

첫페이지는 메모...

[illegible]

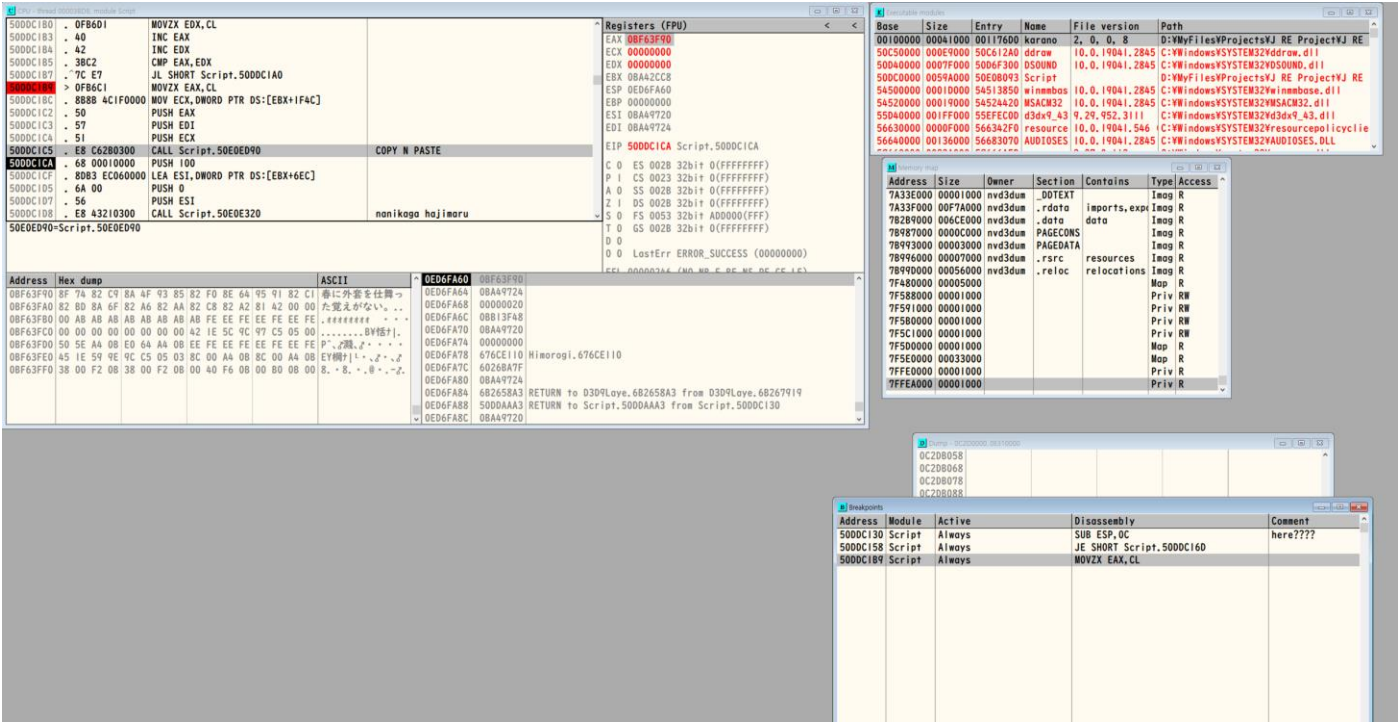
위 사진은 구버전..

[illegible]

