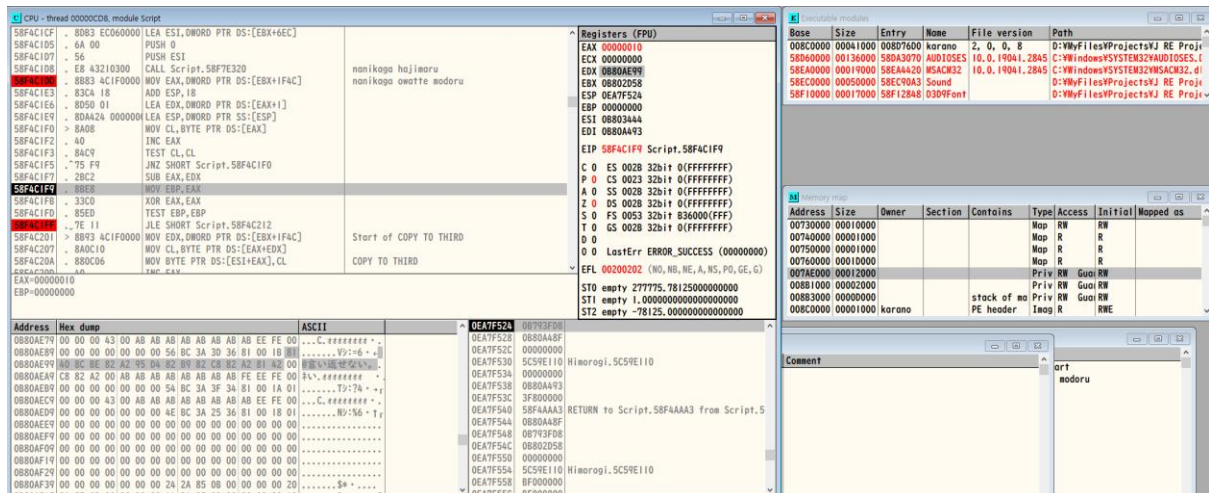


코드파인딩기록10



1C201지점 위쪽에서는 무슨일이 벌어지나 다시 확인해보았다.

EBP에 문장의 길이를 저장하는 작업이 있다.

1C201로 넘어오기 직전 1C1FF에서 점프 하는 상황은, 위에서 확인한 문장의 길이가 0인경우뿐이
구나.

일단 1C1FF의 [EBX+1F4C]에 대사가 담겨있는상황이다.

그렇다면 반복문 이후에는 어떤 일이 일어나나?

Registers (FPU)

Register	Value
EAX	00000054
ECX	00000042
EDX	08C7C110
EBX	08C72D58
ESP	0EDBF864
EBP	00000054
ESI	08C73444
EDI	08B84191
EIP	585BC212 Script.585BC212

Memory map

Address	Size	Owner	Section
08C00000	00001000	karano	
08C10000	0002A000	karano	.text
08EB0000	00007000	karano	.rdata
08F20000	00004000	karano	.data
08F60000	00008000	karano	.rsrc
08FE0000	00003000	karano	.reloc
00AE0000	00010000		
00AF0000	00001000		

반복문이 끝난시점인 1C212에서는 EDX에 어떤 값을 넣는다. 이 값을 V1이라고 하자.

바로다음인 1C219는 EBP에 어떤 값을 넣는다. 이 값을 V2라고 하자.

Registers (FPU)

Register	Value
EAX	00000054
ECX	00000042
EDX	00000058
EBX	08C72D58
ESP	0EDBF864
EBP	08B70048
ESI	08C73444
EDI	08B84191
EIP	585BC21F Script.585BC21F

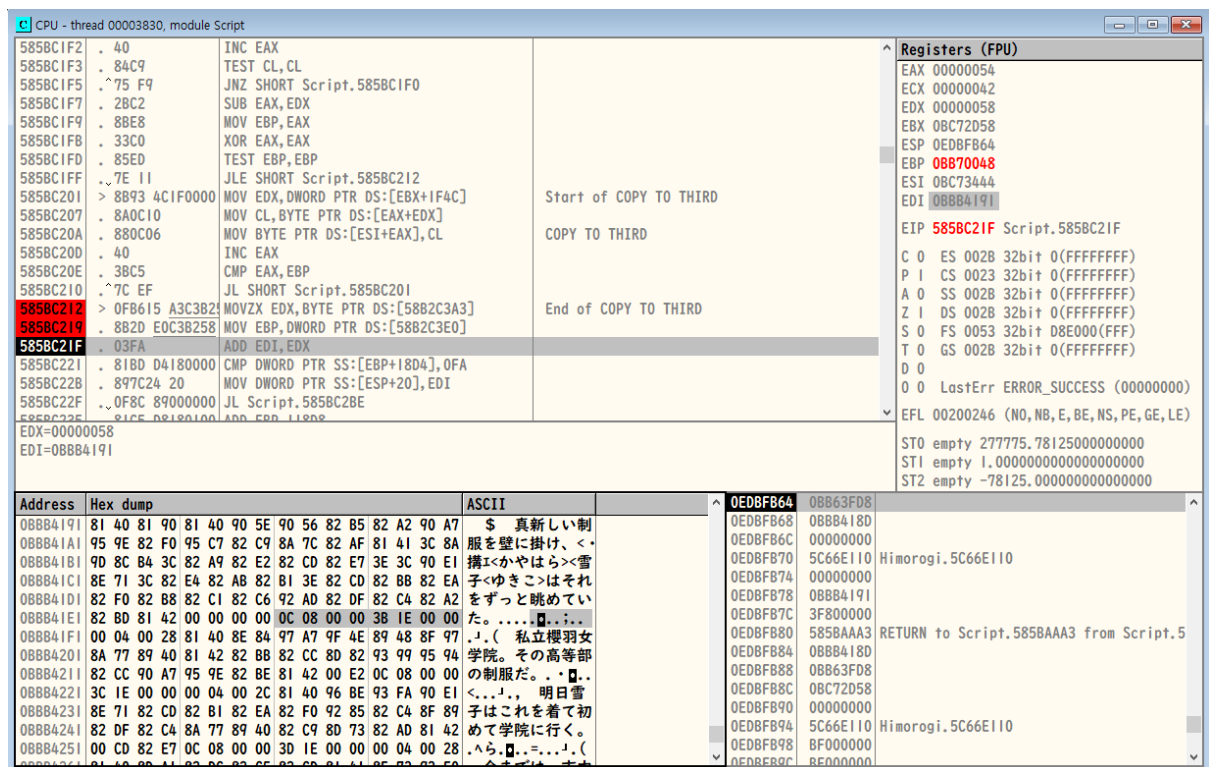
Memory map

Address	Size	Owner	Section
08C00000	00001000	karano	
08C10000	0002A000	karano	.text
08EB0000	00007000	karano	.rdata
08F20000	00004000	karano	.data
08F60000	00008000	karano	.rsrc
08FE0000	00003000	karano	.reloc
00AE0000	00010000		
00AF0000	00001000		

V1은 58, V2는 08B70048이 들어갔다.

