[2022-RE]-[2021350027]-[강재현]

[1] Execute the target PE file properly. How did you do that?

파일을 실행하면 “Do you have a valid name?”이라는 메시지박스가 뜬다. 관련 API도 파일의 경로와 이름에 대한 것이므로, 실행파일의 이름과 관련한 문제인 것 같다.

[1]번을 해결하면 “I know you are familiar with PE file format”이라는 메시지박스가 떠야 한다.

먼저 86비트 파일을 IDA로 열어봤다.

1. 간단한 코드 개요 확인하기

Imports view에서 해당 API들을 찾았다.

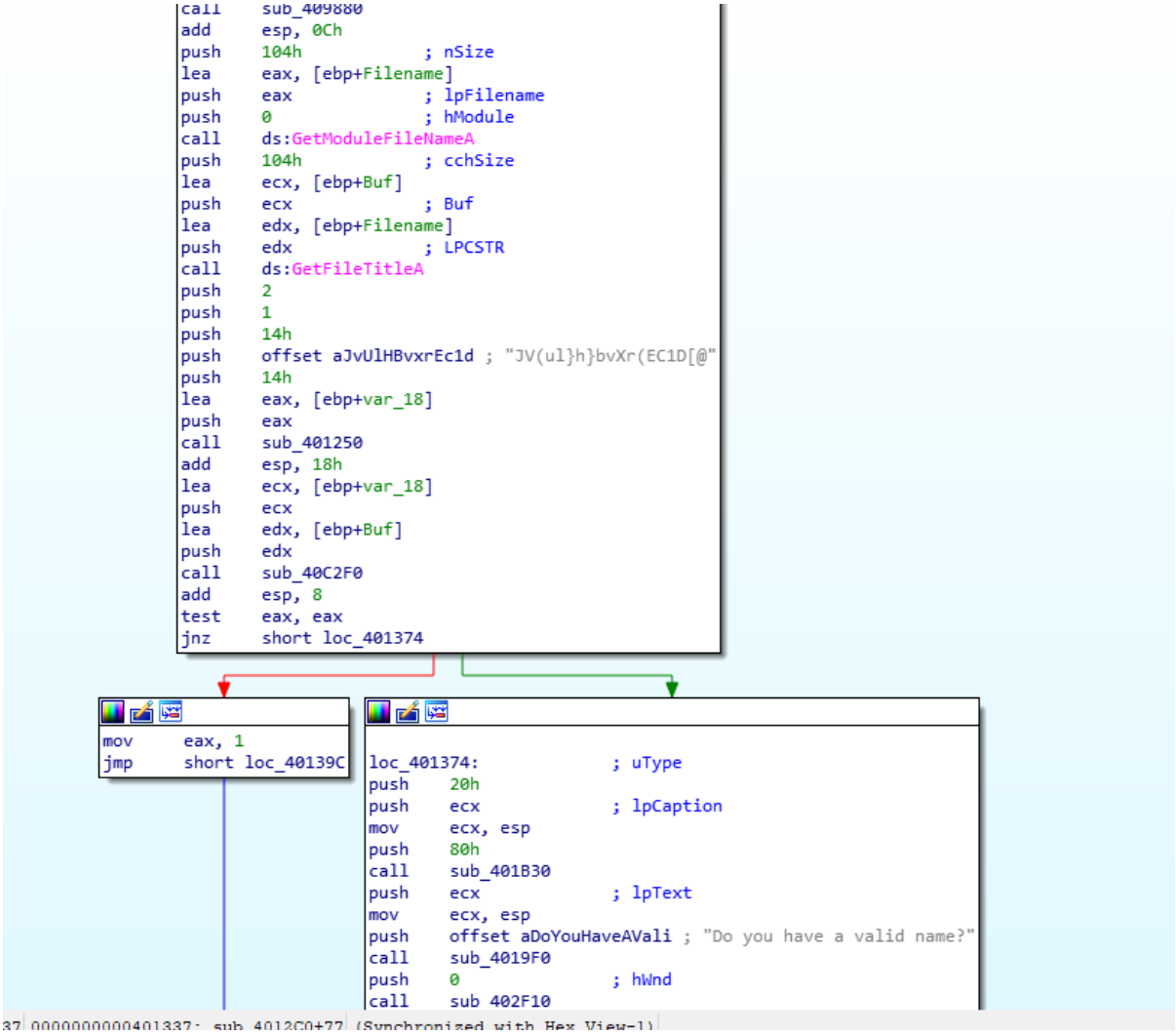
Address	Ordinal	Name	Library
0000000...		GetFileTitleA	COMDLG32
0000000...		GetModuleFileNameA	KERNEL32
0000000...		GetModuleFileNameW	KERNEL32

idata 영역에서의 해당 API들이다.

```
.idata:0041A02C ; __int16 __stdcall GetFileTitleA(LPCSTR, LPSTR Buf, WORD cchSize)
.idata:0041A02C                 extrn GetFileTitleA:dword
.idata:0041A02C                                         ; CODE XREF: sub_4012C0+77↑p
.idata:0041A02C                                         ; DATA XREF: sub_4012C0+77↑r ...
.idata:0041A030
.idata:0041A034 ;
.idata:0041A034 ; Imports from KERNEL32.dll
.idata:0041A034 ;
.idata:0041A034 ; DWORD __stdcall GetModuleFileNameA(HMODULE hModule, LPSTR lpFilename, DWORD nSize)
.idata:0041A034                 extrn GetModuleFileNameA:dword
.idata:0041A034                                         ; CODE XREF: sub_4012C0+5E↑p
.idata:0041A034                                         ; sub_407D50+9A↑p
.idata:0041A034                                         ; DATA XREF: ...
.idata:0041A038 ; DWORD __stdcall GetModuleFileNameW(HMODULE hModule, LPWSTR lpFilename, DWORD nSize)
.idata:0041A038                 extrn GetModuleFileNameW:dword
.idata:0041A038                                         ; CODE XREF: sub_4013B0+37↑p
.idata:0041A038                                         ; sub_4066D0+10E↑p ...
```

GetFileTitleA의 CODE XREF를 클릭한다.

- `sub_4012C0`의 그래프가 나온다.



sub_4012c0

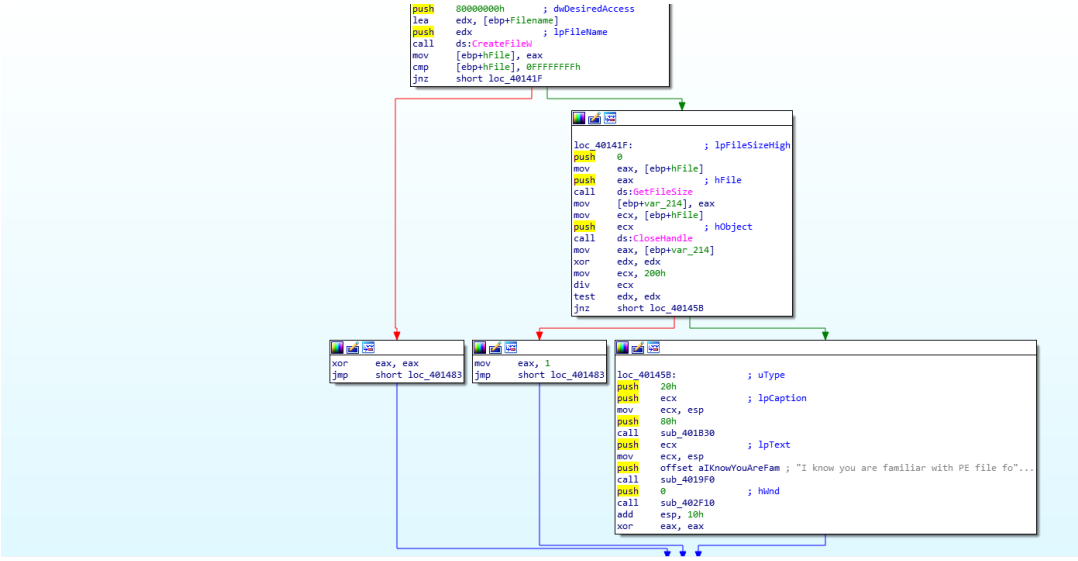
현재 **GetModuleFileName**의 `nSize: 0x0104` , `lpFilename: [ebp+Filename]` , `hModule: 0`

현재 **GetFileTitle**의 `cchSize: 0x0104` , `Buf: [ebp+Buf]` , `LPCSTR: [ebp+Filename]`

두개의 함수의 인자인 사이즈 모두 적당하게 잘 배치되었고, 인자에서 문제가 되는 부분은 없는 것 같다.