EBX 붙여넣기하는곳

EDI 원본주소

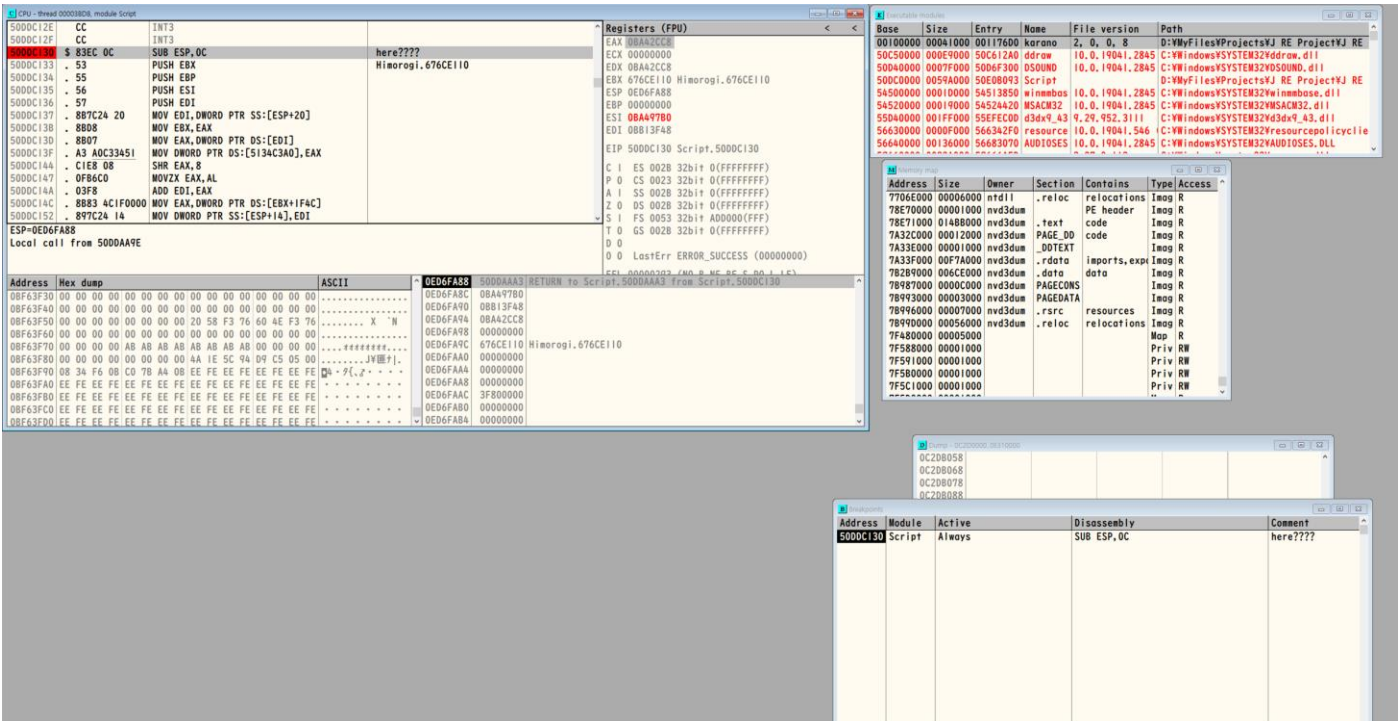
이렇게 테스트해보다가 갑자기 뭔가 떠올라서 기록 남기기 시작



드디어 문자열 copy and paste를 시작하는 최상위 지점을 찾은듯하다..

50DDC1C5 (0018C1C5) = 문자열 붙여넣는 스크립트 이동지점

이 위쪽으로 거슬러 올라가보면...



50DDC130 (0018C130) = 시작지점(?)

이곳이 시작점같다. 그렇다면 이곳 언저리에 후킹하면 되지 않을까 싶은데..

조금 테스트를 해보니 무언가 수상하다



이런식으로 스프라이트가 표시되는 화면에서 브포에 두번 걸린다.

좀더 테스트를 해보며 정확히 어느때 걸리는건지 확인해보자.

테스트 시작은 ??? なにかかしら 가 출력되어있는 상태인데 엄청 많이 보았으니 알겠지요?

테스트 시작 화면은 스프라이트 없고, 이름+대사 출력된 상태..



그리고 클릭하여 대사를 넘기면 위 상태가 되며 브포에 걸린다.



다시 F9를 눌러 진행하면 바로 대사가 출력되며 상태는 Running 이 된다.

→ 대사만 출력되는 상황에서는 브포 한번걸림.

