

CodeEngn Advanced RCE L08

L3m0nTr33
L3m0nTr33.sur3x5f.org

1. Introduction

Advance RCE L08 문제는 **Name 값과 Key 값을 입력 받아 검증**하는 Program 이다. Packing 이 되지 않았으므로, 별도의 Unpacking 과정 필요 없이 바로 OllyDbg 로 Debugging 이 가능하다. Debugging 을 수행하는 결과 **Name 값 길이 (3 ~ 30)** 를 검증 하는 **Code 와 Name 값에 의한 Key 값을 생성하는 함수**를 호출하는 곳을 발견할 수 있었다. (여기서 **Serial** 생성은 Name 값의 길이가 범위에 속하지 않더라도 생성한다.)

Address	Hex dump	Disassembly	Comment
0045B850	55	PUSH EBP	
0045B851	8BEC	MOV EBP,ESP	
0045B853	B9 07000000	MOV ECX,7	
0045B858	6A 00	PUSH 0	
0045B85A	6A 00	PUSH 0	
0045B85C	49	DEC ECX	
0045B85D	75 F9	JNZ SHORT Advance_.0045B858	
0045B85F	51	PUSH ECX	
0045B860	53	PUSH EBX	
0045B861	56	PUSH ESI	
0045B862	57	PUSH EDI	
0045B863	8955 F4	MOV DWORD PTR SS:[EBP-C],EDX	
0045B866	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	
0045B869	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]	
0045B86C	E8 2394FAFF	CALL Advance_.00404C94	
0045B871	33C0	XOR EAX,EAX	
0045B873	55	PUSH EBP	
0045B874	68 ABA4500	PUSH Advance_.0045BAAA	
0045B879	64:FF30	PUSH DWORD PTR FS:[EAX]	
0045B87C	64:8920	MOV DWORD PTR FS:[EAX],ESP	
0045B87F	33D2	XOR EDX,EDX	
0045B881	33F6	XOR ESI,ESI	
0045B883	33C0	XOR EAX,EAX	
0045B885	8945 F0	MOV DWORD PTR SS:[EBP-10],EAX	

- 해당 Code 를 살펴보면 5개의 **Generate Routine** 으로 이루어져 있으며, Generate Routine 을 통해 생성된 값을 4-4-8-4-4 씩 나누는 것을 확인할 수 있다.
- 이를 통해 **Key 값은 4-4-8-4-4**의 구조를 가져야 한다는 것을 확인 가능하다.

Address	Value	Comment
0012F63C	00A8C7F0	ASCII "BBD40E29"
0012F640	00A8C9B8	ASCII "BBD4"
0012F644	00A8C430	ASCII "8EA9292"
0012F648	00A8C940	ASCII "8EA9"
0012F64C	00A8C7A8	ASCII "FE90F705"
0012F650	00A8C748	ASCII "90518E8"
0012F654	00A8C760	ASCII "9051"
0012F658	00A8C700	ASCII "A672E340"
0012F65C	00A8C718	ASCII "A672"

2. Analyze

1'st Generate Routine :

Address	Hex dump	Disassembly	Comment
0045B898	B9 01000000	MOV ECX,1	
0045B89D	8B5D FC	MOV EBX,DWORD PTR SS:[EBP-4]	1'st Generate Routine
0045B8A0	0FB6740B FF	MOVBX EBX, BYTE PTR DS:[EBX+ECX-1]	
0045B8A5	03F2	ADD ESI,EDX	
0045B8A7	69F6 72070000	IMUL ESI,ESI,772	
0045B8AD	8BD6	MOV EDX,ESI	
0045B8AF	0FAFD6	IMUL EDX,ESI	
0045B8B2	03F2	ADD ESI,EDX	
0045B8B4	0BF6	OR ESI,ESI	
0045B8B6	69F6 74040000	IMUL ESI,ESI,474	
0045B8BC	03F6	ADD ESI,ESI	
0045B8BE	8BD6	MOV EDX,ESI	
0045B8C0	41	INC ECX	
0045B8C1	48	DEC EAX	
0045B8C2	75 D9	JNZ SHORT Advance_.0045B89D	

- 분석
 - 우선 ECX 에 1 을 넣고, ESI 에 Name 을 1글자 씩 넣는다.
 - EDX (첫번째 Loop 에는 0) 를 ESI 에 넣고, ESI * ESI * 0x772 를 수행한다.
 - ESI 값을 EDX 에 복사한 후, 두 값을 부호있는 연산으로 곱셈한다. (ESI 제곱과 같은 역할)
 - 해당 값을 ESI 에 더한 후, OR ESI ESI (쓸데 없는 연산) 을 한다.
 - ESI * ESI * 474 를 수행 한 후, ESI + ESI 를 수행한다.
 - ESI 값을 EDX 에 복사한 후, ECX 값을 1 증가, EAX 값을 1 감소 시킨다.
 - EAX 값이 0 이 아니면 다시 해당 Loop 를 수행한다.