2. 구체적인 실행 로드 생각해보기

- “Do you have a valid name?”이라는 메시지박스가 실행되지 않았으면 좋겠다. 그러려면 JNZ가 실행되지 않아야한다. 그러면 점프를 안하면 코드는 어떤식으로 실행이 될까?
- 다음 함수 sub_4013B0으로 넘어가게되며, 아래와같이 내가 원하는 문장인 “I know you are familiar with PE file format”이 보인다.



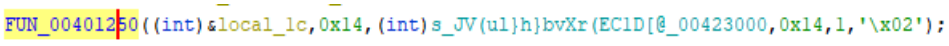
sub_4013b0

3. 디컴파일해서 확인하기

`sub_4012C0`를 디컴파일하고싶지만, IDA로는 되지 않아서 Ghidra를 사용할 것이다.

sub_4012c0

- 첫번째 메시지박스가 나오는 부분이 if문에 있다는 것을 알게 되었다. if문이 실행되지 않으려면 `iVar1=0` 이여야 한다.
- `__strcmp(local_224, (char*)&local_1c)=0` 이어야 한다. strcmp는 두개의 문자열을 비교해서 같으면 0을 return 하는 함수이다.
- `local_224` 는 **GetModuleFileName**에서 받은 파일의 이름이고, `&local_1c` 는 이제 찾아보려 한다.



Decompile: FUN_00401250 - (KU-reverseMe1^^.exe)

```
1
2  undefined4 __cdecl
3  FUN_00401250(int param_1,uint param_2,int param_3,uint param_4,byte param_5,char param_6)
4
5  {
6    undefined4 uVar1;
7    uint local_8;
8
9    if (param_2 == param_4) {
10     for (local_8 = 0; local_8 < param_2; local_8 = local_8 + 1) {
11       *(byte *)(param_1 + local_8) =
12            *(byte *)(param_3 + local_8) ^ param_5 ^ param_6 * (char)local_8;
13     }
14     if (param_2 != 0) {
15       *(undefined *)(param_1 + param_2 + -1) = 0;
16     }
17     uVar1 = 0;
18   }
19   else {
20     uVar1 = 0xffffffff;
21   }
22   return uVar1;
23 }
```

sub_401250

```
00423000  4A 56 28 75 6C 7D 68 7D  62 76 58 72 28 45 43 31  JV(ul}h}bvXr(EC1
00423010  44 5B 40 00 58 11 60 11  60 11 55 11 3E 11 8A 11  D[@.X.`.`.U.>.Š.
```

- 디컴파일이 된 대로 연산을 해보면 답이 나오지 않는다. 그래서 명령어부분을 직접 해석해보니 연산의 순서가 디컴파일된 것과 달랐다… 고생을 좀 했지만 제대로된 연산식을 알아냈다.
- 11번째 줄의 연산을 정리해보면, `local_lc[local_8] = (00432000[local_8] ^ 0x1) ^ (0x2 * local_8)` 이 된다.

4. `local_lc` 계산하기

1. `00423000[local_8] ^ 0x1`

   연산 결과:

   ```
   4b 57 29 74 6d 7c 69 7c  63 77 59 73 29 44 42 30
   45 5a 41
   ```

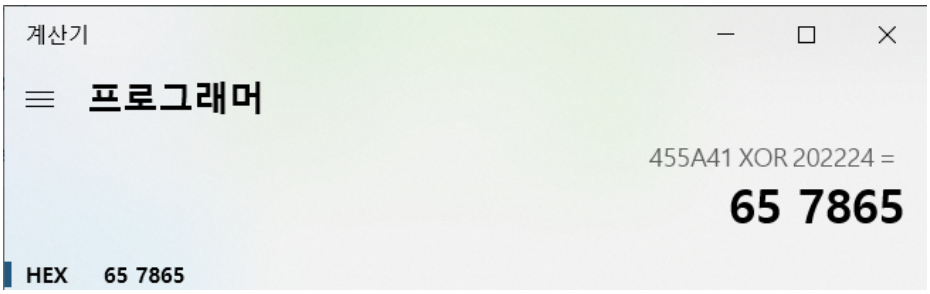
2. `0x2*local_8`

   연산 결과:

   ```
   00 02 04 06 08 0a 0c 0e  10 12 14 16 18 1a 1c 1e
   20 22 24
   ```

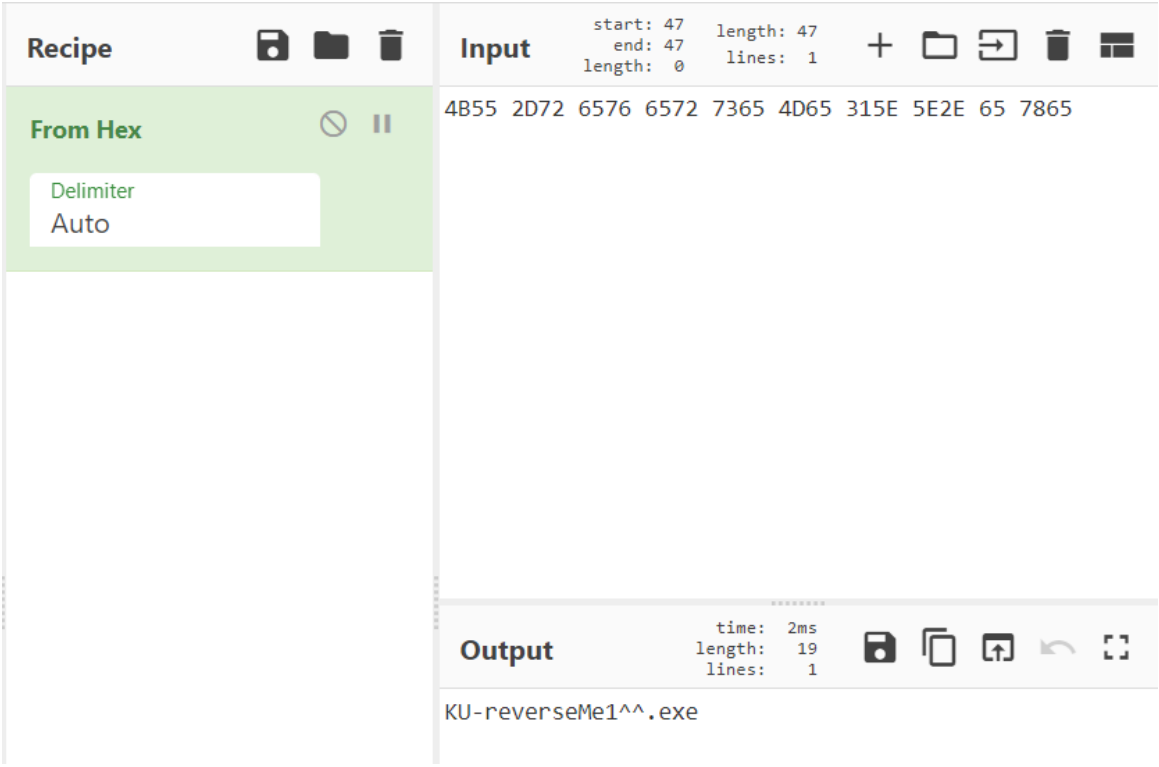
3. `00423000[local_8] ^ 0x1` ^ `0x2*local_8`

   연산결과:




```
4B55 2D72 6576 6572 7365 4D65 315E 5E2E
65 7865
```

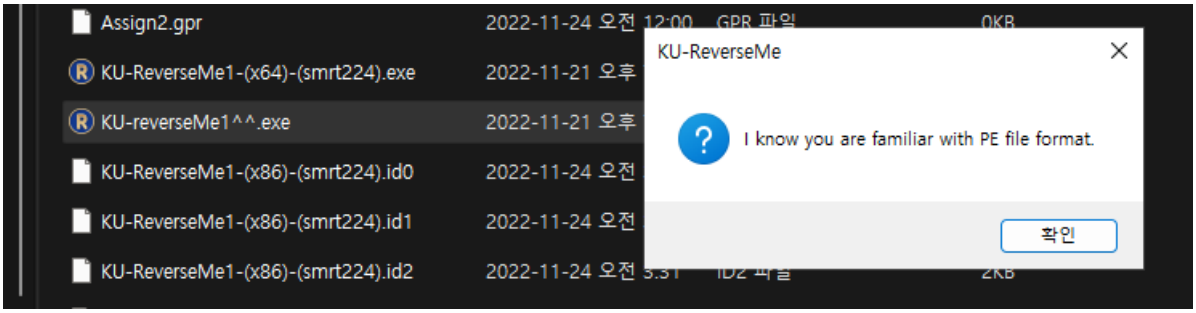
4. 아스키값으로 변환



- 교수님이 파일 내부를 건들 생각은 말라고 하셔서 둘의 strcmp 결과가 0이 되게하려면 파일의 이름을 바꿔야 한다.
- 따라서 파일의 이름을 `KU-reverseMe1^^.exe`로 바꿔보겠다.

