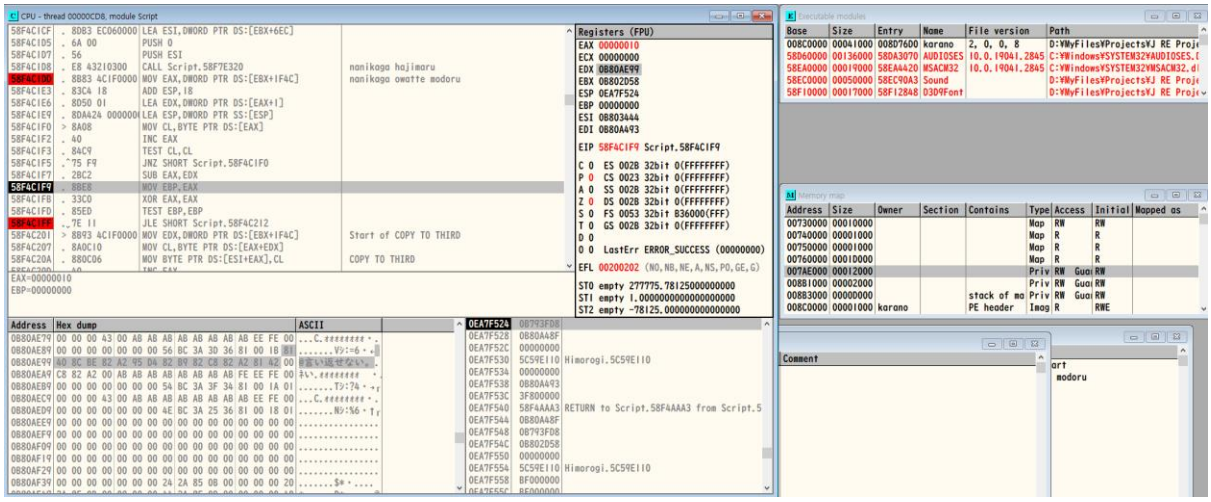
코드파인딩기록10



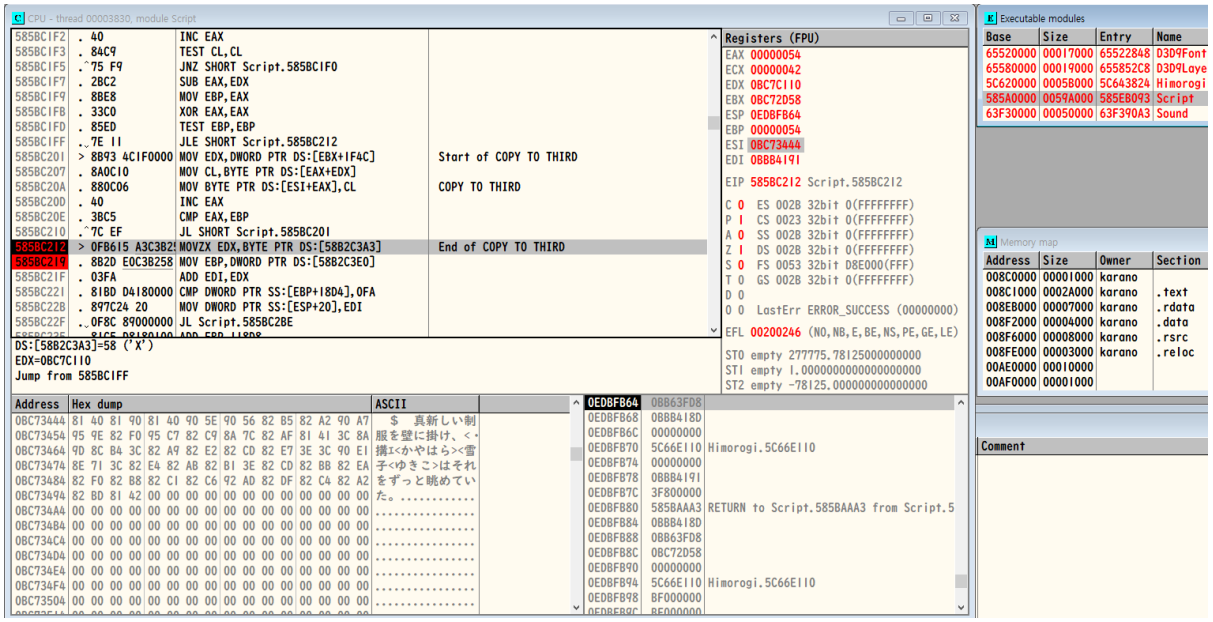
1C201지점 위쪽에서는 무슨일이 벌어지나 다시 확인해보았다.

EBP에 문장의 길이를 저장하는 작업이 있다.