그렇다면 스프라이트가 나오면 어찌될지 보기위해 계속 대사를 넘겨보자

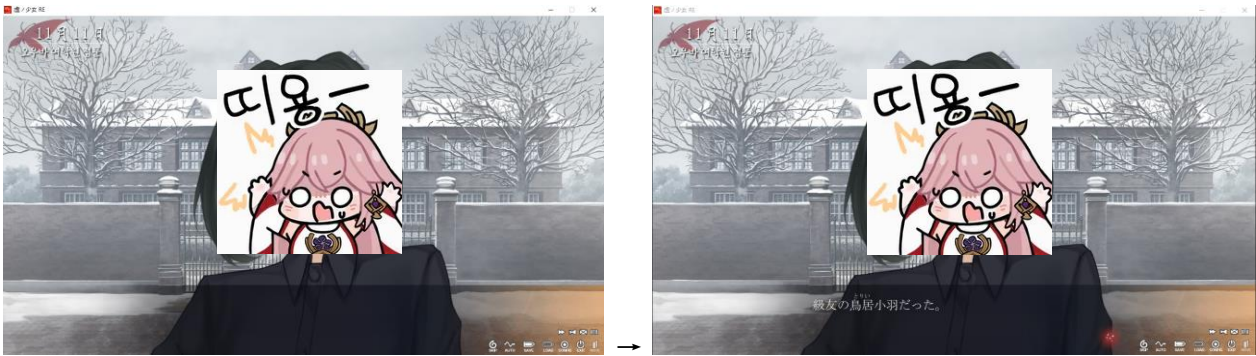
아래는 위 상태에서 클릭을 한번하여 브포에 걸린상태,

이 아래는 왼쪽에서 F9 한번 누른 것



또 F9를 누르니 이름+대사가 출력됨 (상태는 Running)

→ 아하! 스프라이트 출력시 한번, 이름+대사 출력시 한번이 확실하다.



참고로 스프라이트 표시는 이렇게 커다랗게 튀어나올때도 한번 걸린다..(집중해서 테스트해보다가 갑자기 튀어나와서 깜짝놀람;)

올리씨를 잠시 쉬게 돌려보내고 아칼씨를 불러오자.

라고 생각하며 올리디버거를 종료했는데 아이고 어떤 레지스터를 납치해야하는지 안보고 켜네~

다시 키기도 귀찮고 이미 켜오니 무지성으로 이걸가 싶은거 가져와봤는데 아무래도 이건 아닌거같다.

<여담 1> - 원본데이터구조

The screenshot displays a debugger interface with three main panes. The top-left pane shows assembly code with instructions like `SUB ESP, 0C` and `MOV EAX, 00000000`. The top-right pane shows the register state, with `EAX` at `00000000`. The bottom pane shows a memory dump with hex and ASCII values. A comment 'here???' is visible in the assembly pane.

올리씨를 켜서 레지스터를 확인하던데, 코드에 [5114C3A0]이라는 의문의 절대주소가 눈에 띈다.

EAX 를 그대로 옮겨갖다놓는데.. 주시해보자.

SHR EAX,8 MOVZX EAX,AL

3400 0400 -> 0034 0004 -> 0000 0004

그리고 EDI 에 EAX 를 더한다. 이때 EDI 는 PASTE 하는주소값인데, 그럼 1 WORD 만큼 EDI 가 오른것이다.

다시 정리해보면...

- EAX 는 절대주소[5114C3A0]에의해 3400 0400 으로 고정
- EAX 는 Shift Right 이후 AL 만 추출되어 0000 0004 로 변환
- 붙여넣기할 주소를 가리키는 EDI 에 EAX 가 더해져 다음 EDI 로 1WORD 를 추출해올수 있는 준비 완료

Address	Hex dump	ASCII
0BCC9649	97 E2 82 BD 82 AD 82 C8 82 C1 82 C4 82 A2 82 BD	冷たくなっていた
0BCC9659	81 42 00 00 0C 08 00 00 9D 1E 00 00 00 04 00 34	。...。...。...。4
0BCC9669	82 BB 82 EB 82 BB 82 EB 8C 5A 82 CC 8A 4F 93 85	そろそろ兄の外套
0BCC9679	82 E0 97 70 88 D3 82 B5 82 C4 82 A0 82 B0 82 E9	も用意してあげる
0BCC9689	82 D7 82 AB 82 A9 82 E0 82 B5 82 EA 82 C8 82 A2	べきかもしれない
0BCC9699	81 42 00 42 B4 04 00 0C 79 75 6B 66 30 37 30 36	。BI...yukf0706
0BCC96A9	2E 70 6E 67 14 08 00 00 FA 00 00 00 00 04 00 08	.png...。...。...
0BCC96B9	81 94 8E 87 00 1D 46 00 0C 08 00 00 9E 1E 00 00	#紫.F...。...。...
0BCC96C9	27 08 00 00 00 00 00 07 79 75 6B 30 30 30 32 00	'。.....yuk0002.
0BCC96D9	04 00 10 81 75 81 63 81 63 82 A0 82 C1 81 76 00	。+「.....あっ」.
0BCC96E9	E2 46 00 11 08 00 00 00 00 00 00 14 08 00 00 FA	礎。...。...。...