- 해당 Routine 을 수행하면, EDX 에 결과 값이 들어 있다. (이는 1234 라는 Name 에 대한 첫번째 Key Block 값과 관련있다.)

Registers (FPU)	
EAX	00000000
ECX	00000005
EDX	0672E340
EBX	00A8C6E8 ASCII "1234"
ESP	0012F624
EBP	0012F678
ESI	A672E340
EDI	0012FB98

2'nd Generate Routine :

Address	Hex dump	Disassembly	Comment
004588D5	> 8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]	2'nd Generate Routine
004588D8	. 0FB65402 FF	MOVBZ EDX,BYTE PTR DS:[EDX+EAX-1]	
004588DD	. 83C2 11	ADD EDX,11	
004588E0	. 83EA 05	SUB EDX,5	
004588E3	. 69D2 92000000	IMUL EDX,EDX,92	
004588E9	. 83D2	ADD EDX,EDX	
004588EB	. 69D2 19080000	IMUL EDX,EDX,819	
004588F1	. 0155 F0	ADD DWORD PTR SS:[EBP-10],EDX	
004588F4	. 48	DEC EAX	
004588F5	. 85C0	TEST EAX,EAX	
004588F7	. ^75 DC	JNZ SHORT Advance_.004588D5	

- 분석

EDX 에 Name 을 1글자씩 값을 넣은 후, 0x11 을 더하고 0x5 를 뺀다.

EDX * EDX * 0x92 를 수행한 후, EDX + EDX 를 수행한다.

EDX * EDX * 0x819 를 수행한 후, EBP-10 주소에 해당 값을 더한다.

EAX 를 1 감소 시킨다.

EAX 값이 0 이 아니면 다시 해당 Loop 를 수행한다.

- 해당 Routine 을 수행하면, EBP - 10 자리에 결과 값이 들어 있다. (이는 1234 라는 Name 에 대한 두번째 Key Block 값과 관련있다.)

Address	Value	Comment
EBP-10	090518E8	
EBP-C	0012F698	
EBP-8	00000000	
EBP-4	00A8C6E8 ASCII "1234"	
EBP ==>	0012F6AC	
EBP+4	004588A0	RETURN to Advance_.004588A0 from Advance_.004588
EBP+8	00A8C6D0 ASCII "5678"	
EBP+C	0012FD40	Pointer to next SEH record
EBP+10	00458C02	SE handler

3'rd Generation Function :

해당 Function 은 Routine 이 아닌, 함수를 호출하여 값을 도출하게 되는데 이는 MD5 Algorithm 이다.

- PEiD -> Krypto ANALyzer Plugin 을 이용하여 MD5 Algorithm 이 있다는 것을 확인 할 수 있으며, 이 Program 에서는 Name + 정해진 문자열 값을 인

자로 받아서 MD5 Hash 값을 생성한다.

004588FC	. B9 C0BA4500	MOV ECX,Advance_.00458AC0	ASCII "w09/720(="=!)&")?""(=?"
00458901	. 8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]	
00458904	. E8 2392FAFF	CALL Advance_.00404B2C	
00458909	. 8B45 E8	MOV EAX,DWORD PTR SS:[EBP-10]	
0045890C	. 8D55 F8	LEA EDX,DWORD PTR SS:[EBP-8]	
0045890F	. E8 38FCFFFF	CALL Advance_.0045B54C	3'rd Generate Function
Stack SS:[0012F660]=00A8E08. (ASCII "1234w09/720(="=!)&")?""(=?")			
EAX=0045BAC0 (Advance_.0045BAC0), ASCII "w097/720(="=!)&")?""(=?"			

- 함수 내부로 들어가 보면, 어느정도의 Routine 을 수행하는 곳을 찾을 수 있다.