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

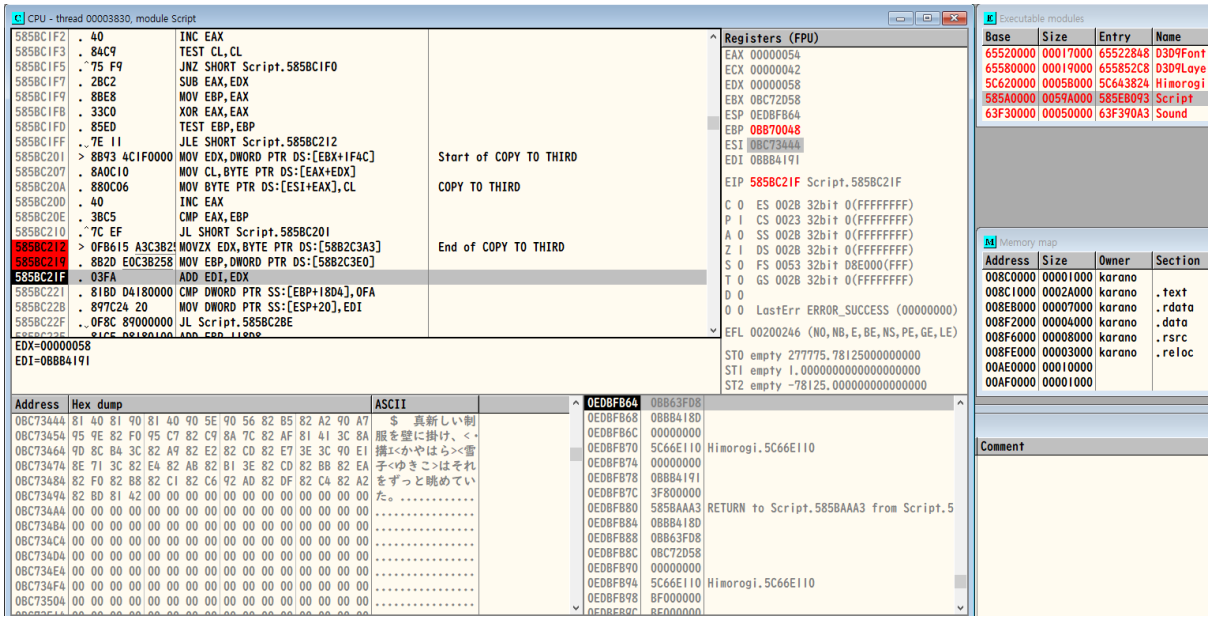
일단 1C1FF의 [EBX+1F4C]에 대사가 담겨있는상황이다.

그렇다면 반복문 이후에는 어떤 일이 일어나나?



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

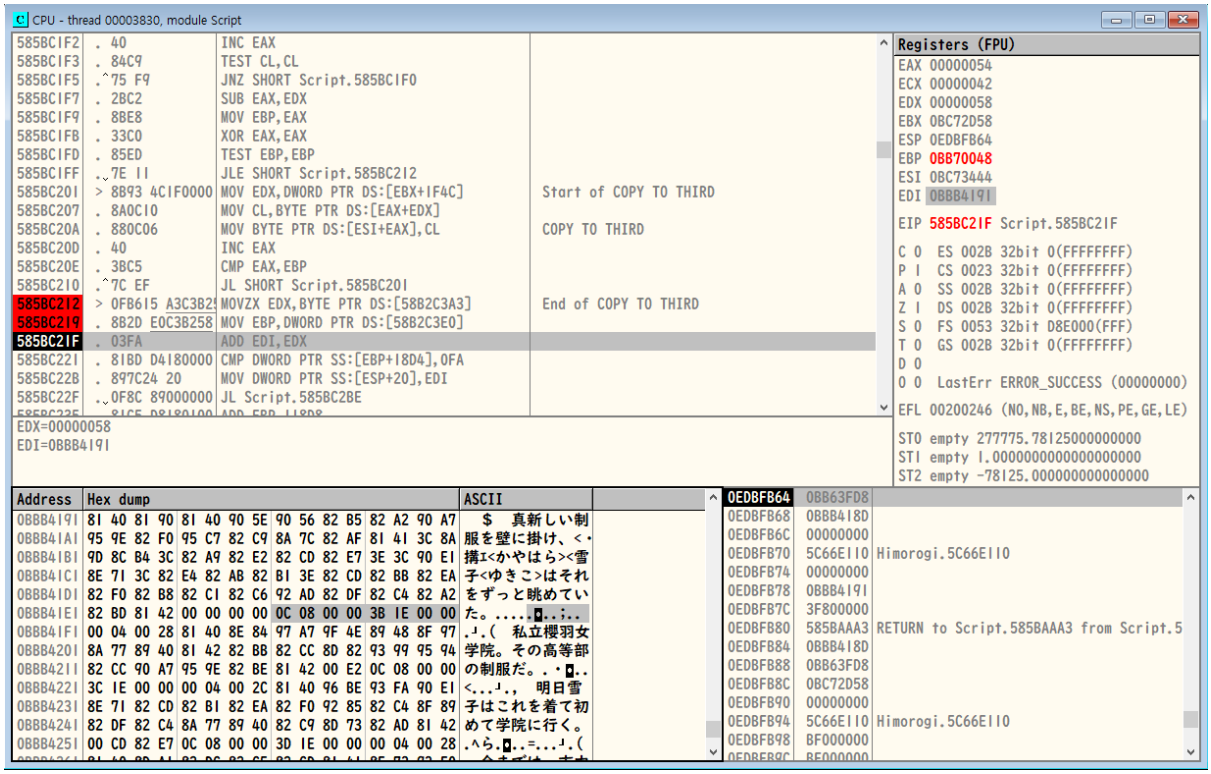
바로다음인 1C219는 EBP에 어떤 값을 넣는다. 이 값을 V2라고 하자.