이제 이쪽을 한번 보자..

Hex dump 와 ASCII 를 보며 비교해보는거다.

대사, 스프라이트파일명, 인물명 등등 직전에 각 문자열의 길이가 나와있다.

<여담 2> - 구버전과의 비교

The screenshot displays a WinDBG session comparing two versions of a program. The left pane shows the current state, and the right pane shows a previous state. The assembly code is disassembled from a hex dump, and the registers (FPU) are shown on the right. The code includes instructions like LEA, MOV, INC, and CALL, along with comments in Japanese. The registers show values like EAX, ECX, EDI, and EIP. The right pane shows a list of executable modules and their properties.

String copy and paste 쿨 하는부분은 상대주소로 0004 C910 에 존재하나보다.

<여담 3> - string copy and paste 이전 상위 코드를 찾아보자.

The screenshot displays a WinDBG session showing assembly code and registers. The assembly code is disassembled from a hex dump, and the registers (FPU) are shown on the right. The code includes instructions like SUB, MOV, XOR, and CALL, along with comments in Japanese. The registers show values like EAX, ECX, EDI, and EIP. The right pane shows a list of executable modules and their properties.

이곳이 더 상위지점. 이곳을 브포 걸면 현재 화면을 지을 때도 걸리고 다음화면을 그릴때도 걸린다. 다음 코드파인딩기록은 이것을 중점적으로 살펴보는것으로 시작할듯하다. 이쪽이던가 후킹 지점을 잡으면 되는거 아닐까..?

저기 call susang point(수상한지점) 잡아놓은곳을 타고 가서 50BDC130(0001 C130)에 브포를 걸고 테스트하는건 이번 코드파인딩기록에서 테스트 해본것이다.

그럼 이제 다시한번 정리해보자.

CPU - thread 0000120C, module Script

50BDC12E CC INT3
50BDC12F CC INT3
50BDC130 \$ 83EC 0C SUB ESP, 0C susang point start
50BDC133 . 53 PUSH EBX
50BDC134 . 55 PUSH EBP
50BDC135 . 56 PUSH ESI
50BDC136 . 57 PUSH EDI
50BDC137 . 8B7C24 20 MOV EDI, DWORD PTR SS:[ESP+20]
50BDC138 . 8B08 MOV EBX, EAX
50BDC13D . 8B07 MOV EAX, DWORD PTR DS:[EDI]
50BDC13F . A3 A0C3A0 MOV DWORD PTR DS:[5114C3A0], EAX SAME from here
50BDC144 . C1E8 08 SHR EAX, 8
50BDC147 . 0FB6C0 MOVZX EAX, AL
50BDC14A . 03F8 ADD EDI, EAX
50BDC14C . 8B83 4C1F0000 MOV EAX, DWORD PTR DS:[EBX+1F4C]
50BDC152 . 897C24 14 MOV DWORD PTR SS:[ESP+14], EDI
50BDC156 . 85C0 TEST EAX, EAX
EAX=08000400
DS:[5114C3A0]=28000400

Registers (FPU)
EAX 08000400
ECX 00000000
EDX 08C2C2C8
EBX 08C2C2C8
ESP 0E9BF86C
EBP 00000000
ESI 08CC9A19
EDI 08CC9A19
EIP 50BDC13F Script.50BDC13F
C 0 ES 0028 32bit 0(FFFFFFFF)
P 0 CS 0023 32bit 0(FFFFFFFF)
A I SS 0028 32bit 0(FFFFFFFF)
Z 0 DS 0028 32bit 0(FFFFFFFF)
S 0 FS 0053 32bit 9A6000(FFF)
T 0 GS 0028 32bit 0(FFFFFFFF)
D 0
0 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000012 (NO, NR, NF, A, NE, DF, CF, C)

