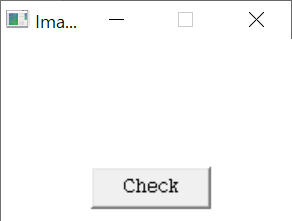
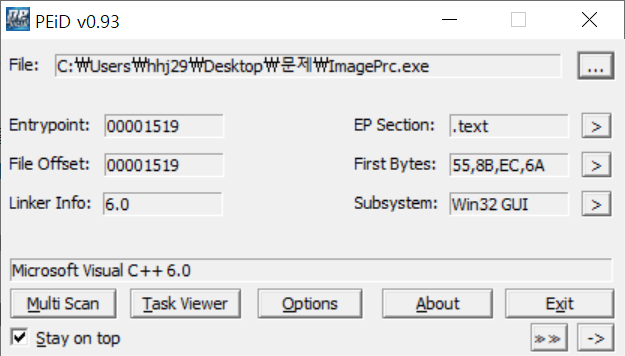
1. 문제 확인

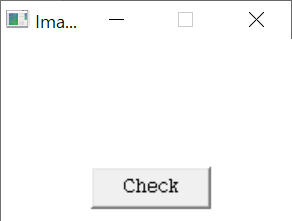


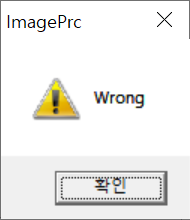
2. 과정

패킹 여부를 확인해보았다. 패킹 되어있지 않았다.



파일을 실행해봤는데 체크를 클릭하면



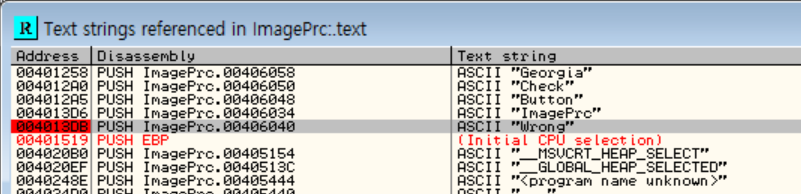
다음과 같이 Wrong이라는 문구가 뜨는 것을 확인할 수 있다.

뭔가 입력하는 게 없는지 확인해봤는데 실행창에 그림이 그려지는 것을 알 수 있다.



버튼을 누르면 위와 같이 Wrong창이 떴다.

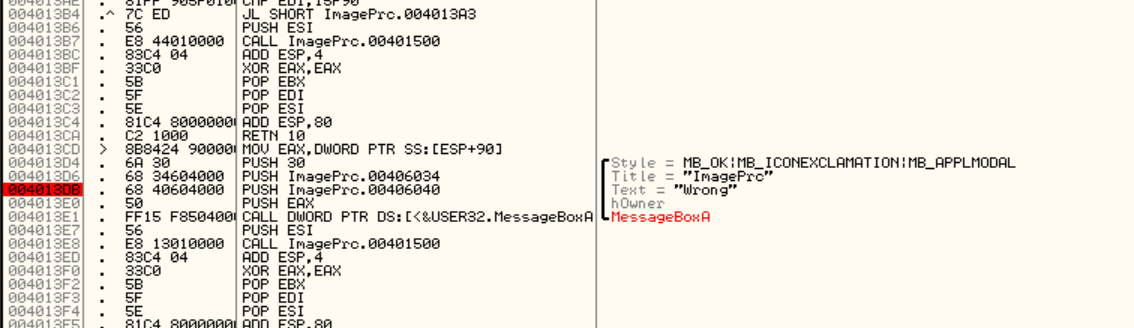
올리디버거를 열어 문자열이 뜨는 부분을 찾아보았다.



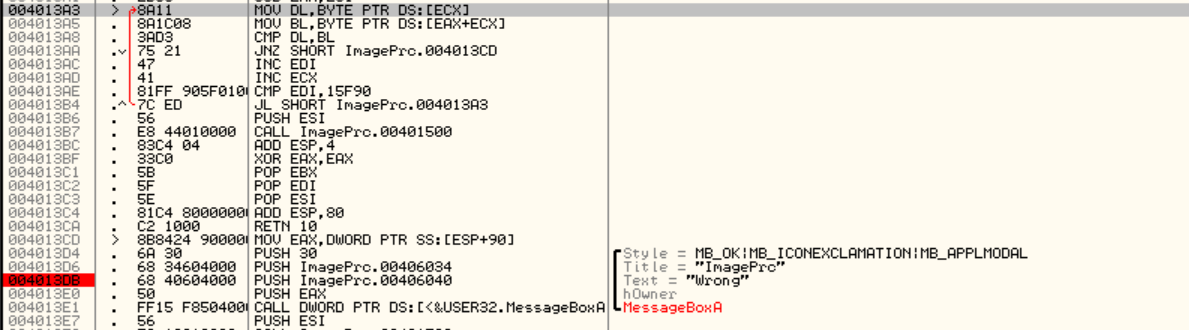
따로 성공문자열은 보이지 않았다.

해당 부분으로 디버깅해보면서 살펴보았다.

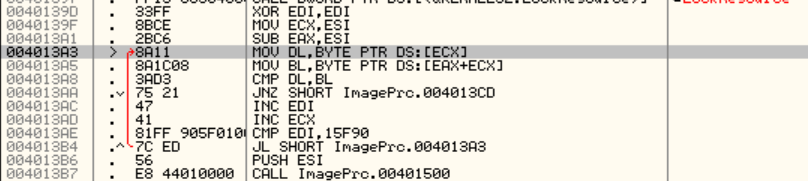
Wrong 문자열이 나오는 곳은 쉽게 찾을 수 있었다.