V1은 58, V2는 0BB70048이 들어갔다.

EDX = V1

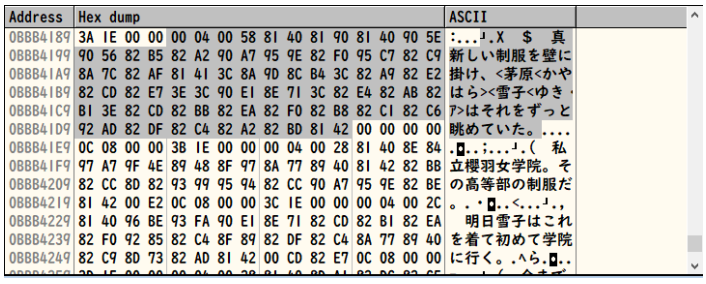
EBP = V2



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)



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?

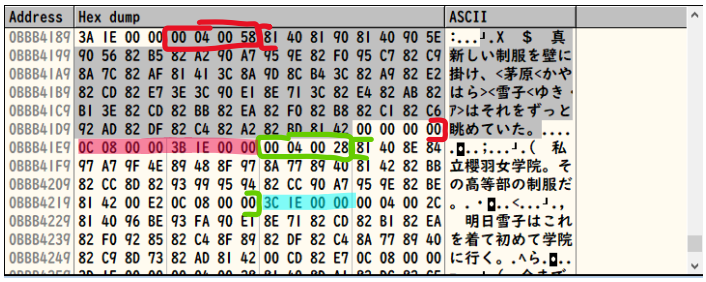
그리고, EDX(V1)과 EAX의 값 또한 이상한점이 있다. 직전 반복문에서 EAX로 문장의 길이를 비교하며 EAX가 문장길이에 도달하면 반복문을 나오는 방식이었다. 그런데 EAX와 EDX의 값을 비교해

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 길이라기보다는, 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.



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

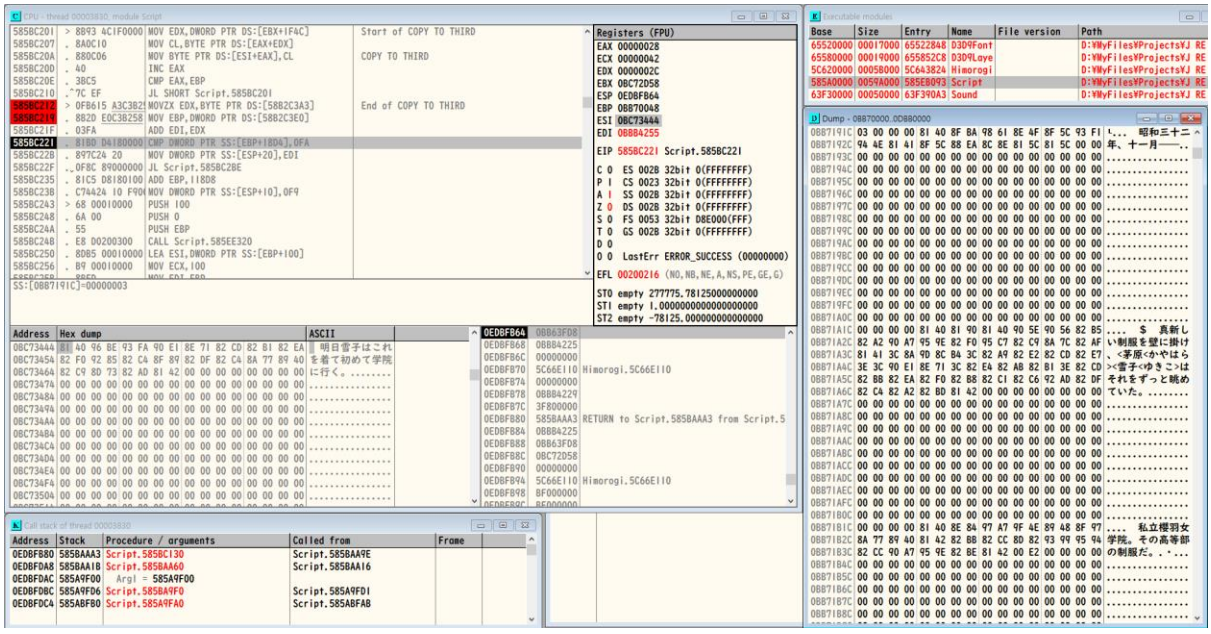
....

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.

이제 V2(EBP)의 값은 어떤용도인지 의문이다. EBP값을 사용하는 다음 커맨드를 살펴보자.