EDX = V1

EBP = V2



위 과정 이후 ADD EDI,EDX를 수행하게된다. EDI의 메모리를 살펴보면 복사해올 문장들이 들어간 곳이란걸 알수있다. 그리고 EDX를 더한다는것의 의미를 알기위해 EDX값을 한번 살펴보자.

EDX는 현재 58이라는 값이 들어와있다. 이전 기록에서 우리는 메모리에 저장된 스크립트의 구조에 대해 알아보았다. 위 사진은 (00 04 00 58)로 시작하는 대사를 준비하는 과정이었으니 EDX에 58이라는 문장길이가 들어가있는것이다. 즉 V1은 문장길이를 나타내는것이다.

EDX = V1 = 문장길이(대사 앞의 정보파트에서 가져온값)

Address	Hex dump	ASCII
08B84189	3A 1E 00 00 00 04 00 58 81 40 81 90 81 40 90 5E	...X \$ 真
08B84190	90 56 82 B5 82 A2 90 A7 95 9E 82 F0 95 C7 82 C9	新しい制服を壁に
08B841A9	8A 7C 82 AF 81 41 3C 8A 90 8C B4 3C 82 A9 82 E2	掛け、<茅原<かや>
08B841B9	82 CD 82 E7 3E 3C 90 E1 8E 71 3C 82 E4 82 AB 82	はら<雪子<ゆき>
08B841C9	81 3E 82 CD 82 B8 82 EA 82 F0 82 B8 82 C1 82 C6	はそれをつと
08B841D9	92 AD 82 DF 82 C4 82 A2 82 BD 81 42 00 00 00 00	眺めていた。....
08B841E9	0C 08 00 00 3B 1E 00 00 00 04 00 28 81 40 8E 84	...私...<私
08B841F9	97 A7 9F 4E 89 48 8F 97 8A 77 89 40 81 42 82 B8	立櫻羽女学院。そ
08B84209	82 CC 8D 82 93 99 95 94 82 CC 90 A7 95 9E 82 BE	の高等部の制服だ
08B84219	81 42 00 E2 0C 08 00 00 3C 1E 00 00 00 04 00 2C	。...<...私,
08B84229	81 40 96 BE 93 FA 90 E1 8E 71 82 CD 82 81 82 EA	明日雪子はこれ
08B84239	82 F0 92 85 82 C4 8F 89 82 DF 82 C4 8A 77 89 40	を着て初めて学院
08B84249	82 C9 8D 73 82 AD 81 42 00 CD 82 E7 0C 08 00 00	に行く。...私,

그런데 이상한점이 있다.

00 04 00 58이후부터 0x58의 길이를 따라가본다면 0BBB41D9 라인의 마지막 00 00 00 00까지 포함되어야하는데 우리는 문장의 끝이 0인경우 반복문이 종료되는 방식을 사용하는걸 확인했다.

그렇다면 00 04 00 58이라는 문장정보 길이까지 포함하여 저장해놓은것일까?

그리고, EDX(V1)과 EAX의 값 또한 이상한점이 있다. 직전 반복문에서 EAX로 문장의 길이를 비교하며 EAX가 문장길이에 도달하면 반복문을 나오는 방식이었다. 그런데 EAX와 EDX의 값을 비교해

비교 다르다는 것을 확인 할 수 있다. 이것은 어떻게 된 일일까?

다시 기억을 되짚어보면, 반복문 내부에서 EAX와 EBP를 비교하던 것을 알 수 있다. 그렇다면 EBP가 실제 문장의 길이를 갖고 있을 테다. EBP의 값이 어디서 왔나 확인해보면 반복문의 직전에 00이 발견될때까지 루프를 돌며 문장길이를 검사하는 부분이 있다. 그렇다면 확실해졌다.

대사 직전에 저장된 현재 스크립트 정보파트(아래 사진의 네모박스 파트)는 문장의 길이이기보다는, 현재 사용할 정보길이,대사길이 등등을 모두 포함하여 다음 스크립트 직전까지의 길이를 갖고 있는것이다.

1. 정보파트를 읽어왔다면 처음 1DWORD이후부터 대사길이 검사를 시작한다.
2. EAX로 증가연산을 반복하며 대사끝(00)에 도달했다면 EBP에 대사길이를 저장한다.
3. 다시 반복문을 통해 문장을 다른 메모리로 복사한다.
4. 다음 스크립트를 읽어오기위해 EDI(현재스크립트의시작점)에 EDX(V1)(정보파트의 후반부값)을 더해준다.
5. 위 과정이 끝났으면 EDI는 이제 다음 스크립트의 시작점에 도착하게된다.

Address	Hex dump	ASCII
08B84189	3A 1E 00 00 00 04 00 58 81 40 81 90 81 40 90 5E	:...X \$ 真
08B84199	90 56 82 B5 82 A2 90 A7 95 9E 82 F0 95 C7 82 C9	新しい制服を壁に
08B841A9	8A 7C 82 AF 81 41 3C 8A 9D 8C 84 3C 82 A9 82 E2	掛け、<矛盾<かや
08B841B9	82 CD 82 E7 3E 3C 90 E1 8E 71 3C 82 E4 82 AB 82	はら><雪子<ゆき
08B841C9	B1 3E 82 CD 82 B8 82 EA 82 F0 82 B8 82 C1 82 C6	フはそれをずっと
08B841D9	92 AD 82 DF 82 C4 82 A2 82 8D 81 42 00 00 00 00	眺めていた。....
08B841E9	0C 08 00 00 38 1E 00 00 00 04 00 28 81 40 8E 84	.<...J.(私
08B841F9	97 A7 9F 4E 89 48 8F 97 8A 77 89 40 81 42 82 B8	立櫻羽女学院。そ
08B84209	82 CC 8D 82 93 99 95 94 82 CC 90 A7 95 9E 82 BE	の高等部の制服だ
08B84219	81 42 00 E2 0C 08 00 00 3C 1E 00 00 00 04 00 2C	。..<...J..
08B84229	81 40 96 BE 93 FA 90 E1 8E 71 82 CD 82 B1 82 EA	明日雪子はこれ
08B84239	82 F0 92 85 82 C4 8F 89 82 DF 82 C4 8A 77 89 40	を見て初めて学院
08B84249	82 C9 8D 73 82 AD 81 42 00 CD 82 E7 0C 08 00 00	に行く。..<...J..
08B84259	3B 1F 88 88 88 88 88 88 88 88 88 88 88 88 88 88	..<...J..