5. 결과

- 파일의 이름을 `KU-reverseMe1^^.exe`로 바꾼다.



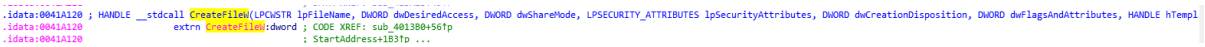
[2] Execute the target PE file properly. How did you do that?

다음 step으로 넘어가려면 입력창이 띄워져야한다.

먼저 입력창의 문자열들인 KU-ReverseMe1, Name, Student ID, Flag 등을 Strings view에서 검색해보겠다.

1. function 찾기

idata → `CreateFileW`



- `CODE XREF: sub_4013B0+56p`

idata→ `GetFileSize`

```
.idata:0041A104 ; DWORD __stdcall GetFileSize(HANDLE hFile, LPDWORD lpFileSizeHigh)
.idata:0041A104                 extrn GetFileSize:dword ; CODE XREF: sub_4013B0+78↑p
.idata:0041A104                                         ; DATA XREF: sub_4013B0+78↑r
```

- `CODE XREF: sub_4013B0+78p`

idata→ `GetModuleFileNameW`

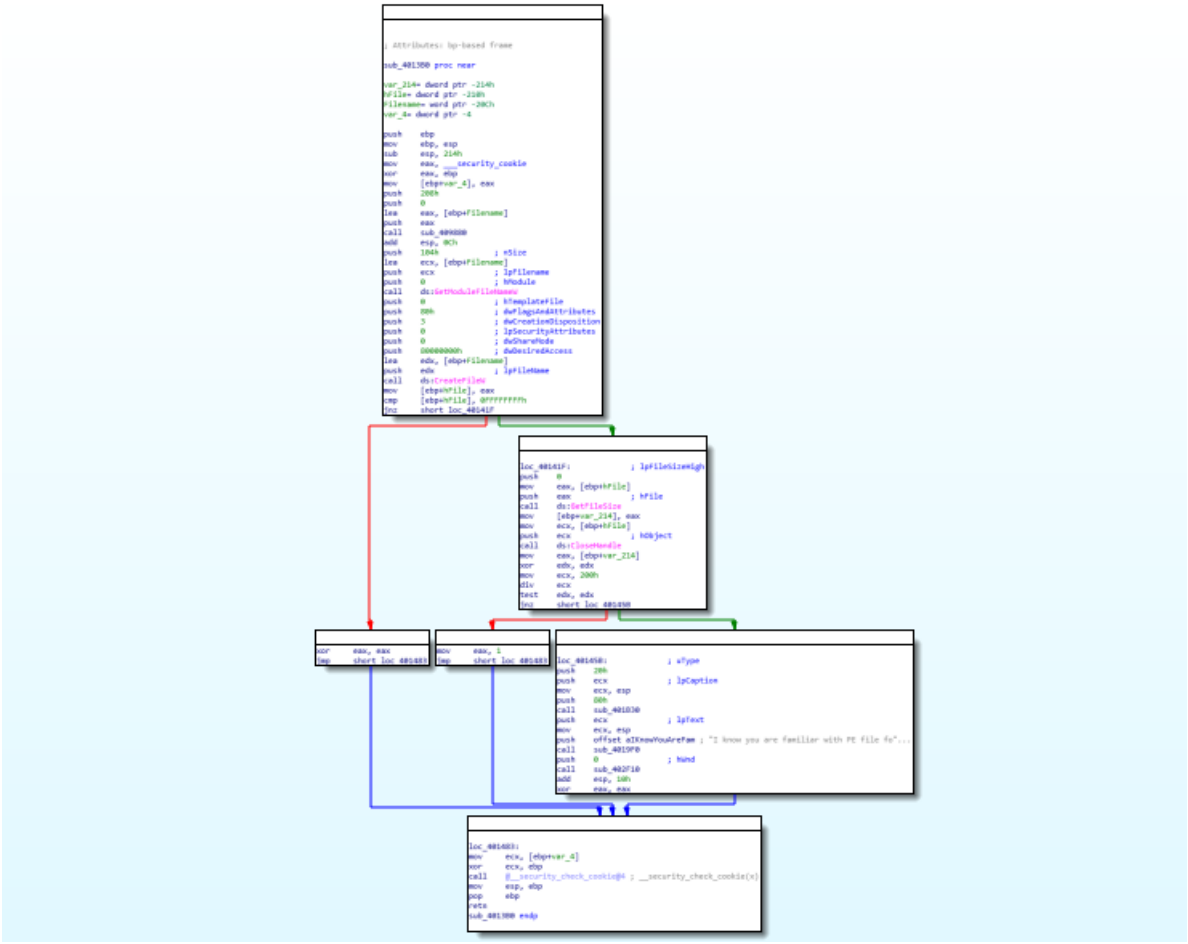
```
.idata:0041A034 |               extrn GetModuleFileNameA:dword
.idata:0041A034                                         ; CODE XREF: sub_4012C0+5E↑p
.idata:0041A034                                         ; sub_407D50+9A↑p
.idata:0041A034                                         ; DATA XREF: ...
.idata:0041A038 ; DWORD __stdcall GetModuleFileNameW(HMODULE hModule, LPWSTR lpFilename, DWORD nSize)
.idata:0041A038                 extrn GetModuleFileNameW:dword
.idata:0041A038                                         ; CODE XREF: sub_4013B0+37↑p
.idata:0041A038                                         ; sub_4066D0+10E↑p ...
```

- A와 W로 두종류가 있는데 위 둘과 같은 funcion이려면

  아마도 GetModuleFileNameW인 `CODE XREF: sub_4013B0+9Ap`로 잡는게 좋겠다.

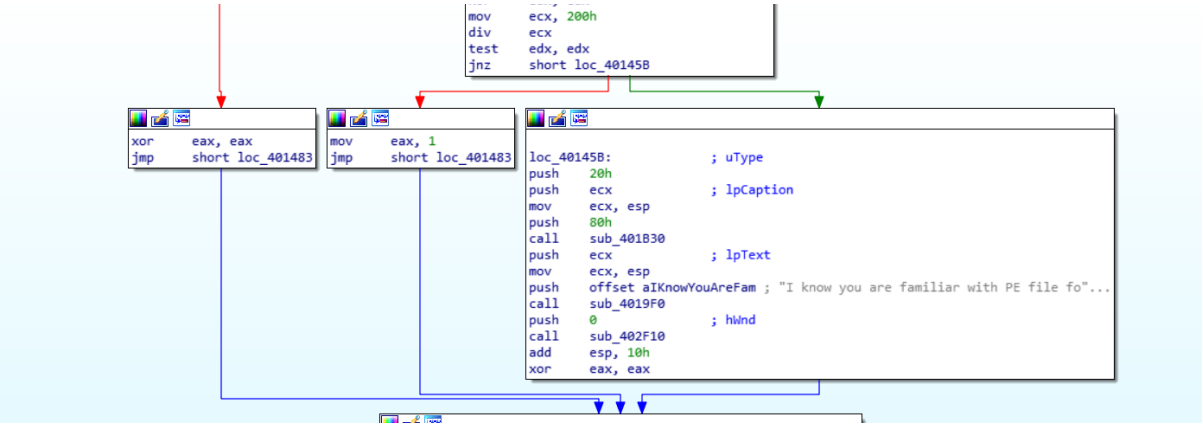
2. finction의 개요 보기

세 API가 이용되는 function인 `sub_4013B0`을 IDA View로 보겠다.