CMP DWORD PTR SS:[EBP+18D4],0FA

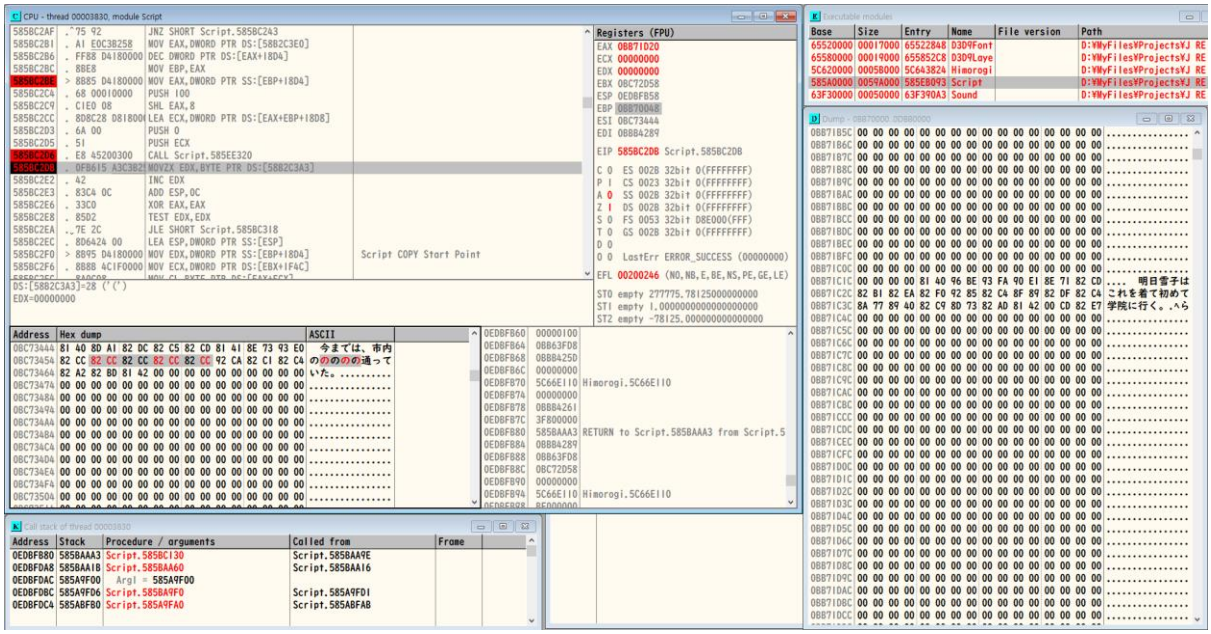
[EBP+18D4]가 무엇인지 궁금해서 메모리 덤프로 이동해보았는데...!

세상에나 이곳이 바로 백로그 메모리였다! V2는 백로그의 시작지점을 나타내느것이었다! 1C219에서 EBP에 수상한값을 넣은 것이 이를 위한것이었을까. 그런데 0FA와 비교하는 것은 무슨 의도인지 모르겠다. 메모리상에서 0FA가 사용된 곳을 발견한다면 좋겠지만 기억에 남아있는곳은 없다.

그 다음 MOV DWORD PTR SS:[ESP+20],EDI 이것은 보아하니 EDI를 현재스택+20에 저장해둔것이다. 분명 어디론가 점프하기 전에 EDI값을 저장해놓고 가려는것이겠지.

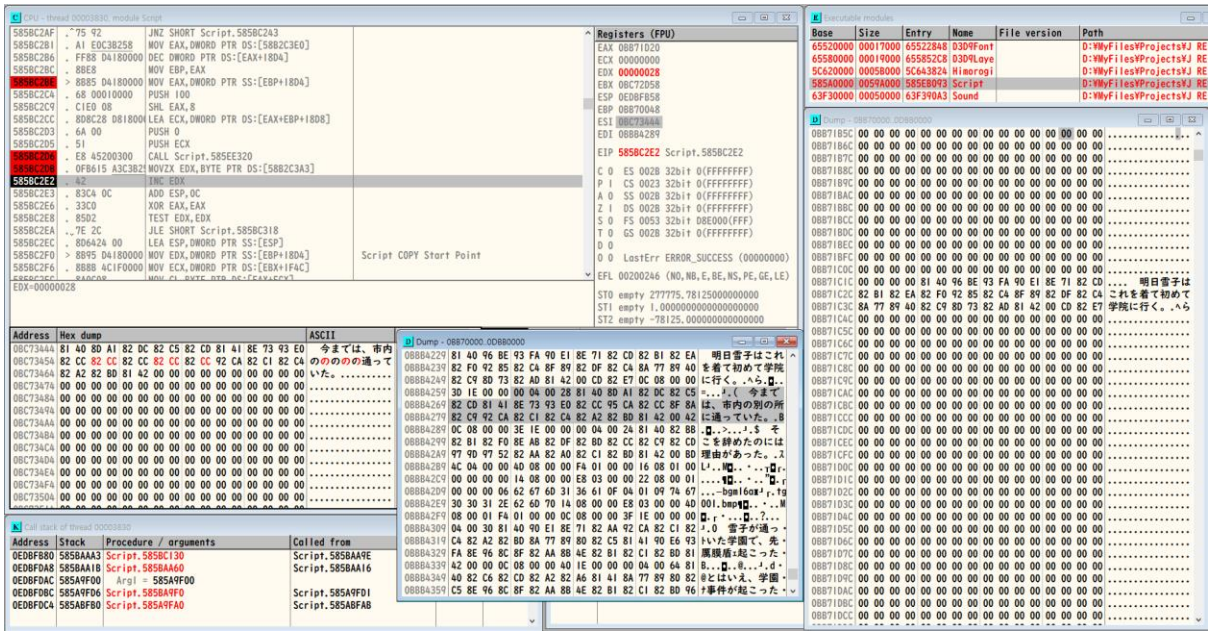
다음줄을 보니 역시나 다른 스크립트로 점프를 한다. 몇번 테스트 해보았는데 이 스크립트가 끝난 이후 뭐가 바뀌는지는 모르겠다.

