다.

O = 0x4F, s = 0x73, i = 0x69, r = 0x72, i = 0x69, s = 0x73 인데 이것으로 연산을 하면 $0x4F * 0x73 * 0x69 * 0x72 * 0x69 * 0x73 = 0x131BC60633E$ 가 됩니다. 하지만 EAX Register에서 표현 가능한 자리 수 때문에 앞부분(0x13100000000)은 소멸됩니다. 이렇게 입력한 Name이 모두 연산되고 난 후 004251B8에서 EBX Register를 1증가 시키고 004251B9에서 004251AF로 분기합니다.

004251B8에서 EBX Register가 1증가 되었을 때 ds:[EBX]의 값은 0이 됩니다. 따라서 다음 코드에서 CL Register에 0이 들어가게 되고 조건 분기문에서 조건을 만족하게 되므로 004251B4에서 004251BB로 분기하게 됩니다.

004251BB EAX = EAX & 0x0FFFFFFF을 하게 됩니다. Osiris를 Name값으로 가지고 연산이 끝났을 때 EAX Register의 값은 0xBC60633E입니다. 이 값을 0x0FFFFFFF과 AND연산을 시키게 되는데 AND연산 후 EAX Register의 값은 0x0C60633E가 됩니다.

00425088	, E8 13010000	call CrackMe,004251A0	Serial 만들기
0042508D	, 8BF8	mov edi, eax	
0042508F	, 3BFE	cmp edi, esi	
00425091	, 74 18	je short CrackMe,004250AB	
00425093	, 6A 00	push 0	
00425095	, B9 20514200	mov ecx, CrackMe,00425120	ASCII "cyT0m!c's CrackMe #1"
0042509A	, BA 38514200	mov edx, CrackMe,00425138	ASCII "That isn't it, keep on trying..."
0042509F	, A1 28764200	mov eax, dword ptr ds:[427628]	
004250A4	, E8 23CAFFFF	call CrackMe,00421ACC	
004250A9	, EB 16	jmp short CrackMe,004250C1	
004250AB	, 6A 00	push 0	
004250AD	, B9 20514200	mov ecx, CrackMe,00425120	ASCII "cyT0m!c's CrackMe #1"
004250B2	, BA 5C514200	mov edx, CrackMe,0042515C	ASCII "Hey, you have done it"
004250B7	, A1 28764200	mov eax, dword ptr ds:[427628]	
004250BC	, E8 0BCAFFFF	call CrackMe,00421ACC	
004250C1	, 3BFE	cmp edi, esi	
004250C3	, 75 26	jnz short CrackMe,004250EB	
004250C5	, BA 7C514200	mov edx, CrackMe,0042517C	ASCII "Registered to "

[그림9-5. 성공 or 실패]

[그림9-5]에서 보는 것처럼 Name으로 모든 연산이 끝난 후 에 호출문을 빠져 나오게 되는데 그 후 바로 비교 문을 통해서 성공메시지 혹은 실패메시지를 보게 됩니다.

0042508D EDI Register에 EAX Register의 값을 복사합니다.

0042508F EDI Register와 ESI Register의 값을 비교합니다.

00425091 EDI Register와 ESI Register의 값이 같다면 004250AB(성공)로 분기합니다.

만약 같지 않다면 분기하지 않으므로 실패메시지로 진행됩니다.

Name : Osiris

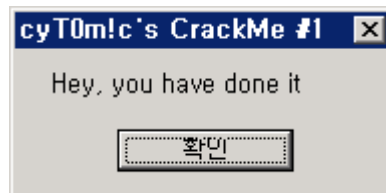
Serial : C60633E

하지만 Serial에는 문자를 사용할 수 없습니다. 따라서 10진수로 변경을 시켜줘야 합니다.

Name : Osiris

Serial : 207643454

올바른 Serial인지 확인해 보겠습니다.



[그림9-6. 성공메시지]

성공하였습니다.