참고로, 문장이 끝난 뒤 다음문장으로 이어질 때 길이가 경우에따라 다른것같다...

....

일단은 이런 사소한건 깊게 생각하지 않고, 넘어가야할듯 하다.

결국 커맨드 {1C21F ADD EDI,EDX}의 목적은

EDI 레지스터를 다음 문장의 시작점으로 보내기 위해 EDX(정보파트 후반부값)(스크립트길이)만큼 더해주는것이다.

Disassembler (CPU)

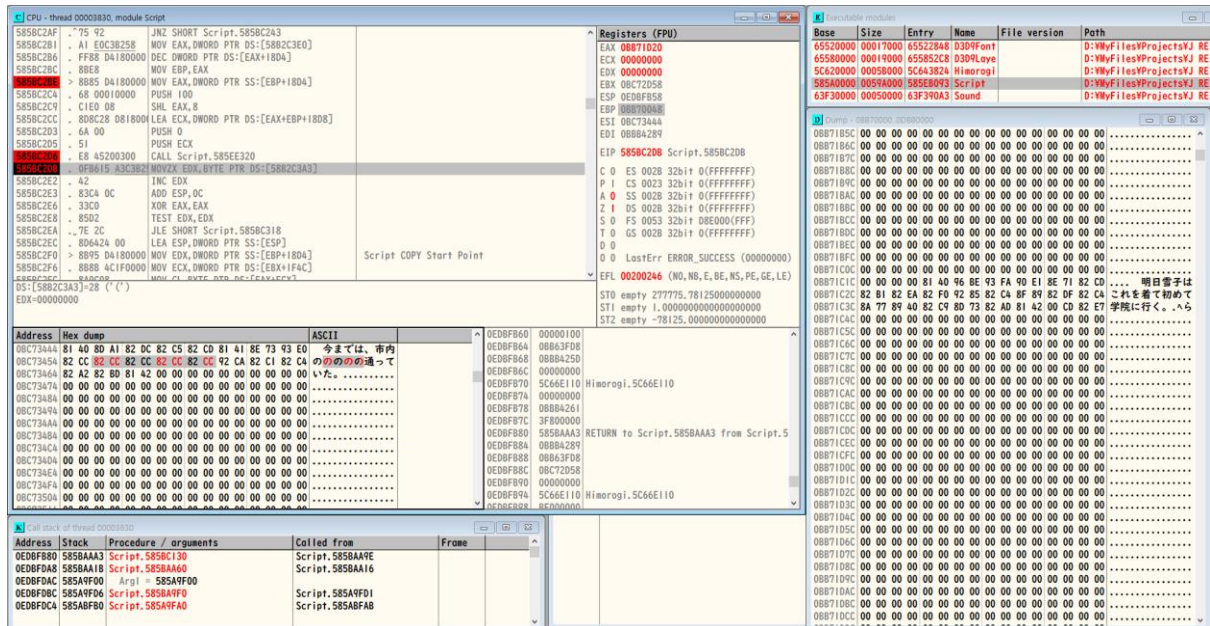
```

585BC201 > 8073 4C10F000 MOV EAX, DWORD PTR DS:[EBX+14C]
585BC207 > 8A10 00000000 MOV CL, BYTE PTR DS:[EAX+ED1]
585BC20A > 74 00 MOV EBP, PTR DS:[ESI+ED1]
585BC20D > 40 INC EAX
585BC20E > 30C5 CAX EAX, EBP
585BC210 > 7C FF JLT SHORT Script.585BC201
585BC212 > 0FBA15 43C3B2 MOVZX EDI, BYTE PTR DS:[5882C3A3]
585BC218 > 8B20 00C3B258 MOV EBP, DWORD PTR DS:[5882C3E0]
585BC21D > 03FA ADD EDI, EDX
585BC221 > 0FBE 1A100000 MOVSI EDI, DWORD PTR DS:[ESI+ED1+100], 0FA
585BC228 > B97C2A 20 MOV DWORD PTR DS:[ESP+20], EDI
585BC22F > 0F8C 07000000 JLT Script.585BC28E
585BC235 > 91C5 08000000 ADD EBP, 1808
585BC23B > C7A42A 10 F000 MOV DWORD PTR DS:[ESP+10], 0F0
585BC238 > 68 00010000 PUSH 10
585BC248 > 6A 00 PUSH 0
585BC24A > 55 PUSH EBP
585BC24D > 08 00200300 CALL Script.585EE32C
585BC250 > 8005 00000000 LEA ESI, DWORD PTR DS:[EBP+100]
585BC256 > 87 00000000 MOV ECX, 0
585BC25D > 90C5 JNC SHORT 585BC26A
585BC261 > 55 PUSH EBP
585BC262 > 55 PUSH EBP
585BC263 > 55 PUSH EBP
585BC264 > 55 PUSH EBP
585BC265 > 55 PUSH EBP
585BC266 > 55 PUSH EBP
585BC267 > 55 PUSH EBP
585BC268 > 55 PUSH EBP
585BC269 > 55 PUSH EBP
585BC26A > 55 PUSH EBP
585BC26B > 55 PUSH EBP
585BC26C > 55 PUSH EBP
585BC26D > 55 PUSH EBP
585BC26E > 55 PUSH EBP
585BC26F > 55 PUSH EBP
585BC270 > 55 PUSH EBP
585BC271 > 55 PUSH EBP
585BC272 > 55 PUSH EBP
585BC273 > 55 PUSH EBP
585BC274 > 55 PUSH EBP
585BC275 > 55 PUSH EBP
585BC276 > 55 PUSH EBP
585BC277 > 55 PUSH EBP
585BC278 > 55 PUSH EBP
585BC279 > 55 PUSH EBP
585BC27A > 55 PUSH EBP
585BC27B > 55 PUSH EBP
585BC27C > 55 PUSH EBP
585BC27D > 55 PUSH EBP
585BC27E > 55 PUSH EBP
585BC27F > 55 PUSH EBP
585BC280 > 55 PUSH EBP
585BC281 > 55 PUSH EBP
585BC282 > 55 PUSH EBP
585BC283 > 55 PUSH EBP
585BC284 > 55 PUSH EBP
585BC285 > 55 PUSH EBP
585BC286 > 55 PUSH EBP
585BC287 > 55 PUSH EBP
585BC288 > 55 PUSH EBP
585BC289 > 55 PUSH EBP
585BC28A > 55 PUSH EBP
585BC28B > 55 PUSH EBP
585BC28C > 55 PUSH EBP
585BC28D > 55 PUSH EBP
585BC28E > 55 PUSH EBP
585BC28F > 55 PUSH EBP
585BC290 > 55 PUSH EBP
585BC291 > 55 PUSH EBP
585BC292 > 55 PUSH EBP
585BC293 > 55 PUSH EBP
585BC294 > 55 PUSH EBP
585BC295 > 55 PUSH EBP
585BC296 > 55 PUSH EBP
585BC297 > 55 PUSH EBP
585BC298 > 55 PUSH EBP
585BC299 > 55 PUSH EBP
585BC29A > 55 PUSH EBP
585BC29B > 55 PUSH EBP
585BC29C > 55 PUSH EBP
585BC29D > 55 PUSH EBP
585BC29E > 55 PUSH EBP
585BC29F > 55 PUSH EBP
585BC2A0 > 55 PUSH EBP
585BC2A1 > 55 PUSH EBP
585BC2A2 > 55 PUSH EBP
585BC2A3 > 55 PUSH EBP
585BC2A4 > 55 PUSH EBP
585BC2A5 > 55 PUSH EBP
585BC2A6 > 55 PUSH EBP
585BC2A7 > 55 PUSH EBP
585BC2A8 > 55 PUSH EBP
585BC2A9 > 55 PUSH EBP
585BC2AA > 55 PUSH EBP
585BC2AB > 55 PUSH EBP