이후의 코드에서 백로그가 기록되는것으로 보이는데 현재 ESI가 가리키는 메모리의 값을 바꾼다면 어떤 변화가 생길지 궁금해졌다.



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

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



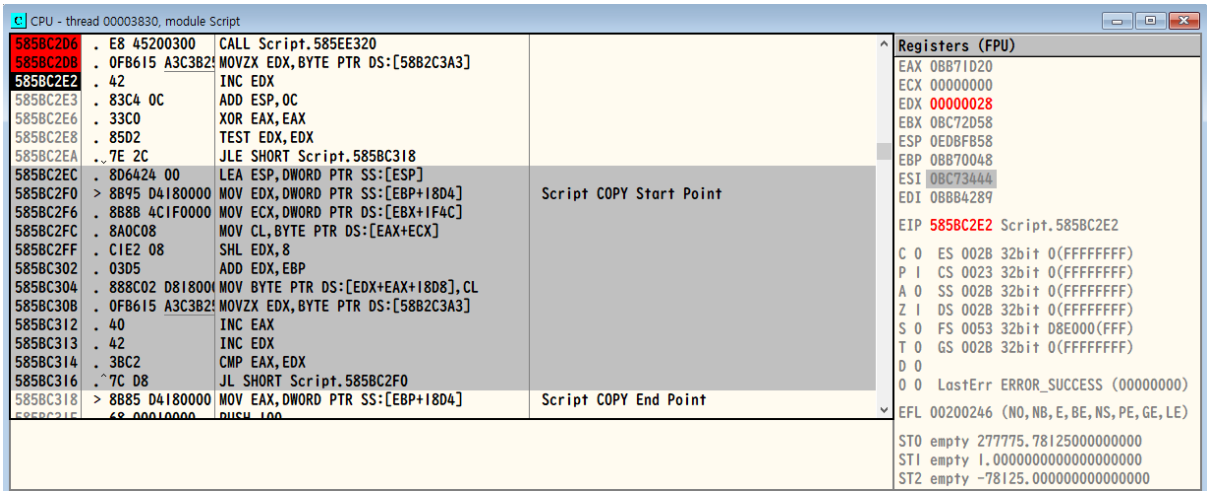
MOVZX부분을 실행하니 EDX에 현재스크립트길이가 들어왔다.

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

MOV EDX, ~~[EBP+18D4]

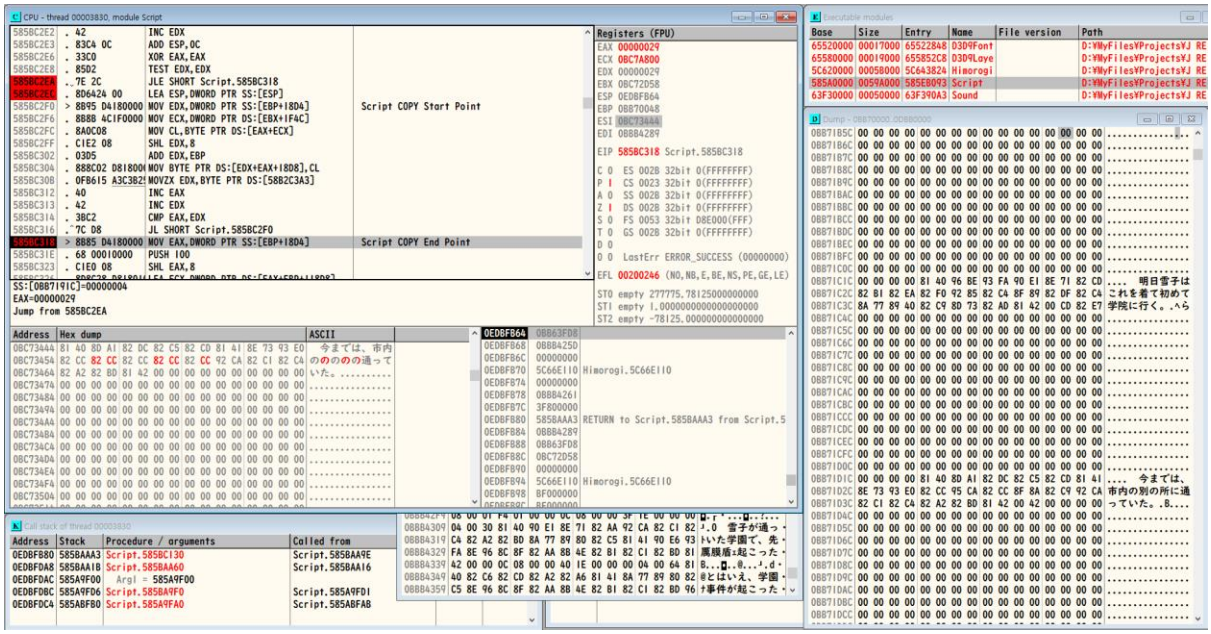
MOV ECX, ~~[EBX+1F4C]