Address Hex dump ASCII
08CC998E 00 04 00 0C 81 94 81 48 81 48 81 48 00 E2 46 00 .!..#????.
08CC999E 0C 08 00 00 A9 1E 00 00 27 08 00 00 00 00 00 07 .
08CC99AE 68 6F 68 30 30 30 31 00 04 00 14 81 75 82 C8 82 koh0001.1.1「な
08CC99BE C9 82 AA 82 A9 82 B5 82 E7 81 48 81 76 00 48 0C /かかしら?」.H.
08CC99CE 08 00 00 AA 1E 00 00 00 04 00 28 99 EA 82 A2 82 .
08CC99DE BD C6 82 AB 81 A1 8C E3 82 EB 82 A9 82 E7 90 スと、後ろから
08CC99EE BA 82 F0 8A 7C 82 AF 82 E7 82 EA 82 BD 81 42 00 コを掛けられた。
08CC99FE 00 82 B5 84 04 00 0C 79 75 68 66 30 32 30 31 ZE .L.L'.yukf0201.
08CC9A0E 70 6E 67 14 08 00 00 FA 00 00 00 04 00 08 81 png
08CC9A1E 94 8E 87 00 1D 46 00 0C 08 00 00 AB 1E 00 00 27 博+
08CC9A2E 08 00 00 00 00 07 79 75 68 30 30 30 35 00 04 .+vuk0005.
0E9BF86C 08763F48
0E9BF870 08CC9A19
0E9BF874 00000000
0E9BF878 6B26E110 Himorogi.6B26E110
0E9BF87C 274F10BD
0E9BF880 FFFFFFFF
0E9BF884 6A9A58A3 RETURN to D3D9Laye.6A9A58A3 from D3D9Laye.6A9A7919
0E9BF888 50BDA0A3 RETURN to Script.50BDA0A3 from Script.50BDC130
0E9BF88C 08CC9A19
0E9BF890 08763F48
0E9BF894 08C2C2C8
0E9BF898 00000000

이렇게 타고와서 후킹하면 다음 스프라이트, (이름+)대사 출력할 때 한번씩 걸리는걸 이번 기록에서 확인할수 있었다.

구버전과 비교했을 때 SAME from here 지점부터 String copy and paste(아래쪽이라 사진에서 잘림)까지가 완전히 똑같은.

그렇다면 위 사진기준 50BDC130~50BDC13D 이 다른 부분이다.

참고로 SAME from here 은 구버전에서 후킹지점으로부터 어떤 작업이 끝나고 점프해오면 도착하는지점이었음.

CPU - thread 00000A94, module Script

6672DE62 > 2B0C SUB ECX, EAX
6672DE65 . 90 NOP
6672DE66 . 90 NOP
6672DE67 . 90 NOP
6672DE68 . 90 NOP
6672DE69 . 90 NOP
6672DE6A . 90 NOP
6672DE6B . 90 NOP
6672DE6C . 33C0 XOR EAX, EAX
6672DE6E > 3E:8070 00 00 CMP BYTE PTR DS:[EBP], 0
6672DE73 . 74 04 JE SHORT Script.6672DE79
6672DE75 . 40 INC EAX
6672DE76 . 45 INC EBP
6672DE77 . EB F5 JMP SHORT Script.6672DE6E
6672DE79 > 03C1 ADD EAX, ECX
6672DE7B > 0FB613 MOVZX EDX, BYTE PTR DS:[EBX]
6672DE7E . 3E:8B55 00 MOV BYTE PTR DS:[EBP], DL
6672DE82 . 45 INC EBP
6672DE83 . 43 INC EBX
6672DE84 . 49 DEC ECX
6672DE85 . 85C9 TEST ECX, ECX
6672DE87 . 75 F2 JNZ SHORT Script.6672DE7B
6672DE89 . 2BE8 SUB EBP, EAX
6672DE8B . 5B POP EBX
6672DE8C . 895D FC MOV DWORD PTR SS:[EBP-4], EBX
6672DE8F . 8B45 FF MOV BYTE PTR SS:[EBP-1], AL
6672DE92 . 59 POP ECX
6672DE93 . 5B POP EBX
6672DE94 . 5A POP EDX
6672DE95 . 83ED 04 SUB EBP, 4
6672DE98 . 8B45 00 MOV EAX, DWORD PTR SS:[EBP]
6672DE9B . E9 C81F8F JMP Script.666DC08C next step
6672DEA0 . 90 NOP
6672DEA1 . 90 NOP
6672DEA2 . 90 NOP
666DC08C-Script.666DC08C