sub_4013b0

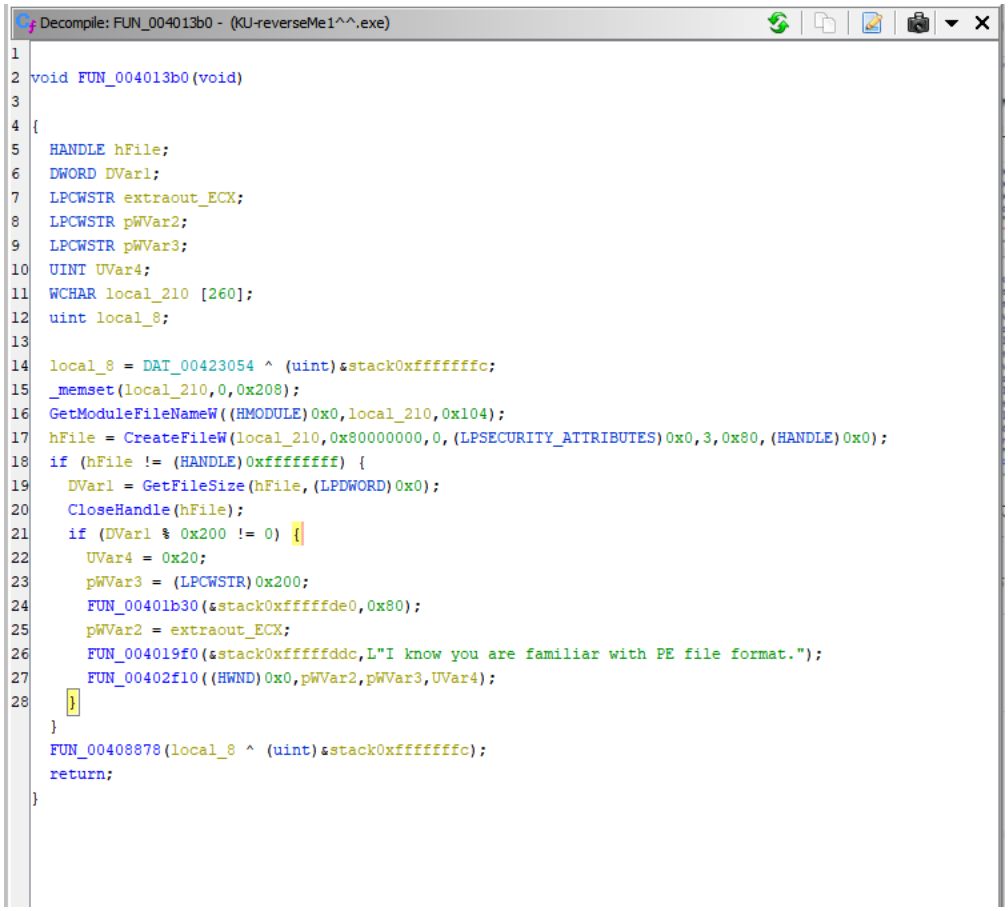
확대해서 보면 이 함수 내에는 아래와같은 부분이 있다.



sub_4013b0의 부분

- 문자열이 두번째 메시지박스와 관련된 것으로 보아... 아마도 이 메시지박스가 실행이 되지 않으면 다음 step으로 넘어갈 수 있을 것 같다.
- 그러면 또 ghidra로 이 함수를 디컴파일해서 제대로 파악해보겠다.

3. 디컴파일



sub_4013b0

- 이번에도 1번과 비슷하게 if문 안에 해당 문자열이 들어가있다.

1. `hFile = (HANDLE)0xffffffff`
2. `DVar1 % 0x200 = 0`

- 둘 중 하나라도 해당하면 다음 스텝으로 넘어갈수있을것같다.

  맨 처음 **GetModuleFileNameW**가 실행

  둘째로 **CreateFileW**가 실행된다.

  세번째 **GetFileSize**도 실행되려면, **2번째 조건인** `DVar1 % 0x200 = 0` **이 충족되어야 할 것 같다.**

4. DVar1이 200의 배수가 되도록 설계

- 이제 `DVar1`이 어떤 변수인지 파악해야한다.

  디컴파일한 코드의 19번째 줄을 보면 `DVar1 = GetFileSize(hFile, (LPWORD)0x0);`이다.

  `hFile`은 **CreateFileW**로 만들어진 파일이므로 `DVar1`은 **CreateFile**로 만들어진 **파일의 크기**일 것이다.

- 처음에는 CreateFileW의 Create에 꽂혀서 파일을 무조건 생성하는 줄 알았다.(이 부분에서 애를 먹었다…ㅠㅠ)

  하지만 다시 코드를 자세히 살펴보니 첫번째 인자인 `lpFileName`이 파일을 생성하는 것뿐만이 아니라 원래 파일을 여는 기능도 있었다.

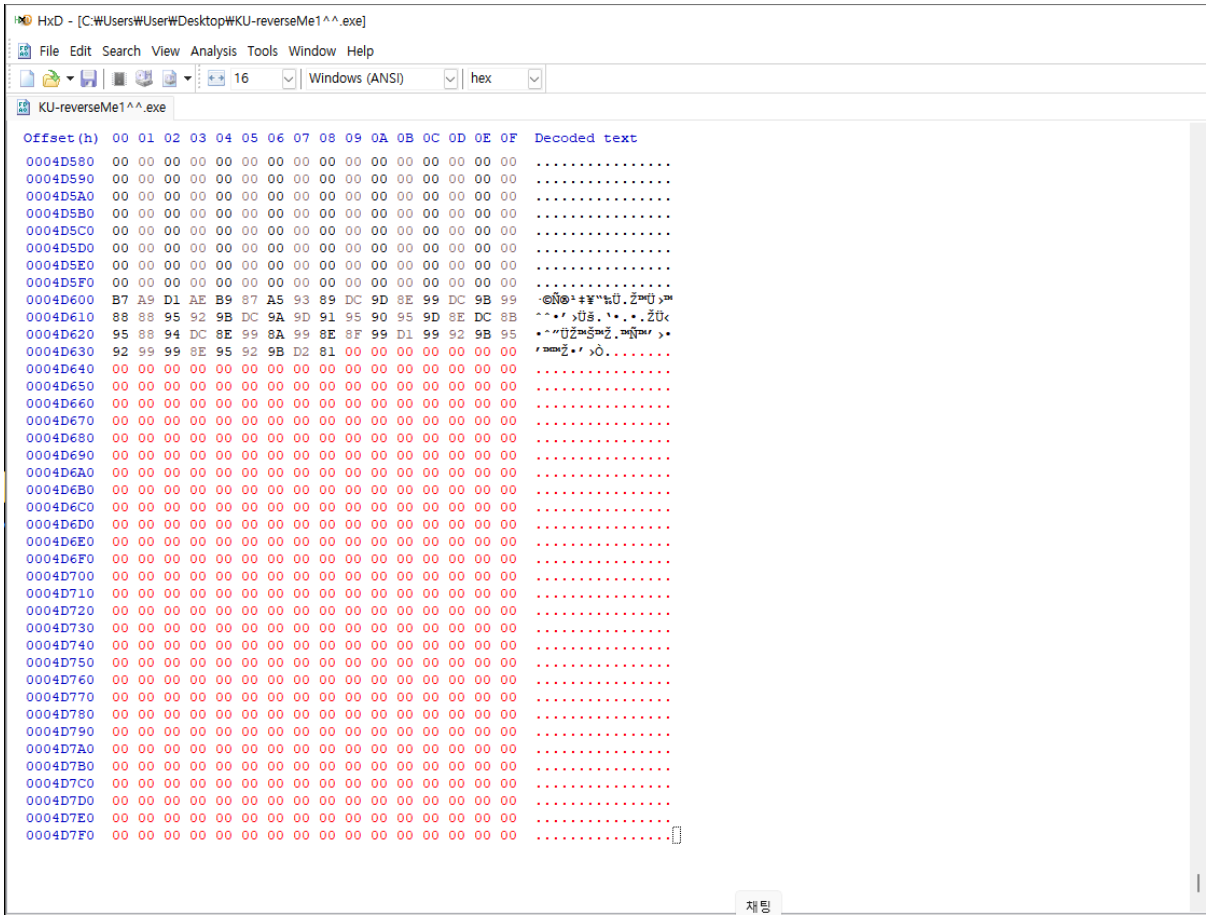
  그래서 다시 첫번째 인자부분을 자세히보니.. **GetModuleFileNameW**에서 받아온 파일의 이름인 `local_210`이 열 파임의 이름인 것을 알 수 있었다.

- 결론적으로는 이 KU-reverseMe1^^.exe 파일 자체의 사이즈를 가져오는것이다. 따라서 이 파일의 속성에서 용량을 확인해보니 크기가 319,488바이트였다. 따라서 이 실행파일의 사이즈가 200바이트의 배수가 아니라서 다음 스텝으로 못 나가는 것인가하는 합리적인 의심을 할 수 있었다.

- 따라서 Hex Editor를 이용해 사이즈를 바꿔보려 한다.