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.

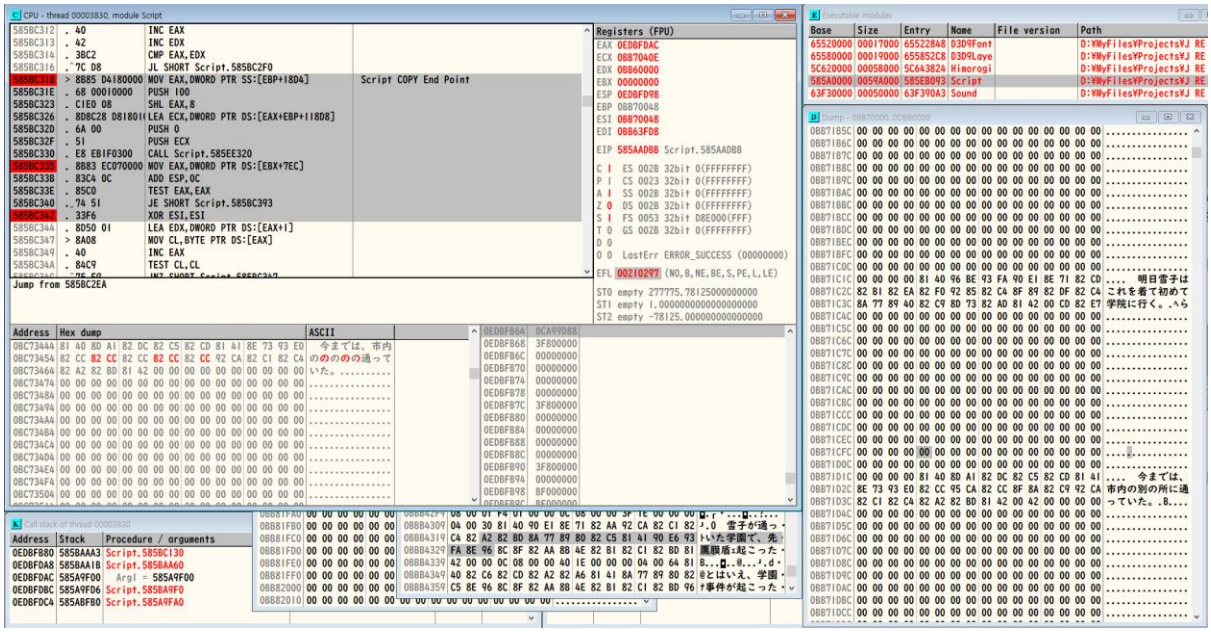
백로그 기록 시작지점 585BC2F0 – 585A0000 = 1C2F0



기록이 잘 되어있는 모습을 확인했다.