Registers (FPU)
EAX 28000400
ECX 00000000
EDX 0A0F0000
EBX 0A0D2088
ESP 00B6F8CA
EBP 0A07E2CE
ESI 0A07E2CE
EDI 00000001
EIP 666DC08C Script.666DC08C
C 0 ES 0028 32bit 0(FFFFFF)
P I CS 0023 32bit 0(FFFFFF)
A I SS 0028 32bit 0(FFFFFF)
Z 0 DS 0028 32bit 0(FFFFFF)
S 0 FS 0053 32bit E1100001
T 0 GS 0028 32bit 0(FFFFFF)
D 0
0 0 LastErr ERROR_TOO_MANY
EFL 00200216 (NO, NB, NE, A, N
ST0 empty 89.022224A262695
ST1 empty 1.0000000000000000
ST2 empty -25.600000381469
ST3 empty 0.5000000000000000
ST4 empty -0.5000000000000000
ST5 empty 59.0000000000000000
ST6 empty 1.0000000000000000
ST7 empty 0.6123046875000000
FST 4020 Cond 1 0 0 0 Er
FCW 027F Prec NEAR, 53 Ma

Address Hex dump ASCII
00B6F8C0 88 2D AD 0A 00 00 AF 0A 00 00 00 88 2D AD 0A .
00B6F8C2 00 00 00 00 00 00 07 AF 6D 66 88 2D AD 0A .

Base Size Entry Name File version Path
00000000 00030000 00017678 Karano.s C:\KaranoProject\Karano_s\Karano\Karano_sld.exe
60300000 00017000 60302888 03D9Font C:\KaranoProject\Karano_s\Karano\plugin\03D9Font.dll
60800000 00097000 608F3AC8 03D9Laye C:\KaranoProject\Karano_s\Karano\plugin\03D9Laye.dll
68100000 000A7000 6813E0E8 Himorogi C:\KaranoProject\Karano_s\Karano\plugin\Himorogi.dll
68600000 00059700 68708081 Script C:\KaranoProject\Karano_s\Karano\plugin\Script.dll
68030000 00050000 6803C822 Sound C:\KaranoProject\Karano_s\Karano\plugin\Sound.dll

-> 구버전 (빨강게 브포걸어둔곳이 후킹지점, EBX 를 납치하는곳..)

그렇다면 추가된 50BDC130~50BDC13D 코드를 좀 자세히 봐야겠다..

〈여담 4〉

CPU - thread 0000320C, module Script

Address	Hex dump	ASCII
50BDA660	81EC 10020000	SUB ESP,210
50BDA666	A1 680CC350	MOV EAX,DWORD PTR DS:[50C30C68]
50BDA66B	33CA	XOR EAX,ESP
50BDA66D	898424 0C0200	MOV DWORD PTR SS:[ESP+20C],EAX
50BDA674	0FB68424 1402	MOVZX EAX,BYTE PTR SS:[ESP+214]
50BDA67C	53	PUSH EBX
50BDA67D	55	PUSH EBP
50BDA67E	56	PUSH ESI
50BDA67F	57	PUSH EDI
50BDA680	8BF1	MOV ESI,ECX
50BDA682	3D B8000000	CMP EAX,0B8
50BDA687	0F87 28080000	JAE Script.50BD02B5
50BDA68D	0F688 8484B0	MOVZX ECX,BYTE PTR DS:[EAX+50BD8484]
50BDA694	FF2480 00B380	JMP DWORD PTR DS:[ECX*4+50BD8300]
50BDA69B	> 56	PUSH ESI
50BDA69C	88C2	MOV EAX,EDX
50BDA69E	E8 8D160000	CALL Script.50BD0C130