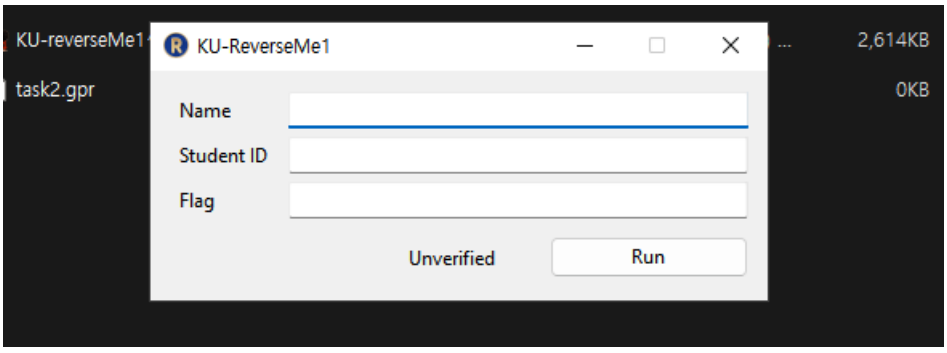
5. 실행파일의 사이즈가 200의 배수가 되도록 패딩을 추가

- 따라서 HxD로 파일의 사이즈가 200바이트의 배수가 되게 0으로 뒷부분을 패딩해줬다.



6. 결과

- 반신반의…하면서 도전을 했지만 성공했다.
- **파일의 사이즈가 200의 배수가 되도록 패딩을 해준다.**

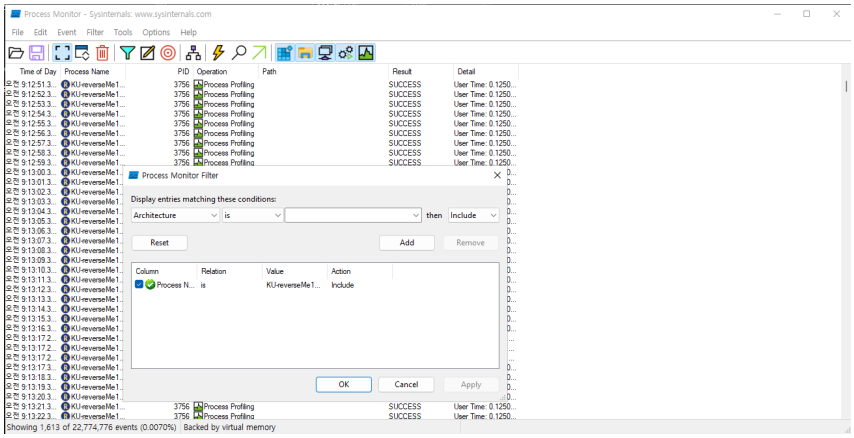


[3] What is the full path of a file automatically generated?

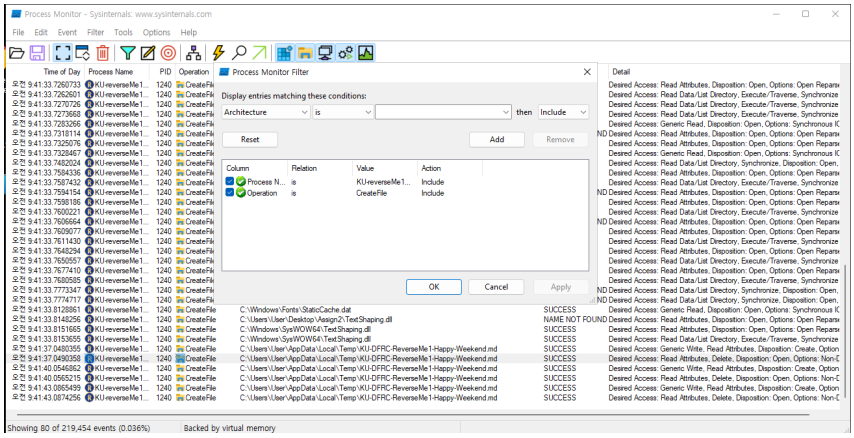
1. process monitor란?

- 프로세스 모니터는 윈도우 운영체제에서 실행중인 프로세스들이 런타임에 호출하는 API를 실시간으로 모니터링할 수 있는 도구이다.
- 문제에서 Run 버튼을 한번 누르고, 프로세스모니터를 보라고했으므로 그렇게 해보겠다.

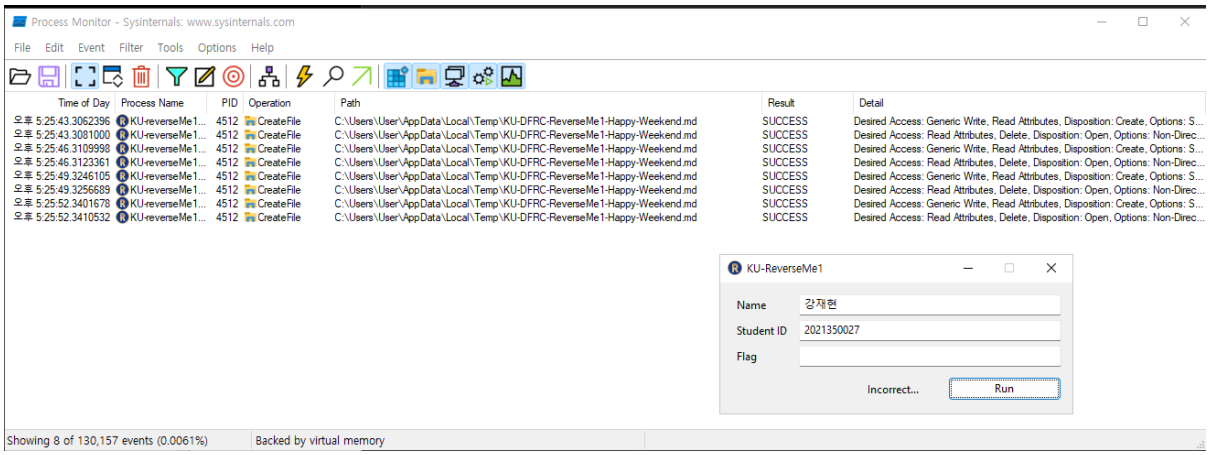
2. procmon 분석



- 프로세스의 이름만으로 필터링을 했더니 너무 많은 로그들이 나왔다. 그래서 필터에 CreateFile을 추가해서 다시 확인해볼 것이다.



- 그래도 너무 많은 파일이 생겨서 당황했다. 아마 맨 아래에 있는 파일이 Run을 눌렀을 때 생성된 파일같다.
- 더 정확히 하기 위해 실행 파일을 누르고, 프로세스모니터를 실행시킨 다음에, Run버튼만을 눌러보았다.



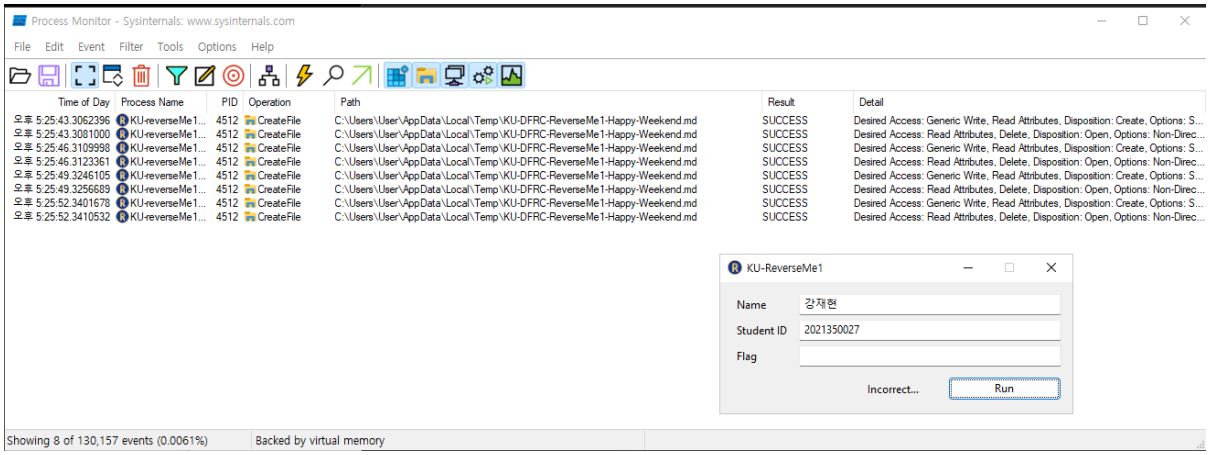
- 깔끔하게 한 이름의 파일만 생성된 것을 알 수 있었다.