585BC2AC > 55 PUSH EBP
585BC2AD > 55 PUSH EBP
585BC2AE > 55 PUSH EBP
585BC2AF > 55 PUSH EBP
585BC2B0 > 55 PUSH EBP
585BC2B1 > 55 PUSH EBP
585BC2B2 > 55 PUSH EBP
585BC2B3 > 55 PUSH EBP
585BC2B4 > 55 PUSH EBP
585BC2B5 > 55 PUSH EBP
585BC2B6 > 55 PUSH EBP
585BC2B7 > 55 PUSH EBP
585BC2B8 > 55 PUSH EBP
585BC2B9 > 55 PUSH EBP
585BC2BA > 55 PUSH EBP
585BC2BB > 55 PUSH EBP
585BC2BC > 55 PUSH EBP
585BC2BD > 55 PUSH EBP
585BC2BE > 55 PUSH EBP
585BC2BF > 55 PUSH EBP
585BC2C0 > 55 PUSH EBP
585BC2C1 > 55 PUSH EBP
585BC2C2 > 55 PUSH EBP
585BC2C3 > 55 PUSH EBP
585BC2C4 > 55 PUSH EBP
585BC2C5 > 55 PUSH EBP
585BC2C6 > 55 PUSH EBP
585BC2C7 > 55 PUSH EBP
585BC2C8 > 55 PUSH EBP
585BC2C9 > 55 PUSH EBP
585BC2CA > 55 PUSH EBP
585BC2CB > 55 PUSH EBP
585BC2CC > 55 PUSH EBP
585BC2CD > 55 PUSH EBP
585BC2CE > 55 PUSH EBP
585BC2CF > 55 PUSH EBP
585BC2D0 > 55 PUSH EBP
585BC2D1 > 55 PUSH EBP
585BC2D2 > 55 PUSH EBP
585BC2D3 > 55 PUSH EBP
585BC2D4 > 55 PUSH EBP
585BC2D5 > 55 PUSH EBP
585BC2D6 > 55 PUSH EBP
585BC2D7 > 55 PUSH EBP
585BC2D8 > 55 PUSH EBP
585BC2D9 > 55 PUSH EBP
585BC2DA > 55 PUSH EBP
585BC2DB > 55 PUSH EBP
585BC2DC > 55 PUSH EBP
585BC2DD > 55 PUSH EBP
585BC2DE > 55 PUSH EBP
585BC2DF > 55 PUSH EBP
585BC2E0 > 55 PUSH EBP
585BC2E1 > 55 PUSH EBP
585BC2E2 > 55 PUSH EBP
585BC2E3 > 55 PUSH EBP
585BC2E4 > 55 PUSH EBP
585BC2E5 > 55 PUSH EBP
585BC2E6 > 55 PUSH EBP
585BC2E7 > 55 PUSH EBP
585BC2E8 > 55 PUSH EBP
585BC2E9 > 55 PUSH EBP
585BC2EA > 55 PUSH EBP
585BC2EB > 55 PUSH EBP
585BC2EC > 55 PUSH EBP
585BC2ED > 55 PUSH EBP
585BC2EE > 55 PUSH EBP
585BC2EF > 55 PUSH EBP
585BC2F0 > 55 PUSH EBP
585BC2F1 > 55 PUSH EBP
585BC2F2 > 55 PUSH EBP
585BC2F3 > 55 PUSH EBP
585BC2F4 > 55 PUSH EBP
585BC2F5 > 55 PUSH EBP
585BC2F6 > 55 PUSH EBP
585BC2F7 > 55 PUSH EBP
585BC2F8 > 55 PUSH EBP
585BC2F9 > 55 PUSH EBP
585BC2FA > 55 PUSH EBP
585BC2FB > 55 PUSH EBP
585BC2FC > 55 PUSH EBP
585BC2FD > 55 PUSH EBP
585BC2FE > 55 PUSH EBP
585BC2FF > 55 PUSH EBP
585BC300 > 55 PUSH EBP
585BC301 > 55 PUSH EBP
585BC302 > 55 PUSH EBP
585BC303 > 55 PUSH EBP
585BC304 > 55 PUSH EBP
585BC305 > 55 PUSH EBP
585BC306 > 55 PUSH EBP
585BC307 > 55 PUSH EBP
585BC308 > 55 PUSH EBP
585BC309 > 55 PUSH EBP
585BC30A > 55 PUSH EBP
585BC30B > 55 PUSH EBP
585BC30C > 55 PUSH EBP
585BC30D > 55 PUSH EBP
585BC30E > 55 PUSH EBP
585BC30F > 55 PUSH EBP
585BC310 > 55 PUSH EBP
585BC311 > 55 PUSH EBP
585BC312 > 55 PUSH EBP
585BC313 > 55 PUSH EBP
585BC314 > 55 PUSH EBP
585BC315 > 55 PUSH EBP
585BC316 > 55 PUSH EBP
585BC317 > 55 PUSH EBP
585BC318 > 55 PUSH EBP
585BC319 > 55 PUSH EBP
585BC31A > 55 PUSH EBP
585BC31B > 55 PUSH EBP
585BC31C > 55 PUSH EBP
585BC31D > 55 PUSH EBP
585BC31E > 55 PUSH EBP
585BC31F > 55 PUSH EBP
585BC320 > 55 PUSH EBP
585BC321 > 55 PUSH EBP
585BC322 > 55 PUSH EBP
585BC323 > 55 PUSH EBP
585BC324 > 55 PUSH EBP
585BC325 > 55 PUSH EBP
585BC326 > 55 PUSH EBP
585BC327 > 55 PUSH EBP
585BC328 > 55 PUSH EBP
585
```