0x0003 결론

그다지 어렵지 않게 분석을 하고 풀이를 마쳤습니다. 이제 Keygen을 만들도록 하겠습니다.

```
#include "stdafx.h"
#include <windows.h>

int _tmain(int argc, _TCHAR* argv[])
{
    char Name[20];
    __int64 serial;
Nameinput:
    printf("Name : ");
    gets(Name);
    if(strlen(Name) > 20)
    {
        printf("20자리 이하로 입력하세요.\n");
        goto Nameinput;
    }
    __asm
    {
        pushad
        //초기화
        XOR ESI,ESI
        XOR EDX,EDX
        XOR EBX,EBX
        XOR ECX,ECX
        MOV EAX,1

serial_loop:
        MOV CL,BYTE PTR DS:[EBX+Name]
        CMP CL,0
        JE serial_cmd
        MUL ECX
        INC EBX
        JMP serial_looP

serial_cmd:
        AND EAX,0x0FFFFFFF
        MOV DWORD PTR DS:[ECX+serial],EAX
        popad
    }
    printf("Serial : %d\n\n", serial);
    system("pause");
    return 0;
}
```

0x0a Bengaly Crackme #3

Crackme: <http://beist.org/research/public/crackme10/bengaly-km3.zip>

Keygen: <http://beist.org/research/public/crackme10/Key4-keygen.zip>

Author: bengaly

Level: ★

Protection: Name / Serial

0x0001 목표

앞에서 다뤘던 CrackMe들과 비슷한 Name에 맞는 Serial을 구하는 CrackMe입니다. 분석을 통해 입력된 Name으로 Serial을 만드는 부분을 찾아서 keygen을 만들어 봅시다.

0x0002 분석 및 풀이



[그림10-1. Crackme실행 화면]

입력을 할 수 있는 공간이 2개 있습니다. 위가 Name이고 아래가 Serial입니다. 제 닉네임인 Osiris에 맞는 Serial을 구해보도록 하겠습니다. 일단 OllyDbg를 이용해서 열어보겠습니다.

004012B1	6A 40	push 40	Count = 40 (64,)
004012B3	68 3F304000	push Key4,0040303F	Buffer = Key4,0040303F
004012B8	6A 6A	push 6A	ControlID = 6A (106,)
004012BA	FF75 08	push [arg.1]	hWnd
004012BD	E8 0C010000	call <jmp,&USER32.GetDlgItemTextA>	GetDlgItemTextA
004012C2	83F8 00	cmp eax, 0	
004012C5	74 18	je short Key4,004012DF	
004012C7	6A 40	push 40	Count = 40 (64,)
004012C9	68 3F314000	push Key4,0040313F	Buffer = Key4,0040313F
004012CE	6A 6B	push 6B	ControlID = 6B (107,)
004012D0	FF75 08	push [arg.1]	hWnd
004012D3	E8 F6000000	call <jmp,&USER32.GetDlgItemTextA>	GetDlgItemTextA
004012D8	83F8 00	cmp eax, 0	
004012DB	74 02	je short Key4,004012DF	
004012DD	EB 17	jmp short Key4,004012F6	
004012DF	6A 00	push 0	Style = MB_OK MB_APPLMODAL
004012E1	68 8C344000	push Key4,0040348C	Title = "KeygenMe #3"
004012E6	68 00304000	push Key4,00403000	Text = " Please Fill In 1 Char to Continue!!"
004012EB	6A 00	push 0	hOwner = NULL
004012ED	E8 00010000	call <jmp,&USER32.MessageBoxA>	MessageBoxA

[그림10-2. GetDlgItemText API와 메시지박스 API]

Name과 Serial을 입력을 받게 되는데 004012C2와 004012D8에서 EAX Register의 값을 0과 비교하여 입력여부를 확인합니다. 만약 EAX Register의 값이 0과 같다면 004012DF로 분기하게 됩니다. 004012DF는 [그림10-2]에서 확인할 수 있습니다. Name과 Serial모두 1글자 이상 입력이 되었다면 004012DD에서 Serial을 만드는 부분인 004012F6으로 분기하게 됩니다.