3. 정답

`C:\Users\User\AppData\Local\Temp\KU-DFRC-ReverseMe1-Happy-Weekend.md`

[4] How many times was the file generated in total per pressing the button?

1. process monitor 분석



- CreateFile 관련해서 8번의 로그가 나왔다.
- 이 중 진짜 파일의 생성을 원하기 때문에 Detail에서 Generic write만을 선택한다.
  - 그러면 나머지 4가지 로그는 무엇일까? Detail에서 파일의 삭제가 일어나는 것을 알 수 있다.

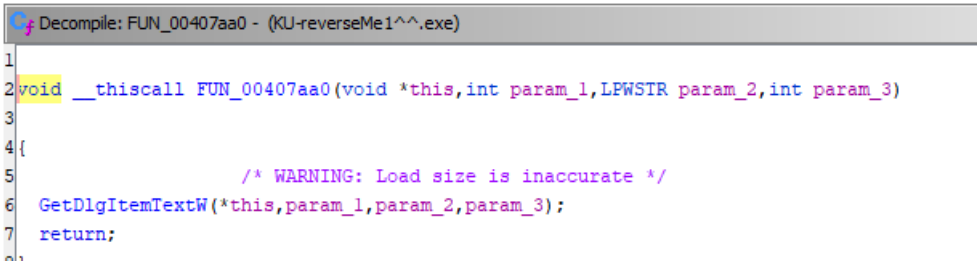
2. 결과

4번의 파일 생성이 일어난다.

[5]Identify and describe an algorithm that determines the number of times.

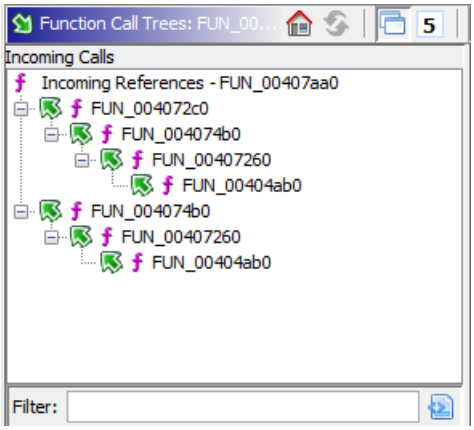
1. 관련 함수 찾기

1. Ghidra에서 `FUN_407aa0`에서 **GetDlgItemTestW**를 찾았다.
2. 형태를 보니 **GetDlgItemTestW** 그 자체라고 보면 된다.



sub_007aa0

3. 이 GetDlgItemTestW `00407aa0`를 사용하는 상위 함수는 두가지가 있었다.



a. `FUN_004072c0`

b. `FUN_004074b0`

4. 먼저 `FUN_004072c0`을 보겠다.

Decompile: FUN_004072c0 - (KU-reverseMe1^^.exe)

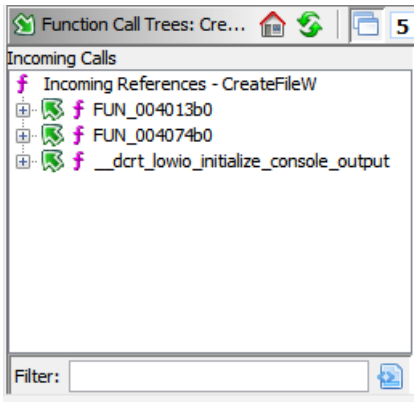
```
12  UINT UVar7;
13  int local_134;
14  undefined4 local_12c [12];
15  byte local_fc [52];
16  WCHAR local_c8 [32];
17  WCHAR local_88 [32];
18  WCHAR local_48 [32];
19  uint local_8;
20
21  local_8 = DAT_00423054 ^ (uint)&stack0xfffffffc;
22  puVar4 = &DAT_0041a82c;
23  puVar5 = local_12c;
24  for (iVar2 = 0xb; iVar2 != 0; iVar2 = iVar2 + -1) {
25    *puVar5 = *puVar4;
26    puVar4 = puVar4 + 1;
27    puVar5 = puVar5 + 1;
28  }
29  *(undefined2 *)puVar5 = *(undefined2 *)puVar4;
30  _memset(local_88,0,0x40);
31  _memset(local_c8,0,0x40);
32  _memset(local_48,0,0x40);
33  FUN_00407aa0((void *)(param_1 + 4),1000,local_88,0x20);
34  FUN_00407aa0((void *)(param_1 + 4),0x3ec,local_c8,0x20);
35  FUN_00407aa0((void *)(param_1 + 4),0x3ed,local_48,0x20);
36  if (((local_88[0] != L'\0') && (local_c8[0] != L'\0')) && (local_48[0] != L'\0')) {
37    _memset(local_fc,0,0x34);
38    for (local_134 = 0; local_134 < 0x34; local_134 = local_134 + 1) {
39      bVar1 = *(char *)((int)local_48 + local_134) + 0x11U ^ (&DAT_00423014)[local_134];
40      pWVar3 = (LPCWSTR)(uint)bVar1;
41      local_fc[local_134] = bVar1;
42      if (bVar1 != 0) {
43        UVar7 = 0x20;
44        FUN_00401b30(&stack0xffffffb0,0x80);
45        pWVar6 = extraout_ECX;
46        FUN_004019f0(&stack0xffffffac,L"Hmm... have you tried with all available candidates?");
47        FUN_00402f10((HWND)0x0,pWVar6,pWVar3,UVar7);
48        break;
49      }
50    }
51  }
52  FUN_00408878(local_8 ^ (uint)&stack0xfffffffc);
53  return;
54}
55
```

sub_4072c0

- **GetDlgItemTestW**를 세번 실행시키고, 그 결과값들로 if문을 실행시키고 있다.
- 그 if문 내에서는 또 `bVar1` 값에 따라 경고문이 나올지 안나올지가 결정된다.

- 지금은 파일의 생성 과정에 대해서 알아보는 중이기 때문에 이 함수는 관련이 없어 보인다.

5. `FUN_004074b0`을 보겠다.



CreateFileW와도 접점이 있는 것을 보아 이게 관련 함수로 보인다.

Decompile: FUN_004074b0 - (KU-reverseMe1^^.exe)

```
31
32  local_8 = DAT_00423054 ^ (uint)&stack0xfffffffc;
33  local_448 = param_1;
34  BVar6 = 0;
35  pvVar1 = FUN_00407a70((void *)(param_1 + 4),local_45c,0x3eb);
36  FUN_00407a30(pvVar1,BVar6);
37  puVar4 = &DAT_0041a8cc;
38  puVar5 = local_10c;
39  for (iVar3 = 0xb; iVar3 != 0; iVar3 = iVar3 + -1) {
40    *puVar5 = *puVar4;
41    puVar4 = puVar4 + 1;
42    puVar5 = puVar5 + 1;
43  }
44  *(undefined2 *)puVar5 = *(undefined2 *)puVar4;
45  iVar3 = FUN_004072c0(local_448);
46  if (iVar3 == 0) {
47    FUN_00407af0((void *)(local_448 + 4),0x3ee,L"Incorrect...");
48  }
49  else {
50    FUN_00407af0((void *)(local_448 + 4),0x3ee,L"Correct!");
51  }
52  puVar4 = (undefined4 *)L"KU-DFRC-ReverseMe1-Happy-Weekend.txt";
53  puVar5 = local_158;
54  for (iVar3 = 0x12; iVar3 != 0; iVar3 = iVar3 + -1) {
55    *puVar5 = *puVar4;
56    puVar4 = puVar4 + 1;
57    puVar5 = puVar5 + 1;
58  }
59  *(undefined2 *)puVar5 = *(undefined2 *)puVar4;
60  puVar4 = (undefined4 *)L"KU-DFRC-ReverseMe1-Happy-Weekend.yaml";
61  puVar5 = local_1f0;
62  for (iVar3 = 0x13; iVar3 != 0; iVar3 = iVar3 + -1) {
63    *puVar5 = *puVar4;
64    puVar4 = puVar4 + 1;
65    puVar5 = puVar5 + 1;
66  }
67  puVar4 = (undefined4 *)L"KU-DFRC-ReverseMe1-Happy-Weekend.json";
68  puVar5 = local_23c;
69  for (iVar3 = 0x13; iVar3 != 0; iVar3 = iVar3 + -1) {
70    *puVar5 = *puVar4;
71    puVar4 = puVar4 + 1;
72    puVar5 = puVar5 + 1;
73  }
74  puVar4 = (undefined4 *)L"KU-DFRC-ReverseMe1-Happy-Weekend.md";
```