[EBP+18D4]가 무엇인지 궁금해서 메모리 덤프로 이동해보았는데...

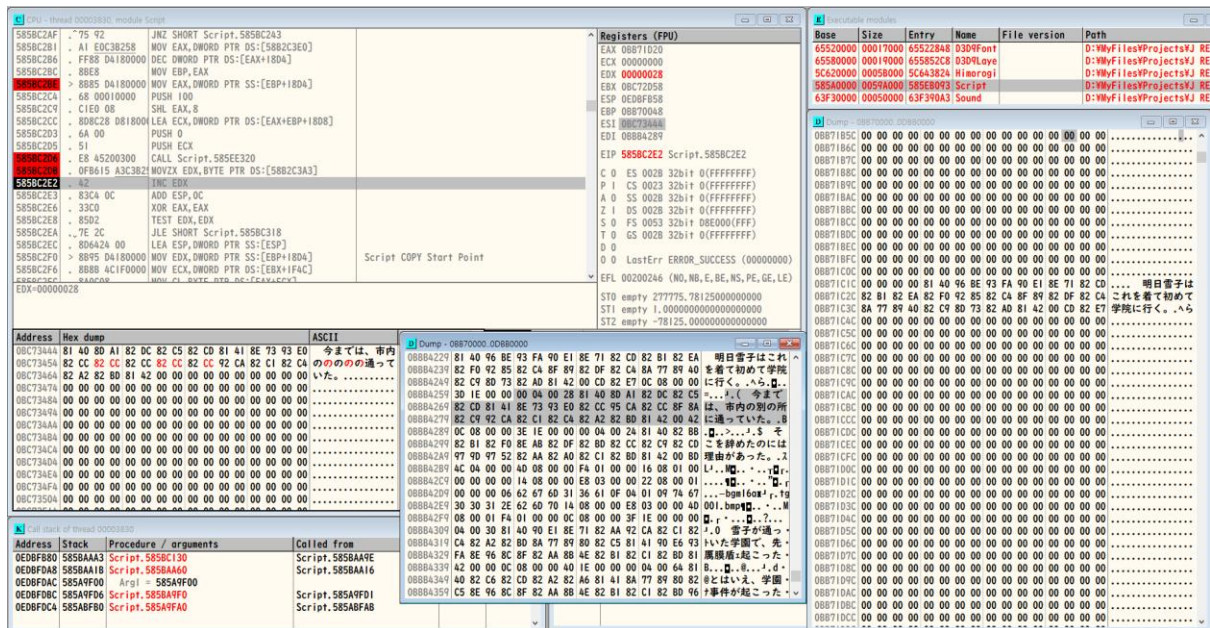
그 다음 `MOV DWORD PTR SS:[ESP+20],EDI` 이것은 보아하니 EDI를 현재스택+20에 저장해둔 것이
다. 분명 어디론가 점프하기 전에 EDI값을 저장해놓고 가려는 것이겠지.

면 어떤 변화가 생길지 궁금해졌다.



ESI가 가리키는 주소인 0BC73444의 값 중 중간간의 일부를 82CC(の)로 바꾸었다. 바꾼부분 중간중간이 붉은 색으로 안바뀌었는데 버그인가? 아무튼 바꾼거임.

우측의 메모리덤프는 백로그 메모리이며, 아랫부분에 바꾼 대사가 기록이 될것으로 기대중이다.



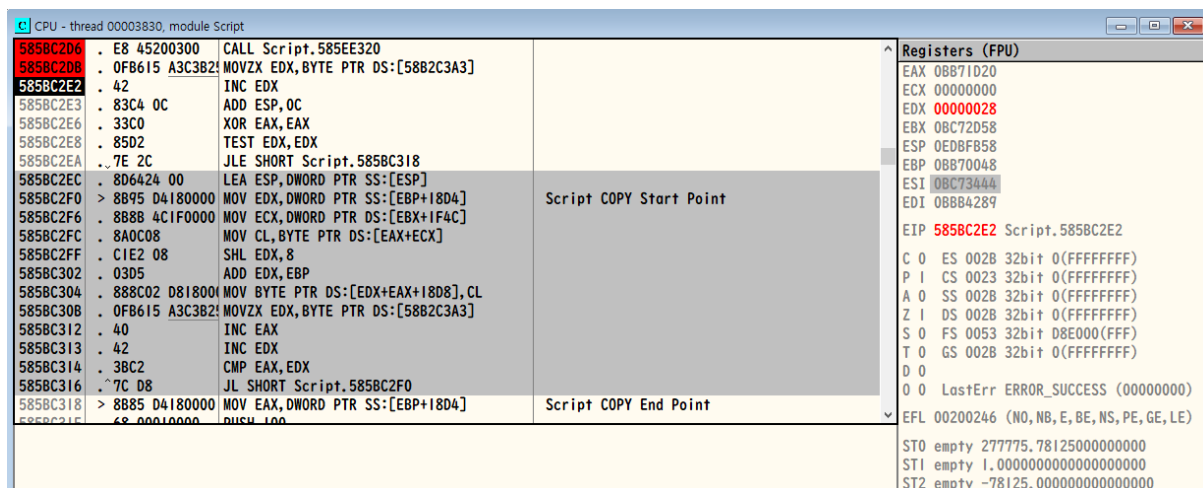
MOVZX부분을 실행하니 EDX에 현재스크립트길이가 들어왔다.