Address	Hex dump	Disassembly	Comment
00458500	. 8D75 EC	LEA ESI,DWORD PTR SS:[EBP-14]	
00458503	> 8D40 90	LEA ECX,DWORD PTR SS:[EBP-70]	
00458506	. 0FB606	MOVBZ EAX,BYTE PTR DS:[ESI]	
00458509	. BA 02000000	MOV EDX,2	
0045850E	. E8 B5D1FAFF	CALL Advance_.004086C8	
00458513	. 8B55 90	MOV EDX,DWORD PTR SS:[EBP-70]	
00458516	. 8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]	
00458519	. E8 C295FAFF	CALL Advance_.00404AE0	
0045851E	. 8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]	
00458521	. 46	INC ESI	
00458522	. 48	DEC EBX	
00458523	. ^75 DE	JNZ SHORT Advance_.00458503	

- 우선 해당 MD5-Algorithm 이 옳은 것인지를 확인하기 위해, 간단히 Hash 계산 Program 을 수행하여 값을 비교하여 보았으나, 값이 일치하지 않았다. 이는 일 반적으로 사용하는 MD5와 Load magis initialization Constants 가 다르기 때문이라는 것을 확인 할 수 있었다.

Address	Hex dump	Disassembly	Comment
0045A860	\$ 53	PUSH EBX	
0045A861	. 8BD8	MOV EBX,EAX	
0045A863	. 8BC3	MOV EAX,EBX	
0045A865	. 33C9	XOR ECX,ECX	
0045A867	. BA 58000000	MOV EDX,58	
0045A86C	. E8 C88FAFF	CALL Advance_.0040333C	
0045A871	. C703 077D55A3	MOV DWORD PTR DS:[EBX],A3557D07	origin : 67452301
0045A877	. C743 04 D312F	MOV DWORD PTR DS:[EBX+4],62FB12D3	origin : EFC0AB89
0045A87E	. C743 08 F645D	MOV DWORD PTR DS:[EBX+8],EFD945F6	origin : 98BADCFC
0045A885	. C743 0C 9EE27	MOV DWORD PTR DS:[EBX+C],E57AE29E	origin : 10325476
0045A88C	. 5B	POP EBX	
0045A88D	. C3	RETN	

4'th Generation Routine :

Address	Hex dump	Disassembly	Comment
0045B914	. 33FF	XOR EDI,EDI	
0045B916	. 33C0	XOR EAX,EAX	
0045B918	. 8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]	
0045B91B	. 85D2	TEST EDX,EDX	
0045B91D	~ 74 05	JE SHORT Advance_.0045B924	
0045B91F	. 83EA 04	SUB EDX,4	
0045B922	. 8B12	MOV EDX,DWORD PTR DS:[EDX]	
0045B924	> 83FA 01	CMP EDX,1	
0045B927	~ 7C 2D	JL SHORT Advance_.0045B956	
0045B929	> 8B4D FC	MOV ECX,DWORD PTR SS:[EBP-4]	4'th Generate Routine
0045B92C	. 0FB64C11 FF	MOVZX ECX,BYTE PTR DS:[ECX+EDX-1]	
0045B931	. 03F9	ADD EDI,ECX	
0045B933	. 81C7 29090000	ADD EDI,929	
0045B939	. 81C7 67070000	ADD EDI,767	
0045B93F	. 03F8	ADD EDI,EAX	
0045B941	. 69FF 92830000	IMUL EDI,EDI,8392	
0045B947	. 8BC7	MOV EAX,EDI	
0045B949	. 83E8 33	SUB EAX,33	
0045B94C	. 0FAFC7	IMUL EAX,EDI	
0045B94F	. 03C7	ADD EAX,EDI	
0045B951	. 4A	DEC EDX	
0045B952	. 85D2	TEST EDX,EDX	
0045B954	~ ^ 75 D3	JNZ SHORT Advance_.0045B929	

○ 분석

Code 의 위에서 EDI와 EAX 를 XOR 연산하여 0으로 초기화 시킨후, Name 값을 한글자 씩 ECX 에 넣는다.

EDI 에 ECX , 0x929, 0x767, EAX (첫번째 Loop 에는 0) 을 더한다.

EDI * EDI * 0x8392 를 수행한 후, EAX 에 해당 값을 복사한다.

EAX 에서 0x33 을 뺀 후, EAX 에 EDI 값을 더한다.

EDX 를 1 감소 후, EDX 를 Test 하여 JNZ 를 함으로써, 해당 Loop 를 수행한다.

○ 해당 Routine 을 수행하면, EAX 에 결과 값이 들어 있다. (이는 1234 라는 Name 에 대한 네번째 Key Block 값과 관련있다.)