sub_4074b0

- 앞에서 생성된 파일의 이름들이 보이므로, 이 함수가 파일의 생성에 관여하는 함수같다.
- 또한 97번째 줄에서 **CreateFileW**가 보이고, 112번째 줄에서 그 파일을 삭제하는 함수로 보아 관련함수가 확실하다.

```
for (; local_450 != 0; local_450 = local_450 + -1) {
  local_44c = CreateFileW(local_444,0x40000000,0,(LPSECURITY_ATTRIBUTES)0x0,1,2,(HANDLE)0x0);
  if (local_44c != (HANDLE)0xffffffff) {
    puVar4 = (undefined4 *)L"Hello, KU. Become a Reverse-engineer!";
    puVar5 = local_94;
    for (iVar3 = 0x13; iVar3 != 0; iVar3 = iVar3 + -1) {
      *puVar5 = *puVar4;
      puVar4 = puVar4 + 1;
      puVar5 = puVar5 + 1;
    }
    lpOverlapped = (LPOVERLAPPED)0x0;
    lpNumberOfBytesWritten = &local_460;
    sVar2 = _wcslen((wchar_t *)local_94);
    WriteFile(local_44c,local_94,sVar2 << 1,lpNumberOfBytesWritten,lpOverlapped);
    CloseHandle(local_44c);
  }
  DeleteFileW(local_444);
  Sleep(local_458);
}
```

sub_4074b0

2. `FUN_004074b0` 분석

- 2번에서 말했듯이 `FUN_004074b0`이 RUN을 눌렀을 때 파일의 생성 횟수를 결정하는 함수이다.

```
for (; local_450 != 0; local_450 = local_450 + -1) {
  local_44c = CreateFileW(local_444,0x40000000,0,(LPSECURITY_ATTRIBUTES)0x0,1,2,(HANDLE)0x0);
  if (local_44c != (HANDLE)0xffffffff) {
    puVar4 = (undefined4 *)L"Hello, KU. Become a Reverse-engineer!";
    puVar5 = local_94;
    for (iVar3 = 0x13; iVar3 != 0; iVar3 = iVar3 + -1) {
      *puVar5 = *puVar4;
      puVar4 = puVar4 + 1;
      puVar5 = puVar5 + 1;
    }
    lpOverlapped = (LPOVERLAPPED)0x0;
    lpNumberOfBytesWritten = &local_460;
    sVar2 = _wcslen((wchar_t *)local_94);
    WriteFile(local_44c,local_94,sVar2 << 1,lpNumberOfBytesWritten,lpOverlapped);
    CloseHandle(local_44c);
  }
  DeleteFileW(local_444);
  Sleep(local_458);
}
```

sub_4074b0에서의 반복문

- `local_450`을 -1씩 하면서 0이 될 때 까지 반복문을 돌린다. 즉, `local_450`만큼 파일의 생성된다.
- `local_450`은 어떻게 결정이 될까?

```
FUN_00407aa0((void *)(local_448 + 4),0x3ec,local_48,0x20);
local_454 = _wcstoul(local_48,(wchar_t **)0x0,10);
local_458 = 3000;
local_450 = local_454 % 2 + 3;
```

sub_4074b0의 반복문 이전의 변수들 정의 과정

  - 여기서 FUN_00407aa0은 **GetDlgItemTextW**의 그대로로, 컨트롤 식별자가 `0x3ec`인 것으로 보아 **학번**에 대한 함수이다.
    - 사실 컨트롤 식별자가 학번 입력란의 식별자인지는 정확하지는 않았다.
    - 하지만 6번 문제에서 본 `FUN_004072c0`에서 `FUN_00407aa0` **(GetDlgItemW)**을 3번 부르는 부분이 있는데, 그 식별자의 순서가 `1000, 0x3ec, 0x3ed`로 되어있으므로, 순서대로 `이름, 학번, flag`일 것이라 예상

할 수 있었다.

- 그리고 `0x3ed`는 Flag인 것이 6번 과제에서 발견되었고,
- 더 정확히 하려면,, `0x3ec`는 학번과 이름 둘 중 하나인데, 변수들을 고정하고 학번과 이름의 값을 홀수 짝수로 대입했을 때, 학번에서 파일 생성의 개수가 4개와 3개로 나뉘는 것을 알 수 있었다.
- 따라서 `0x3ec`를 식별자로 가진 것은 **학번**의 입력이라는 것을 알 수 있었다.

- 따라서 `local_454`는

```
local_454 = _wcstoul(학번 buffer 주소, 0x0, 10)
```

으로 결정이 된다.

- **wcstoul**

Decompile: _wcstoul - (KU-reverseMe1^^.exe)

```
1
2  /* Library Function - Single Match
3      _wcstoul
4
5     Library: Visual Studio 2019 Release */
6
7  ulong __cdecl _wcstoul(wchar_t *_Str,wchar_t **_EndPtr,int _Radix)
8
9  {
10   ulong uVar1;
11   c_string_character_source<wchar_t> extraout_var;
12   int extraout_var_00;
13
14   make_c_string_character_source<>((undefined4 *)&stack0xfffffff8,_Str,_EndPtr);
15   uVar1 = __crt_strtox::
16           parse_integer<unsigned_long,class___crt_strtox::c_string_character_source<wchar_t>_>
17                     ((__crt_locale_pointers *)0x0,extraout_var,extraout_var_00,
18                      (bool)(undefined)_Radix);
19   return uVar1;
20 }
```

__wcstoul

- 처음에는 wcstoul이라는 함수를 직접 해석하려고했다. 왜냐하면… 인터넷에 검색을 해도 나오지 않았기 때문이다.
- 그렇게 이 요상한 함수를 계속 봐도 답이 나오지가 않았다.
- 그래서 다시 인터넷에 검색을 해보니 설명이 나왔다. 왜,,, 이전에는 검색이 안됐을까.. 오타를 냈었나보다.
- 그래서 결국 wcstoul은 입력받은 스트링을 정수로 바꿔주는 함수이다. 즉 내가 입력한 학번의 문자열을 숫자형으로 바꿔주는 것이다.
- 따라서 uVar1은 내 학번의 숫자 모습이라는 것을 알 수 있었다.

3. `local_450`을 구하기

- `local_450 = local_454 % 2 + 3`을 구하기 위해서 우리는 **wcstoul**의 반환값을 위에서 알아냈다.
- 따라서 local_454=2021350027인 나의 학번으로, 홀수임을 알 수 있다.
- `local_450 = 2021350027 % 2 + 3 = 1 + 3 = 4`

- 따라서 for문의 인자인 `local_450`은 4임을 알 수 있고, **CreateFileW**가 4번 실행됨을 알았다.

4. 결과

- `FUN_004074b0`이 **CreateFileW**가 실행되게 하는 함수의 이름이다.
- 그 중 Ghidra에서 디컴파일했을 때, 96번째 줄의 for문에 의해서 **CreateFileW**이 실행된다.
- for문의 인자인 `local_450`에 의해서 반복 횟수가 정해지며, `local_450`은 **GetDlgItemTextW**로 받은 내가 입력한 학번인 `local_454`를 사용해서 만든다.
- 학번을 2로 나눴을 때의 나머지와 3을 더한다. 즉, 내 학번은 홀수이기 때문에 1+3=4가 `local_450`이다.
- 따라서 4번에서 알아낸 것과 같이 **CreateFile**이 **4번** 실행됨을 알 수 있다.

[6] Find a valid flag.