그리고 INC EDX부터 JLE Script...까지는 다음과 같은 역할이다.

EDX를 1 증가, ESP를 0C(4칸)증가, EAX 0으로 초기화,

EDX가 0인지 검사, EDX가 0이면 다음코드 스킵

이때 다음코드라는 것은 아래 사진에 회색으로 표시된 것이다.



Script COPY Start Point라는 주석을 달아놓은것으로 보아 뭔가 복사하는 부분인듯하다.

뭘 복사하는건지 안적어놔는데 이전 기록 찾아보면서 상대주소 계산하기 귀찮아서 그냥 실행해보면서 다시 알아보도록 하자. 아니 잠시만요? 시작하는 모습을 보십시오.

MOV EDX, ~~[EBP+18D4]

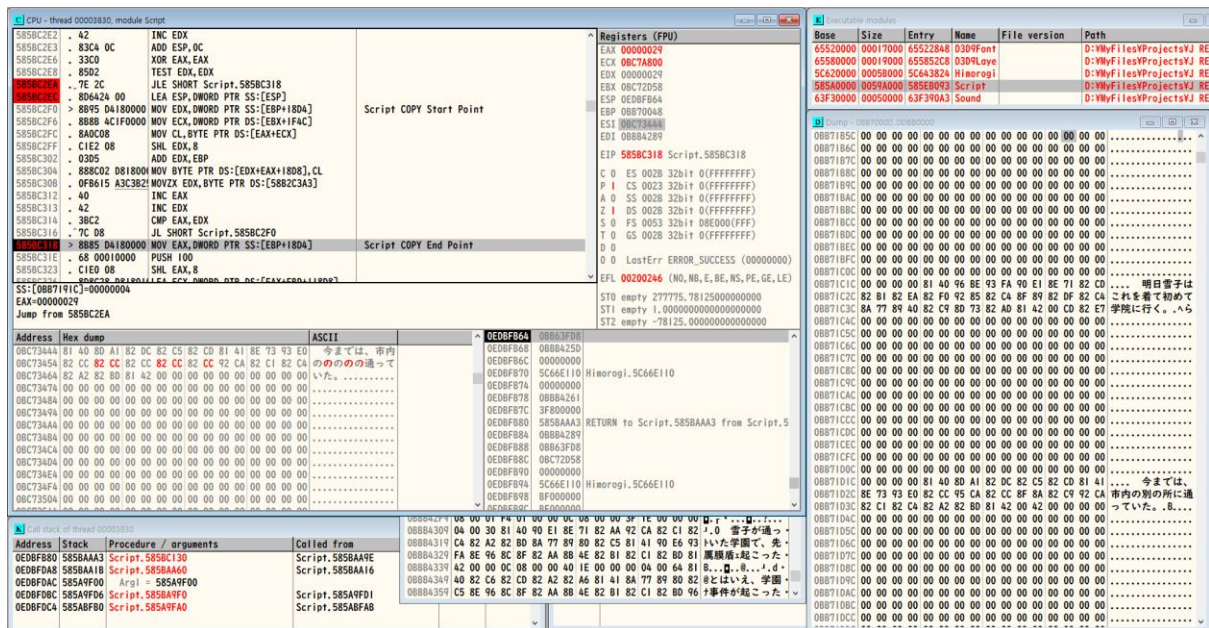
MOV ECX, ~~[EBX+1F4C]

참으로 익숙한 주소값들이다. [EBP+18D4]는 백로그 시작지점, [EBX+1F4C]는 대사메모리.

그렇다면 대사를 백로그로 복사하는 코드가 맞는것같다!

뒤에 나오는 코드를 보면 백로그 자리 계산하는 코드와 CL에 대사를 담아서 EDX(EBP+18D4를 이용해 계산한 백로그 메모리자리)로 옮기는 코드 등이 보인다. 이곳을 잘 기억해두자.

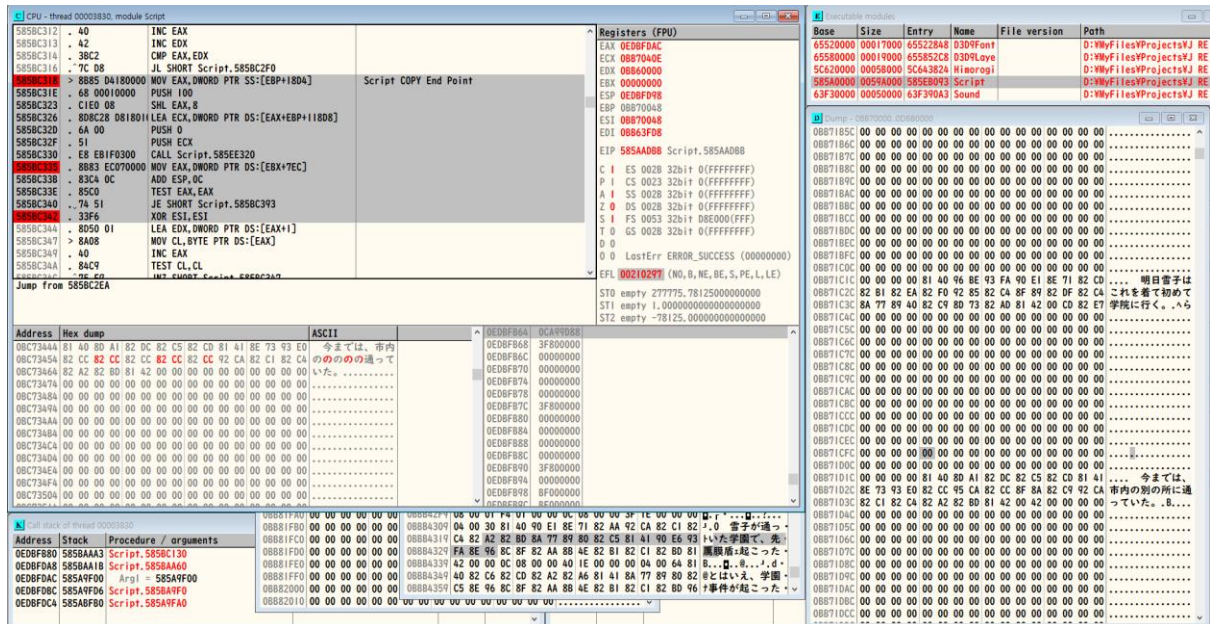
백로그 기록 시작지점 585BC2F0 – 585A0000 = 1C2F0



기록이 잘 되어있는 모습을 확인했다.