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

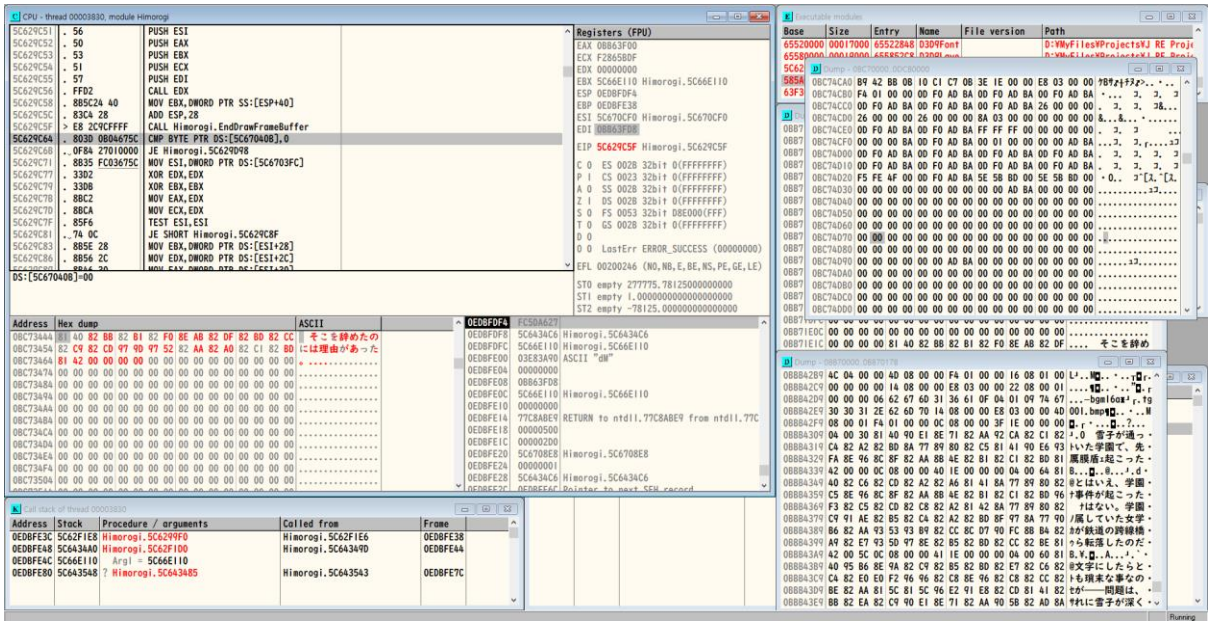
일단 백로그 기록 이후 코드에서 CALL문이 나오는 지점마다 바로 다음에 브포를 걸고, 스크립트 콜 이후에 어떤일이 일어나는지 알아보았다.



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.

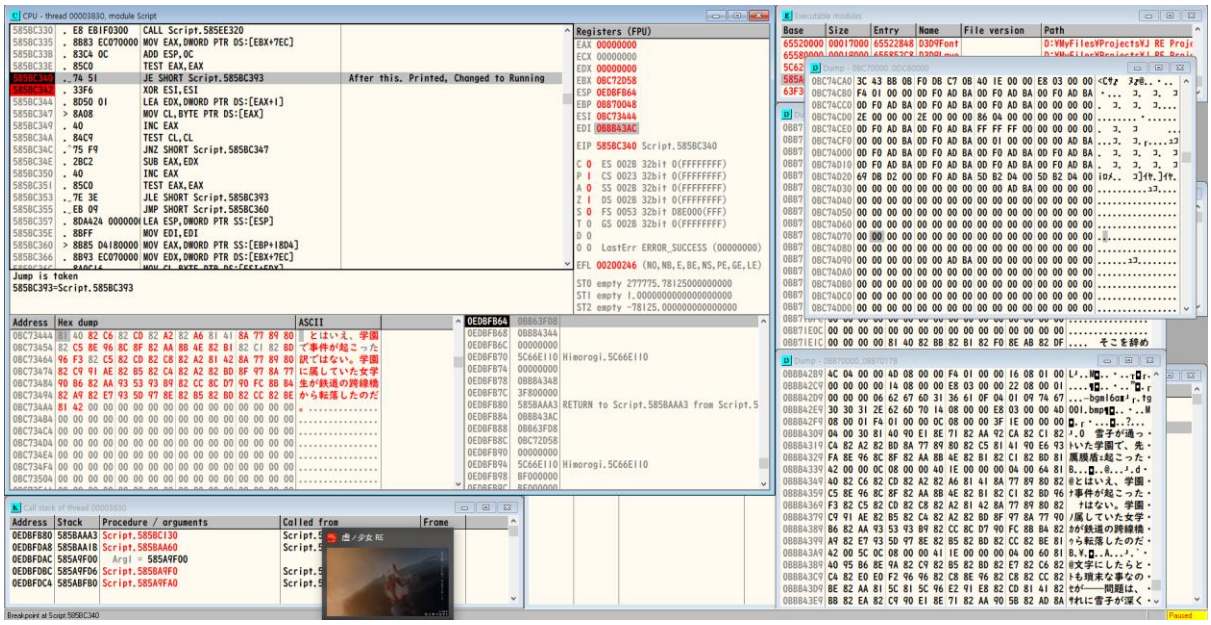
585BC340 (1C340) 이후 화면 출력완료, 프로그램 Running으로 바뀜.