004012F6	> 68 3F304000	push Key4,0040303F	[String = "Osiris"
004012F6	, E8 34010000	call <jmp,&KERNEL32.lstrlenA>	lstrlenA
00401300	, 33F6	xor esi, esi	
00401302	, 33DB	xor ebx, ebx	
00401304	, 8BC8	mov ecx, eax	
00401306	, B8 01000000	mov eax, 1	
0040130B	> 8B1D 3F304000	mov ebx, dword ptr ds:[40303F]	
00401311	, 0FB901 1F3540	movsx edx, byte ptr ds:[eax+40351F]	
00401318	, 2BDA	sub ebx, edx	ebx = ebx + edx
0040131A	, 0FAFDA	imul ebx, edx	
0040131D	, 8BF3	mov esi, ebx	
0040131F	, 2BD8	sub ebx, eax	
00401321	, 81C3 4335350	add ebx, 4353543	
00401327	, 03F3	add esi, ebx	
00401329	, 33F2	xor esi, edx	
0040132B	, B8 04000000	mov eax, 4	
00401330	, 49	dec ecx	
00401331	, ^ 75 D8	jnz short Key4,0040130B	
00401333	, 56	push esi	ASCII "aaaaaa"
00401334	, 68 3F314000	push Key4,0040313F	
00401336	, E8 4A000000	call Key4,00401368	
0040133E	, 5E	pop esi	
0040133F	, 3BC6	cmp eax, esi	
00401341	, ^ 75 15	jnz short Key4,00401358	
00401343	, 6A 00	push 0	Style = MB_OK MB_APPLMODAL
00401345	, 68 8C344000	push Key4,0040348C	Title = "KeygenMe #3"
0040134A	, 68 DD344000	push Key4,004034D0	Text = " Great, You are ranked as Level-3 at Keygening now"
0040134F	, 6A 00	push 0	hOwner = NULL
00401351	, E8 9C000000	call <jmp,&USER32.MessageBoxA>	MessageBoxA
00401356	, ^ EB 13	jmp short Key4,0040136B	
00401358	, > 6A 00	push 0	Style = MB_OK MB_APPLMODAL
0040135A	, 68 8C344000	push Key4,0040348C	Title = "KeygenMe #3"
0040135F	, 68 AA344000	push Key4,004034AA	Text = " You Have Entered A Wrong Serial, Please Try Again"
00401364	, 6A 00	push 0	hOwner = NULL
00401366	, E8 87000000	call <jmp,&USER32.MessageBoxA>	MessageBoxA

[그림10-3. 수 많은 코드들]

004012F6을 보면 0040303F의 값을 PUSH하는 것을 볼 수 있는데 0040303F를 확인해보면 입력한 닉네임인 Osiris가 들어있는 것을 알 수 있습니다. 그리고 그 근처를 보면 입력했던 Serial도 확인할 수 있습니다. Name은 0040303F에 있고 Serial은 0040313F에 있습니다.

Address	Hex dump	ASCII
0040303F	4F 73 69 72 69 73 00 00 00 00 00 00 00 00 00 00	Osiris
0040304F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040305F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040306F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040307F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040308F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040309F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030AF	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030BF	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030CF	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030DF	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030EF	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030FF	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040310F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040311F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040312F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040313F	61 61 61 61 61 61 00 00 00 00 00 00 00 00 00 00	aaaaaa

[그림10-4. Name과 Serial]

004012FB에서 strlen을 호출하여 Name의 길이를 확인합니다. 확인된 길이는 EAX Register에 저장됩니다. 00401300과 00401302에서 ESI와 EDX Register를 자기 자신들과 XOR 연산을 하여 초기화 시킵니다. 그리고 00401304에서 EAX Register의 값을 ECX Register로 복사합니다.