백로그 기록까지 완료했는데 아직 화면 출력이 안되었다. 프로그램 상태도 디버거에 의해 Pause 상태이고.. 그렇다면 화면 출력은 언제하는것인가?

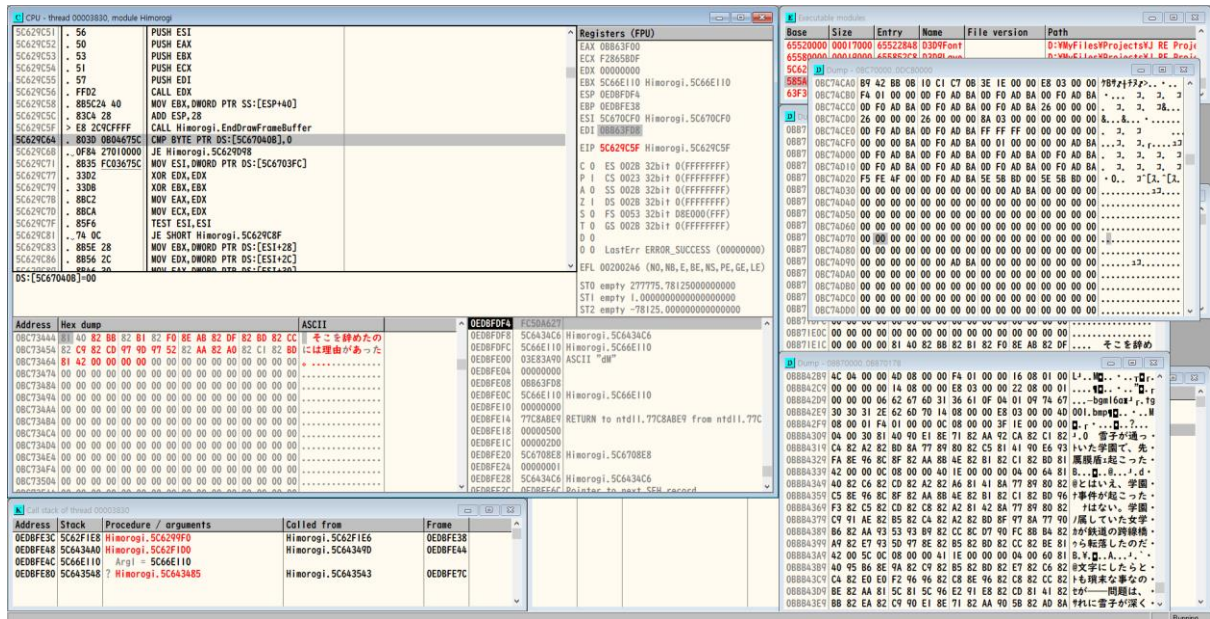
일단 백로그 기록 이후 코드에서 CALL문이 나오는 지점마다 바로 다음에 브포를 걸고, 스크립트 콜 이후에 어떤일이 일어나는지 알아보았다.



첫번째 콜(585BC330) 이후에는 무슨 일이 일어났는지 모르겠다. 상태는 디버거에 의해 일시정지된 상태이다.

두번째 콜(585BC340) 이후에는 프로그램이 Running으로 바뀌며 화면 출력이 완료되고 클릭하여 다음 대사로 넘어갈 수 있는 상태가 되었다. 그렇다면 이곳이 중요한 포인트라는 이야기다.

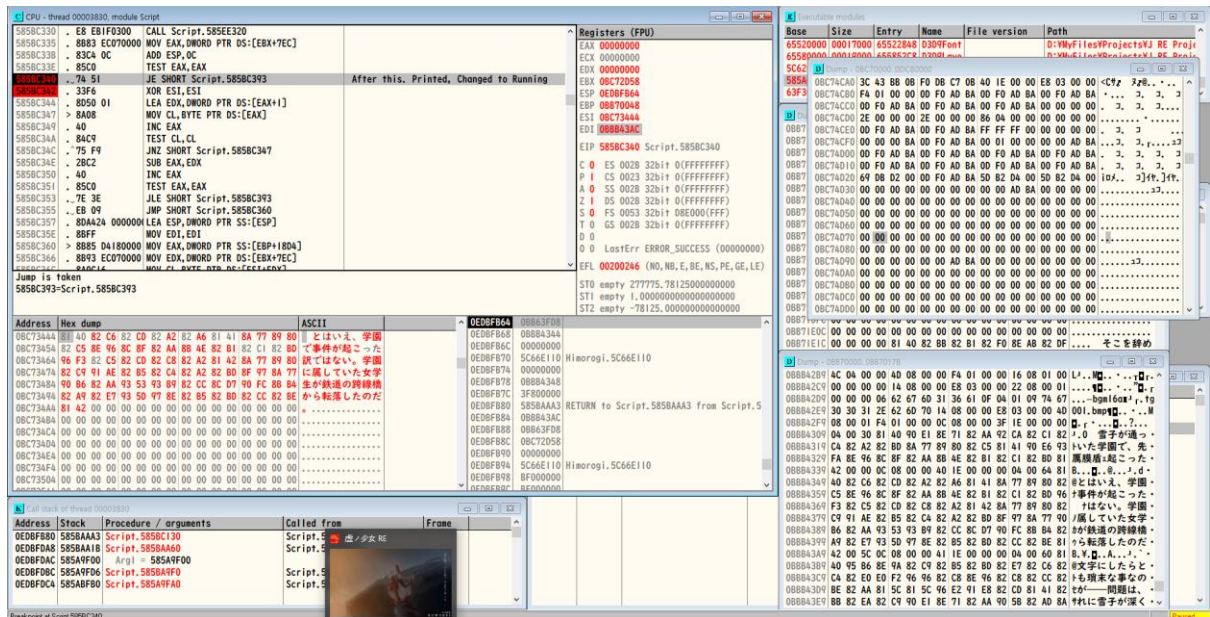
585BC340 (1C340) 이후 화면 출력완료, 프로그램 Running으로 바뀜.