1. 오류 메시지 박스가 뜨지 않게 하려면?

```
33  FUN_00407aa0((void *)(param_1 + 4),1000,local_88,0x20);
34  FUN_00407aa0((void *)(param_1 + 4),0x3ec,local_c8,0x20);
35  FUN_00407aa0((void *)(param_1 + 4),0x3ed,local_48,0x20);
36  if (((local_88[0] != L'\0') && (local_c8[0] != L'\0')) && (local_48[0] != L'\0')) {
37    _memset(local_fc,0,0x34);
38    for (local_134 = 0; local_134 < 0x34; local_134 = local_134 + 1) {
39      bVar1 = *(char *)((int)local_48 + local_134) + 0x11U ^ (&DAT_00423014)[local_134];
40      pWVar3 = (LPCWSTR)(uint)bVar1;
41      local_fc[local_134] = bVar1;
42      if (bVar1 != 0) {
43        UVar7 = 0x20;
44        FUN_00401b30(&stack0xfffffeb0,0x80);
45        pWVar6 = extraout_ECX;
46        FUN_004019f0(&stack0xfffffeac,L"Hmm... have you tried with all available candidates?");
47        FUN_00402f10((HWND)0x0,pWVar6,pWVar3,UVar7);
48        break;
49      }
50    }
51  }
```

sub_4072c0의 일부분

- 일단 오류 메시지박스가 뜨지 않게 해야 성공이 나온다.
  - `local_48`은 세번째 `GetDlgItemText`의 인자로 우리가 입력한 **Flag**가 담겨있는 주소이다. (왜냐하면, `FUN_00407aa0`은 `GetDlgItemTaextW`함수가 들어있으며, 3번째 인자의 위치는 입력받는 버퍼의 주소가 있기 때문이다.)
  - 우리가 입력한 **Flag**를 **DAT_00423014**와 잘 조합해서 결과값이 0이 되면 오류가 뜨지 않는다. (42번 줄의 if문을 통과한다.)
- 이 if문을 잘 넘어가면 다음 함수로 넘어가서 Correct가 뜨는 것을 보아 이 부분을 보는게 맞다.
- 쉽게 코드를 요약하자면 아래와 같다.

```
for (int i = 0; i < 0x34; i++) {
  bVar1 = *(char*)((int)loacal_48+i) + 0xll ^ (&DAT_00423014)[i];
  //local_48은 우리가 내린 flag값
  ..
  if (bVar1 != 0){
```

```
    메시지박스 실행
}
```

```
00423010  44 5B 40 00 58 11 60 11  60 11 55 11 3E 11 8A 11  D[@.X.`.`.U.>.Š.
00423020  41 11 86 11 3E 11 77 11  41 11 86 11 7F 11 75 11  A.†.>.w.A.†...u.
00423030  3E 11 72 11 3E 11 87 11  72 11 7D 11 7A 11 75 11  >.r.>.‡.r.}.z.u.
00423040  3E 11 7C 11 43 11 8A 11  FF FF FF FF 00 00 00 00  >.|.C.Š.ÿÿÿÿ....
```

DAT_00423014 부근의 hex값

2. 연산 실수를 계속했다...

```
bVar1 = *(char*)((int)loacal_48+local_134) + 0xll ^ (&DAT_00423014)[local_134];
```

- 처음에는 연산의 순서가 디컴파일 과정에서 정확하게 써 있는 줄 알고 계속 **DAT_00423014**에 0x11을 XOR하고 입력한 Flag값과 더했을 때 0이 되어야 하는 줄 알았다.
- 그래서 계속 `Flag = - (0x11 ^ DAT_00423014)`라고 생각했다.
  - 하지만 당연히 제대로된 아스키값이 나오지 않고, 계속 다른 함수들을 보면서 이게 아닌건가 찾아다녔다.

3. 다시 명령어를 자세히 읽기

```
00407402 0f 8d 8d        JGE         LAB_00407495
         00 00 00
00407408 8b 85 c4        MOV         EAX,dword ptr [EBP + local_140]
         fe ff ff
0040740e 03 85 d0        ADD         EAX,dword ptr [EBP + local_134]
         fe ff ff
00407414 8a 08           MOV         param_1,byte ptr [EAX]
00407416 88 8d d7        MOV         byte ptr [EBP + local_12d],param_1
         fe ff ff
0040741c 0f b6 95        MOVZX       EDX,byte ptr [EBP + local_12d]
         d7 fe ff ff
00407423 83 c2 11        ADD         EDX,0x11
00407426 88 95 d7        MOV         byte ptr [EBP + local_12d],DL
         fe ff ff
0040742c 8b 85 d0        MOV         EAX,dword ptr [EBP + local_134]
         fe ff ff
00407432 0f b6 88        MOVZX       param_1,byte ptr [EAX + DAT_00423014]          = 58h
         14 30 42 00
00407439 0f b6 95        MOVZX       EDX,byte ptr [EBP + local_12d]
         d7 fe ff ff
00407440 33 d1           XOR         EDX,param_1
00407442 88 95 d7        MOV         byte ptr [EBP + local_12d],DL
         fe ff ff
00407448 8b 85 d0        MOV         EAX,dword ptr [EBP + local_134]
         fe ff ff
0040744e 8a 8d d7        MOV         param_1,byte ptr [EBP + local_12d]
```

- 명령어를 다시 해석해보니 우리가 입력한 `Flag`에 `0x11`을 더한 다음에 `DAT_00423014`를 XOR한 것이 `iVar1`이었다.
- 따라서 `bVar1 = (Flag[i] + 0x11) ^ DAT_00423014[i]`가 정확한 계산 방법이었다.
  - 조금만 더 명령어를 자세히 볼걸.. 1번에서 후회를 했으면서 시간이 지나서 까먹었다.
- `bVar1=0`이 되어야하기 때문에 `Flag[i] + 0x11 = DAT_00423014[i]`이 되어야 한다.

정리하면 `Flag[i] = DAT_00423014[i] - 0x11`이다.

- `DAT_00423014`에는 `11`이 문자들 중간중간에 껴있다. `0x11`을 거기서 빼면 0이 되므로, 문자열을 입력할 때 단어 사이사이가 0x0이 되는 것과도 일치한다.

- i는 위 함수에서 local_134로 0부터 0x34까지 이루어진다. 이에 맞춰서 DAT를 본다.

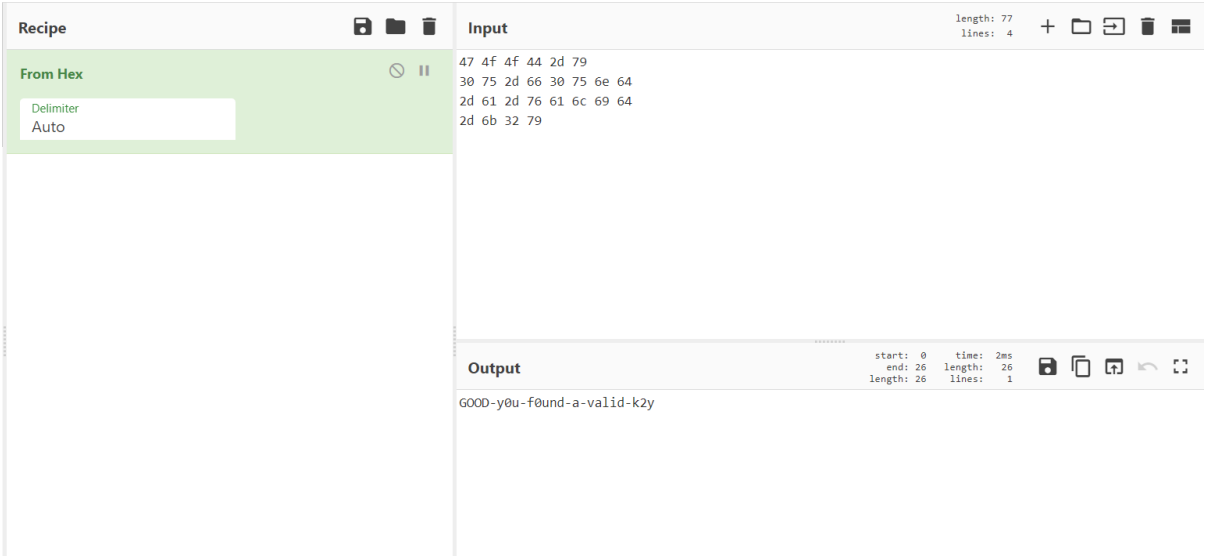
```
00423010        58 60 60 55 3e 8a
00423020  41 86 3e 77 41 86 7f 75
00423030  3e 72 3e 87 72 7d 7a 75
00423040  3e 7c 43 8a

(중간에 껴있는 11은 제거했다.)
```

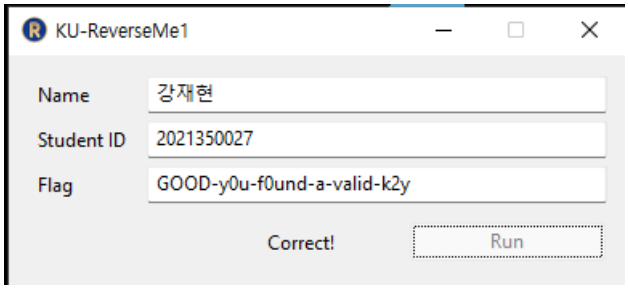
- 이 부분에서 0x11을 빼면 아래와 같다.

```
      47 4f 4f 44 2d 79
30 75 2d 66 30 75 6e 64
2d 61 2d 76 61 6c 69 64
2d 6b 32 79
```

- 이 hex들을 Cyber chief에서 아스키 코드로 변환하면 `GOOd-y0u-f0und-a-valid-k2y`가 Flag로 나온다.



4. 결과



Flag: `GOOD-y0u-f0und-a-valid-k2y`