00401306 EAX Register에 1을 복사합니다.

0040130B EBX Register에 dword ptr ds:[40303F]의 값을 복사합니다.

ds:[0040303F]=7269734F
Jump from 00401331

[그림10-5. ds:[0040303F]]

[그림10-5]처럼 EBX Register에 7269734F가 들어가게 됩니다. 0x72 = 'r' 0x69 = 'i' 0x73 = 's' 0x4F = 'O'입니다. 입력한 Name인 Osiris모두가 들어가지 않고 뒷부분이 잘려서 EBX Register에 들어가는걸 알 수 있습니다.

Address	Hex dump	ASCII
0040351F	20 25 40 24 65 72 77 72 23 40 24 24 21 40 23 32	%@\$erwr#@\$\$!@#2
0040352F	31 24 40 5E 26 2A 26 28 25 72 74 68 64 68 64 66	1\$@^&*%(&rtthdhdf
0040353F	77 34 32 33 25 23 44 53 67 66 59 24 25 5E 23 24	w423%#DSgfY\$%^#\$
0040354F	25 62 72 65 23 42 40 40 25 23 47 33 72 65 00 00	%bre#B@@%#G3re,,

[그림10-6. 0040351F]

00401311 EDX Register에 byte ptr ds:[eax+40351F]의 값을 넣습니다. 처음 이 코드가 실행 될 때 EAX Register의 값은 1입니다. 따라서 byte prt ds:[1+40351F]의 값이 %(0x25)인 것을 [그림10-6]에서 확인할 수 있습니다.

00401318 EBX = EBX - EDX를 합니다. EBX(0x7269734F) - EDX(0x25)를 하면 0x7269732A가 됩니다.

0040131A EBX = EBX * EDX를 합니다. EBX(0x7269732A) * EDX(0x25)를 하면 0x10893DA512가 됩니다. 그런데 32bit Register에서 표현 가능한 자리 수 때문에 앞의 2자리가 소멸됩니다. 따라서 EBX Register는 0x893D4512가 됩니다.

0040131D EBX Register의 값을 ESI Register로 복사합니다.

0040131F EBX = EBX - EAX를 합니다.

EBX(0x893DA512) - EAX(0x01)을 하면 0x893DA511이 됩니다.

00401321 EBX Register에 0x04353543를 더합니다.

그러면 EBX Register의 값은 0x8D72DA54가 됩니다.

00401327 ESI = ESI + EBX를 합니다.

계산을 하면 ESI는 0x116B01F66이 되는데 자리 수 때문에 앞의 1자리가 소멸됩니다.

따라서 ESI Register의 값은 0x16B01F66이 됩니다.

00401329 ESI = ESI ^ EDX를 합니다. 그러면 ESI는 0x16B01F43이 됩니다.

0040132B EAX Register에 0x04를 복사합니다.

00401330 ECX Register의 값에서 0x01을 뺍니다.

이런 식으로 입력한 Name의 문자들 하나하나를 계산해서 ESI Register에 값을 넣습니다.

0040133F EAX Register의 값과 ESI Register의 값을 비교합니다.

00401341 0040133F의 비교 결과에 따라 00401358로 분기하게 됩니다.

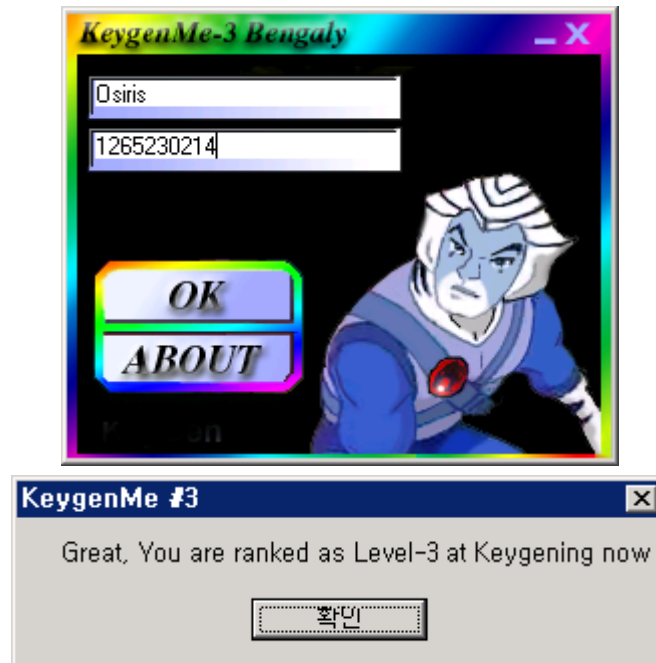
0040133F에서 EAX Register의 값은 입력한 Serial이 16진수로 변환되어 저장된 것이고, 비교하게 되는 ESI Register의 값은 입력한 Name으로 만들어진 값입니다. 이 값이 바로 Serial입니다. Name에 Osiris를 Serial에 255를 넣어서 ESI Register의 값과 EAX Register의 값을 확인해 보겠습니다.

```
esi=4B69E186
eax=000000FF
```