이 코드 윗부분을 확인해봤는데



004013A3에서 루프문이 있는데 중간에 점프문으로 값이 다르면 실패문자열로 향하는 것을 알 수 있다. 이 부분을 좀 자세히 보았다.



처음에는 EDI의 값을 XOR로 0 초기화를 하는 것 같고,

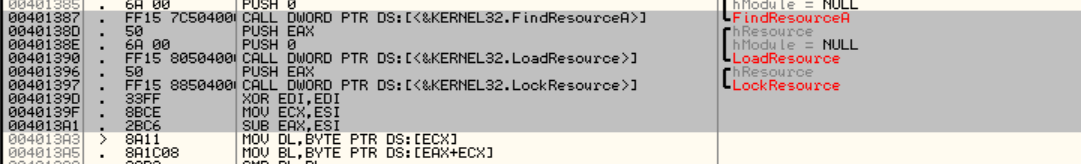
DL에 ECX 값을, BL에 EAX+ECX 값을 넣고 이 둘의 값을 비교해 일치하지 않으면 실패문자열로 점프한다.

같다면 EDI와 EAX 각각 1씩 증가시키고 EDI와 15F90(90000)을 비교한다. 반복문이 끝나면 CALL 함수가 실행되는 것 같다.

여기서 다음에는 어떤 부분을 봐야 하는지 막혀서 다른 분의 풀이를 참고해봤다.

CMP문으로 비교하던 루프문은 사용자가 그린 이미지와 정답 이미지의 픽셀 하나하나를 비교하는 부분이라고 한다.

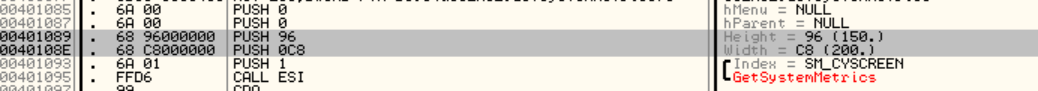
좀 놓친 곳은 아까 루프문의 윗부분인데



이곳을 거치면 각각 사용자가 그린 그림영역의 주소와 리소스영역 주소가 ESI와 EAX에 저장된다고 한다. (FindResource: 리소스 찾기, LoadResource: 리소스 로드)

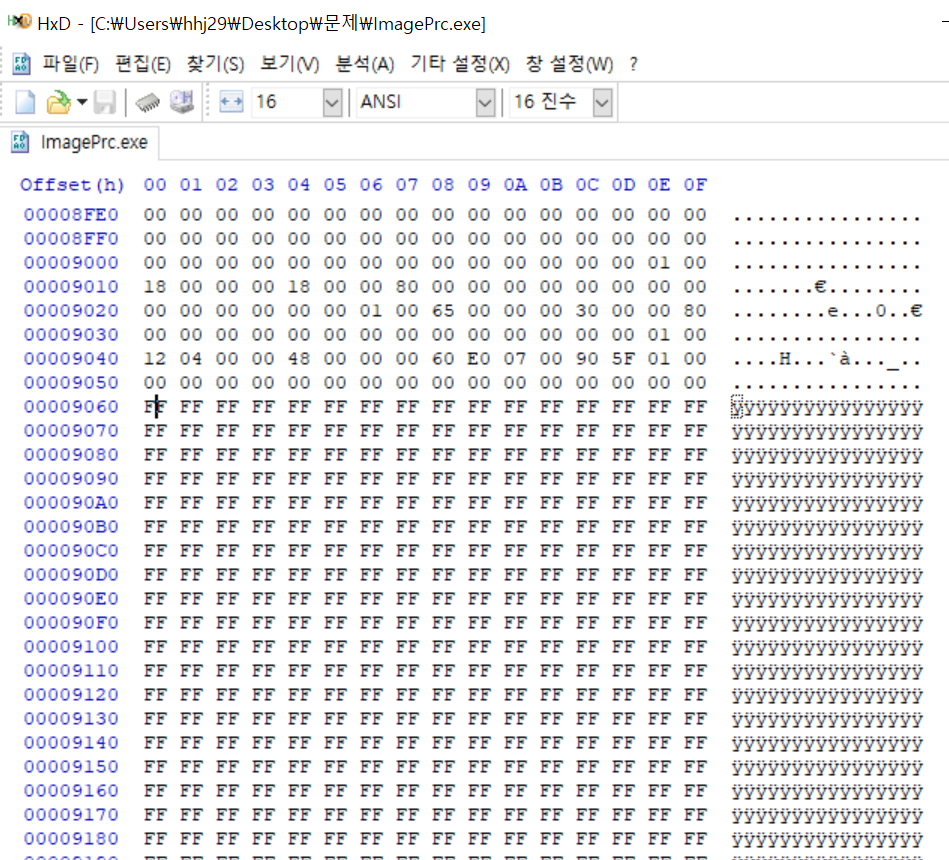
이제 아래 루프문을 돌면서 리소스의 픽셀을 비교하게 될 것이다.

여기서 BMP 파일의 사이즈는 200x150 라고 한다.



리소스 영역을 복사해 새로 만든 BMP 파일에 로드해보면 된다고 한다. 리소스는 비트맵이라서 파일 구조를 살펴보면 리소스부분을 찾을 수 있다고 한다.

HxD로 파일의 바이너리를 확인해보았다.



이 부분의 리소스 영역을 따서 새로운 비트맵을 생성해 그곳에 붙이면 해결할 수 있다.

리소스 영역은 FF으로 시작되는 부분부터 따서 그림판으로 200x150 크기의 비트맵을 생성해 리소스영역에 붙여넣기 해준다.

해당 파일을 열어보면 다음과 같은 글자가 보이게 된다.