Registers (FPU)	
EAX	A90230C0
ECX	00000031
EDX	00000000
EBX	00A8C6E8 ASCII "1234"
ESP	0012F624
EBP	0012F678
ESI	A672E340
EDI	08EA9292

5'th Generate Routine :

Address	Hex dump	Disassembly	Comment
0045B956	> 33DB	XOR EBX,EBX	
0045B958	. 8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]	
0045B95B	. 85C0	TEST EAX,EAX	
0045B95D	~ 74 05	JE SHORT Advance_.0045B964	
0045B95F	. 83E8 04	SUB EAX,4	
0045B962	. 8B00	MOV EAX,DWORD PTR DS:[EAX]	
0045B964	> 85C0	TEST EAX,EAX	
0045B966	~ 7E 48	JLE SHORT Advance_.0045B9B0	
0045B968	. C745 EC 01000	MOV DWORD PTR SS:[EBP-14],1	
0045B96F	> 8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]	5'th Generate Routine
0045B972	. 8B4D EC	MOV ECX,DWORD PTR SS:[EBP-14]	
0045B975	. 0FB6540A FF	MOVZX EDX,BYTE PTR DS:[EDX+ECX-1]	
0045B97A	. 03DA	ADD EBX,EDX	
0045B97C	. 03DB	ADD EBX,EBX	
0045B97E	. 8BD3	MOV EDX,EBX	
0045B980	. 0FAFD3	IMUL EDX,EBX	
0045B983	. 0FAFDA	IMUL EBX,EDX	
0045B986	. 83F3 10	XOR EBX,10	
0045B989	. 83CB 44	OR EBX,44	
0045B98C	. 69D3 73030000	IMUL EDX,EBX,373	
0045B992	. 81C2 43040000	ADD EDX,443	
0045B998	. 8BD0	MOV EBX,EDX	
0045B99A	. 8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]	
0045B99D	. 8B4D EC	MOV ECX,DWORD PTR SS:[EBP-14]	
0045B9A0	. 0FB6540A FF	MOVZX EDX,BYTE PTR DS:[EDX+ECX-1]	
0045B9A5	. 03DA	ADD EBX,EDX	
0045B9A7	. 0FAFD0	IMUL EBX,EBX	
0045B9AA	. FF45 EC	INC DWORD PTR SS:[EBP-14]	
0045B9AD	. 48	DEC EAX	
0045B9AE	~ ^ 75 BF	JNZ SHORT Advance_.0045B96F	

○ 분석

Loop 전에 EBX 를 XOR 연산하여 0으로 초기화 시키고, EBP - 14 지점에 1을 넣어준다. (단순히 ECX 에 1을 넣어준 것과 같은 동작)

Name 값을 1글자 씩 읽어 들어서 EDX 에 복사한다.

EBX 에 EDX 를 더한 후, EBX 끼리 더한 값을 EDX 에 복사한다.

EDX * EBX 수행 후, EBX * EDX 를 수행한다.

EBX ^ 0x10, EBX || 0x44 를 수행한 후 EBX * EBX * 0x373 을 수행한다.

EDX 에 0x443 을 더한 후, 이 값을 EBX 에 복사한다.

Loop 처음 3줄에 나온 코드가 다시 나오는데 이는 필요 없으므로 Junk Code 로 해석한다.

EBX 에 EDX 를 더한 후, EBX 값을 제공한다.

EBX - 14 (ECX 로 생각해도 무방) 값을 1 증가 한 후, EAX 값을 감소시킨다.

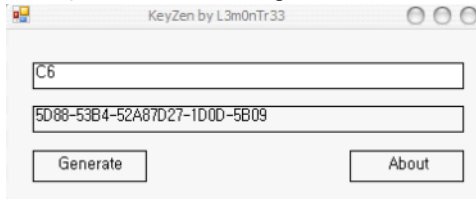
JNZ 구문에 의해 EAX 가 0이 될 때 까지 해당 Loop 를 수행하게 된다.

- 해당 Routine 을 수행하면, EBX 에 결과 값이 들어 있다. (이는 1234 라는 Name 에 대한 다섯번째 Key Block 값과 관련있다.)

Registers (FPU)	
EAX	00000000
ECX	00000004
EDX	00000034
EBX	BB040E29
ESP	0012F624
EBP	0012F678
ESI	A672E340
EDI	08EA9292

3. Conclusion

MD5 Algorithm 을 구해 Load Magic Initialization 부분을 수정하고, 각각의 Routine 에 대한 C Code 를 작성하여 Compile 하여 KeyZen 을 만들어 보았다.



※ 문제는 Key 값을 통해 Name 값을 알아내는 것인데, Name 을 대입하여, 첫번째 Block 의 값이 일치하는 것을 찾아내는 Coding으로도 답을 도출해 낼 수 있다.

답 : C6 md5 = 7E8B9F5CAB4A8FE24FAD9FE4B7452702