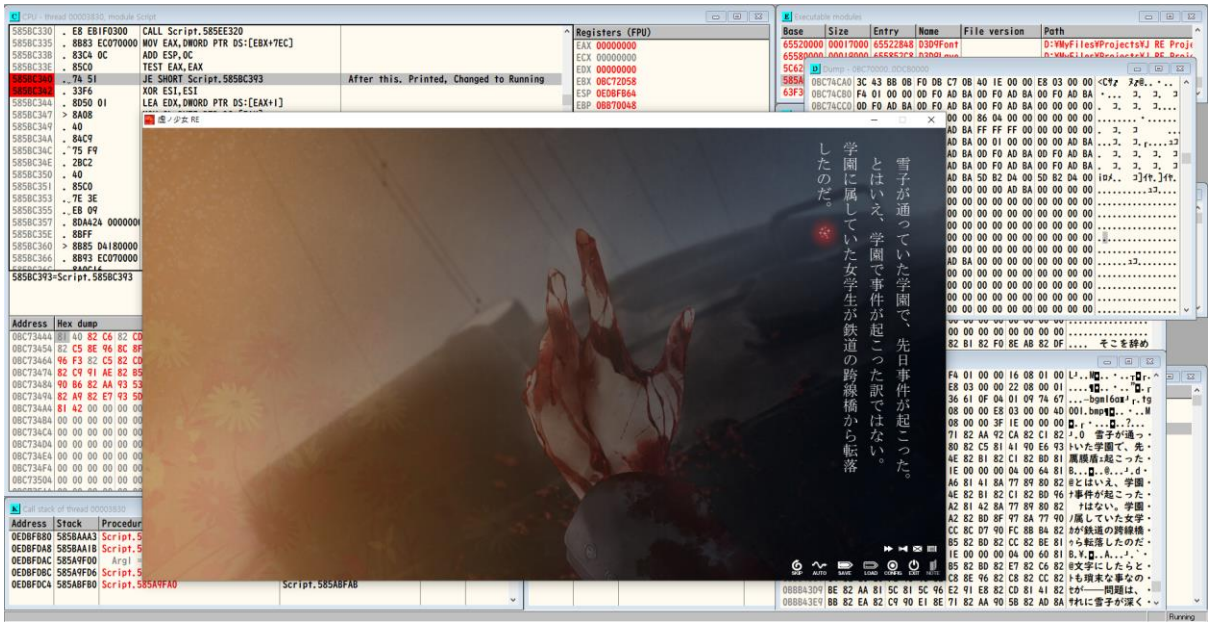
위 코드 이후를 테스트해보면서 도대체 이게 뭐냐..하면서 넘겼는데 Himorogi.dll 넘어가고 점프문에 정신을 못차리다가 어느순간 EndDrawFrameBuffer까지 도달했을때의 당혹감은 말로 표현할수 없다. 정말 몇번 넘긴게 없다. 100번은 고사하고 50 라인조차 지나지 않은것같은데 벌써 화면 출력인가..? 아무튼 이렇게 도달해버리면 Running으로 전환되면서 화면에는 대사가 다 출력되어있고 클릭 가능해진다.

다시 이전 상황까지 가보자.

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



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 깊숙한곳까지 들어온것이었다. 백로그 번역이 되는지 확인을 해보지 않았던건 왜지?? 화면출력 번역에 정신이 팔려서 그런것같은데.. 다음과 같은 사고과정이 있던것같다.

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...

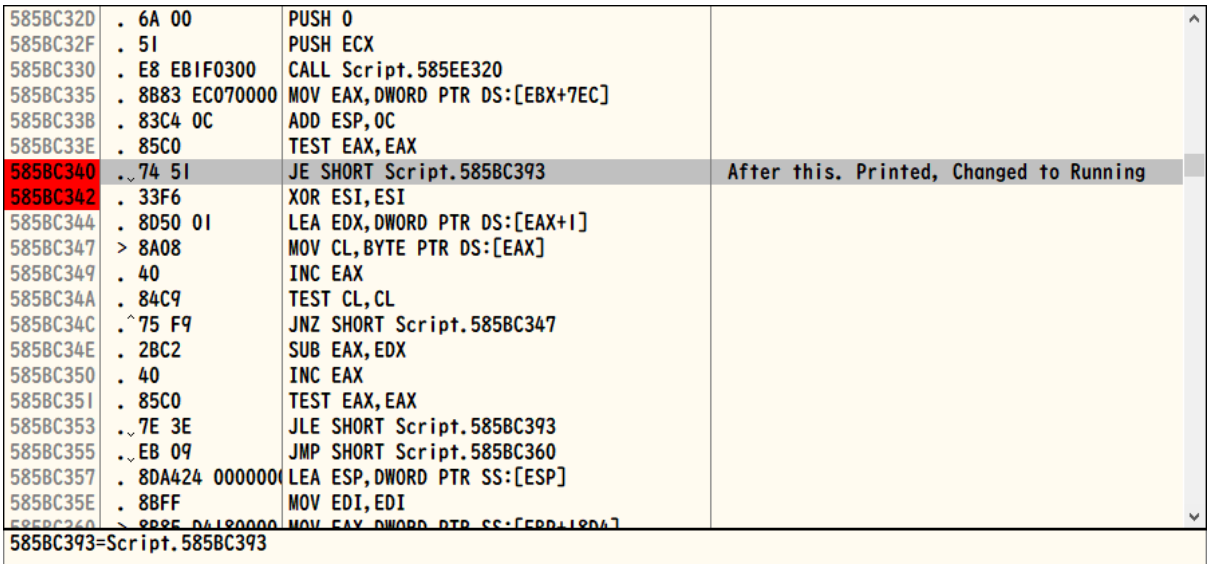
그럼 이번 기록은 여기서 마무리.

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 낌새를 발견했기 때문으로 두가지 이유를 들 수 있다.

<다음에 해야할 것 정리>

1. 이번 기록에서 찾은 이 지점



1C340 부근을 자세히 살펴보며 화면출력 번역에 한걸음 더 나아가야한다.

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