ESP=0E9BFD80
Local calls from 50BDA616, 50BDA643

Registers (FPU)

Register	Value
EAX	00000012
ECX	08CC9B08
EDX	08CC2C88
EBX	6826E110 Himorogi.6826E110
EIP	0E9BFD80
EBP	00000000
ESI	08CC2C88
EDI	08763F48
EIP	50BDA6A0 Script.50BDA6A0
C 0	ES 0028 32bit 0(FFFFFFFF)
P 1	CS 0023 32bit 0(FFFFFFFF)
A 0	SS 0028 32bit 0(FFFFFFFF)
Z 1	DS 0028 32bit 0(FFFFFFFF)
S 0	FS 0053 32bit 9A6000(FFF)
T 0	GS 0028 32bit 0(FFFFFFFF)
D 0	
D 0	LastErr ERROR_SUCCESS (00000000)
EFL	00000014 (OF 0F 0F 0F 0F 0F 0F 0F)

Executable modules

Base	Size	Entry	Name
00100000	00041000	00117600	karano
509E0000	00063000	50A2FEA0	D3D1W700
50A50000	000E9000	50A612A0	ddraw
50840000	0007F000	5086F300	DSOUND
508C0000	0059A000	50C0B093	Script
51160000	001FF000	5131EC00	d3dx9.4.3
54A50000	0001D000	54513850	winnmbas
54520000	00019000	54524A20	NSACM32
55D10000	00005000	55D190A3	Sound

Memory map

Address	Size	Owner	Section
00100000	00001000	karano	
00101000	0002A000	karano	.text
00128000	00007000	karano	.rdata
00132000	00004000	karano	.data
00136000	00008000	karano	.rsrc
0013E000	00003000	karano	.reloc
00690000	00010000		
006A0000	00001000		
006B0000	00001000		
006C0000	0001D000		
0070E000	00012000		
00720000	00004000		
00730000	00001000		
00740000	00002000		
00785000	00008000		
00790000	00009000		

Script.50BDA6A0

Address	Hex dump	ASCII
08CC998E	00 04 00 0C 81 94 81 48 81 48 81 48 00 E2 46 00	. . . # ? ? ? . 確 .
08CC999E	0C 08 00 00 A9 1E 00 00 27 08 00 00 00 00 00 07
08CC99AE	68 6F 68 30 30 A3 1 31 00 04 14 81 75 82 C8 08	koh0001.
08CC998E	C8 02 AA 82 A9 82 85 82 E7 81 48 81 76 00 48 0C	/かかしら?
08CC99CE	08 00 00 AA 1E 00 00 00 04 00 28 99 EA 82 A2 82
08CC99DE	80 82 C6 82 AB 81 41 8C 83 82 EB 82 A9 82 E7 90
08CC99EE	84 C2 F0 8A 7C 82 AF 82 E7 82 EA 82 B0 81 42 00
08CC99FE	00 82 85 84 00 00 0C 79 75 68 66 30 32 30 31 2E
08CC9A0E	70 6E 67 14 08 00 00 FA 00 00 00 00 04 00 08 81
08CC9A1E	94 8E 87 00 10 46 00 0C 08 00 00 A8 1E 00 00 27
08CC9A2E	08 00 00 00 00 00 79 75 68 30 30 35 00 04 00

이렇게 브포를 걸면 다음 대사로 넘어갈 때 스프라이트와 대사를 지우는 동작을 하며 걸린다.

참고로 지을 때는 위에서 찾은 수상한 지점이 브레이크에 걸리지 않고 화면의 스프라이트와 대사가 지워지고,

새로운 스프라이트와 대사를 출력할 때 그제야 비로소 수상한 지점의 콜이 호출되어 브레이크가 걸리게 된다.

그렇다면 확실한건 위의 두 브포 사이는 화면 지우기를 담당하고, 화면 출력은 아래쪽의 콜을 타고 간 이후에 작동한다는 것이다!

다음 코드 파인딩 기록은 이곳을 중점적으로 바라보게 될 듯하다.