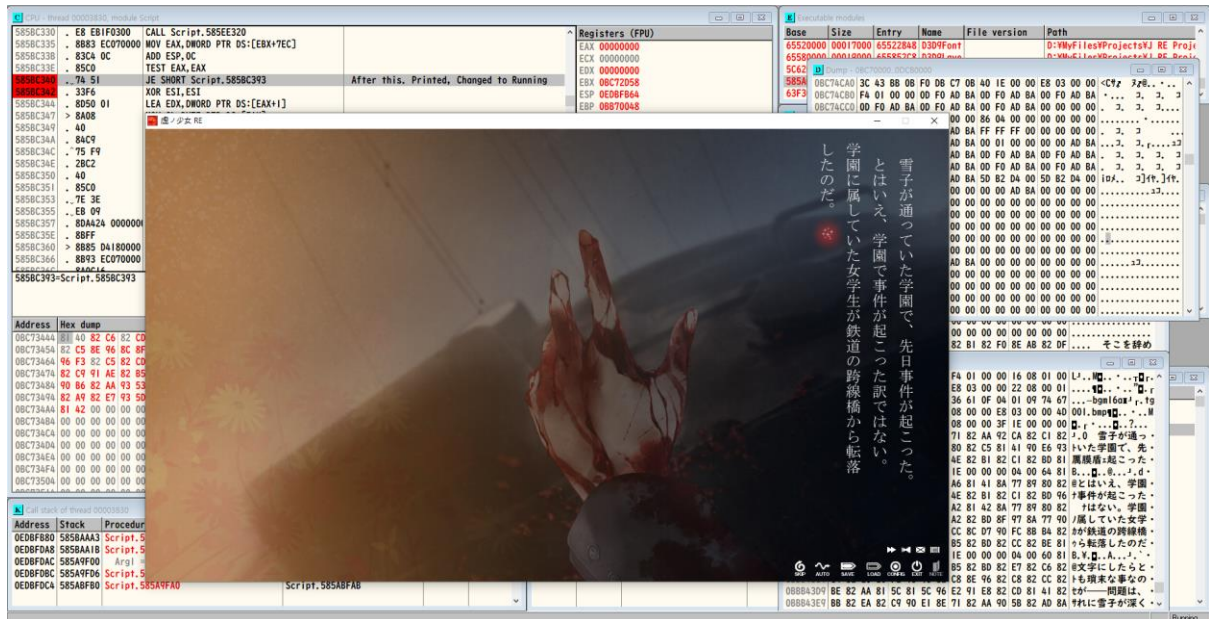
위 코드 이후를 테스트해보면서 도대체 이게 뭐냐..하면서 넘겼는데 Himorogi.dll 넘어가고 점프문에 정신을 못차리다가 어느순간 EndDrawFrameBuffer까지 도달했을때의 당혹감은 말로 표현할수 없다. 정말 몇번 넘긴게 없다. 100번은 고사하고 50 라인조차 지나지 않은것같은데 벌써 화면 출력인가..? 아무튼 이렇게 도달해버리면 Running으로 전환되면서 화면에는 대사가 다 출력되어있고 클릭 가능해진다.

다시 이전 상황까지 가보자.

585BC340(1C340)에 브포를 걸어보고 F9로 넘겨보는거다. 분명 Running으로 바뀌고 화면출력되고 마찬가지로겠지?



혹시 이곳으로 다시 돌아오려나 싶어서 바로 다음줄에도 브포를 걸어놓음. F9로 진행해보자.



음.. 다음줄로는 안가고 끝나버렸다. 그럴것같긴했는데, 그럼 이 부근에 후킹을 걸어줘야하는건가?

근데 지금 이런짓을 하고있는이유가 뭐였지? 문득 생각나버렸다.

곰곰이(쿠마쿠마쿠마베어 아님) 생각해보니 화면출력번역을 하려고 깊은곳까지 들어온것이었다. 백로그 번역이 되는지 확인을 해보지 않았던건 왜지?? 화면출력 번역에 정신이 팔려서 그런것같은데.. 다음과 같은 사고과정이 있던것 같다.

1. 화면출력과 백로그기록은 같은 메모리로부터 대사를 받아오지 않을까?
2. 코드 분석을 하다보니 백로그 기록이 먼저나옴.
3. 백로그쪽에서 대사 받아오는부분이 어딘가 찾음.
4. 후킹해봤는데 화면출력이 안바뀜. 뭔가 이상함을 느낌.
5. 근데 백로그 번역은 되나 확인을 안해봄.<-완전 바보야



6.

그렇게 된것입니다...

그럼 이번 기록은 여기서 마무리.

오늘의 정리

화면 출력과 백로그 기록은 다른 메모리로부터 가져오는것으로 추정되는데 확실치는 않다. 이렇게 추정하는 이유는 백로그 기록을 가져오는 지점에서 아무리 후킹을 해도 DumpText에는 뜨는데 화면출력은 번역이 되지 않았기 때문으로 한가지 이유, 그리고 화면출력쪽으로 코드를 더 분석해 나가면서 뭔가 다른 메모리를 사용하는 김새를 발견했기 때문으로 두가지 이유를 들 수 있다.

<다음에 해야할 것 정리>

1. 이번 기록에서 찾은 이 지점

585BC32D	. 6A 00	PUSH 0	
585BC32F	. 51	PUSH ECX	
585BC330	. E8 EB1F0300	CALL Script.585EE320	
585BC335	. 8B83 EC070000	MOV EAX,DWORD PTR DS:[EBX+7EC]	
585BC33B	. 83C4 0C	ADD ESP,0C	
585BC33E	. 85C0	TEST EAX,EAX	
585BC340	. 74 51	JE SHORT Script.585BC393	After this. Printed, Changed to Running
585BC342	. 33F6	XOR ESI,ESI	
585BC344	. 8D50 01	LEA EDX,DWORD PTR DS:[EAX+1]	
585BC347	> 8A08	MOV CL,BYTE PTR DS:[EAX]	
585BC349	. 40	INC EAX	
585BC34A	. 84C9	TEST CL,CL	
585BC34C	. 75 F9	JNZ SHORT Script.585BC347	
585BC34E	. 2BC2	SUB EAX,EDX	
585BC350	. 40	INC EAX	
585BC351	. 85C0	TEST EAX,EAX	
585BC353	. 7E 3E	JLE SHORT Script.585BC393	
585BC355	. EB 09	JMP SHORT Script.585BC360	
585BC357	. 8DA424 00000000	LEA ESP,DWORD PTR SS:[ESP]	
585BC35E	. 8BFF	MOV EDI,EDI	
585BC360	> 8B8F D4180000	MOV EAX,DWORD PTR SS:[EBP+18D4]	
585BC393=Script.585BC393			

1C340 부근을 자세히 살펴보며 화면출력 번역에 한걸음 더 나아가야한다.

2. 지난 기록에서 백로그쪽을 살펴보며 찾은 1C201, 1C212 주소들을 다시 테스트해보면서 백로그 번역이 되는지도 확인해야겠다.