[그림10-7. 0040133F 코드에서의 ESI와 EAX Register의 값]

[그림10-7]을 통해서 입력한 Serial이 확실히 16진수화 되어 EAX Register에 들어가는 것을 확인하였습니다. 따라서 우리는 Serial을 얻기 위해 Name으로 만들어져 ESI Register에 들어 있는 4B69E186의 값을 10진수화 시키면 됩니다.

0x0003 결론



[그림10-8. 성공메시지]

이제 Name을 이용해서 Serial을 생성하는 Keygen을 만들도록 하겠습니다.

```
#include "stdafx.h"
#include <windows.h>

int _tmain(int argc, _TCHAR* argv[])
{
    char name[20];
    char
key코드[] = " %@$erwr#@$!@#21$@^&*&(%rthdhdfw423%#DSgfy$%^#$$%bre#B@@@%#G3
re ";
    int namelength;
    unsigned long serial;
    __int64 serial_1;

    nameinput:
    printf("Name : ");
    gets(name);
    if(strlen(name) > 20)
    {
        printf("20자리 이하로 입력해주세요\n");
        goto nameinput;
    }
    namelength = strlen(name);

    __asm
    {
        pushad
        //초기화
        MOV EAX, namelength
        XOR ESI, ESI
```



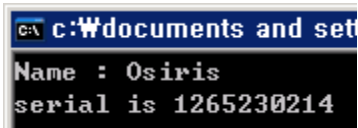
```

XOR EDX, EDX
XOR EBX, EBX
MOV ECX, EAX
MOV EAX, 1

//시리얼 생성 Loop
bigloop:
MOV EBX,DWORD PTR DS:[name]
MOVSX EDX,BYTE PTR DS:[EAX+key코드]
SUB EBX,EDX
IMUL EBX,EDX
MOV ESI,EBX
SUB EBX,EAX
ADD EBX,0x4353543
ADD ESI,EBX
XOR ESI,EDX
MOV EAX,4
DEC ECX
JNZ SHORT bigloop
MOV serial, ESI
ending:
popad
}
serial_1 = serial;
printf("serial is %I64d\n", serial_1);
printf("\n\n\n");
system("pause");

return 0;
}

```



```

C:\> c:\documents and settings\Osiris\...
Name : Osiris
serial is 1265230214

```

[그림10-8. Keygen]

0x0b. 참고사이트 & 참고문헌

- **learn2crack** - <http://learn2crack.com/>

초보자를 위한 크랙과 리버스 엔지니어링 참고 자료가 많은 사이트

- **Windows API 연구사이트** - <http://www.winapi.co.kr>

윈도우 API에 관한 많은 내용이 있음

- **codeDiver님의 홈페이지** - <http://web.kaist.ac.kr/~taekwonv/>

Reverse Engineering에 대한 기초적이고 전반적이 내용이 있음

- **검이님의 블로그** - <http://gum2.tistory.com/>

리버싱과 크랙에 관해서 다루고 있음

- **OllyDbg** - <http://www.ollydbg.de>

본 문서에서 사용하는 디버거 공식사이트

- **MSDN**

MS에서 제공하는 엄청난 양의 문서