리버스 공학 / 기말 레포트

태그	
체크박스	☑
최종 편집 일시	@2024년 12월 6일 오후 6:04

문서 개요

- 해당 문서는 리버스 공학 기말 프로젝트를 위해 작성되었다.
- 작성자 : 이창엽 / 20117669 / 컴퓨터소프트웨어학부 컴퓨터공학 전공

스타크래프트 시디키 알고리즘 분석

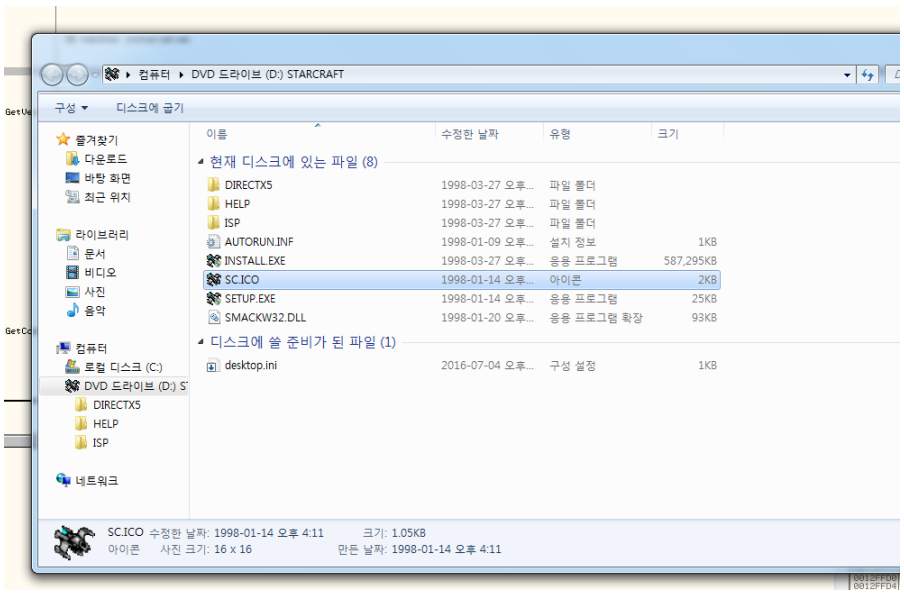
0. 설정 타겟

> 스타크래프트 ISO

- https://m.blog.naver.com/lh6746/220579576600
  - 스타크래프트 오리지널 버전 (시디키 알고리즘 분석)

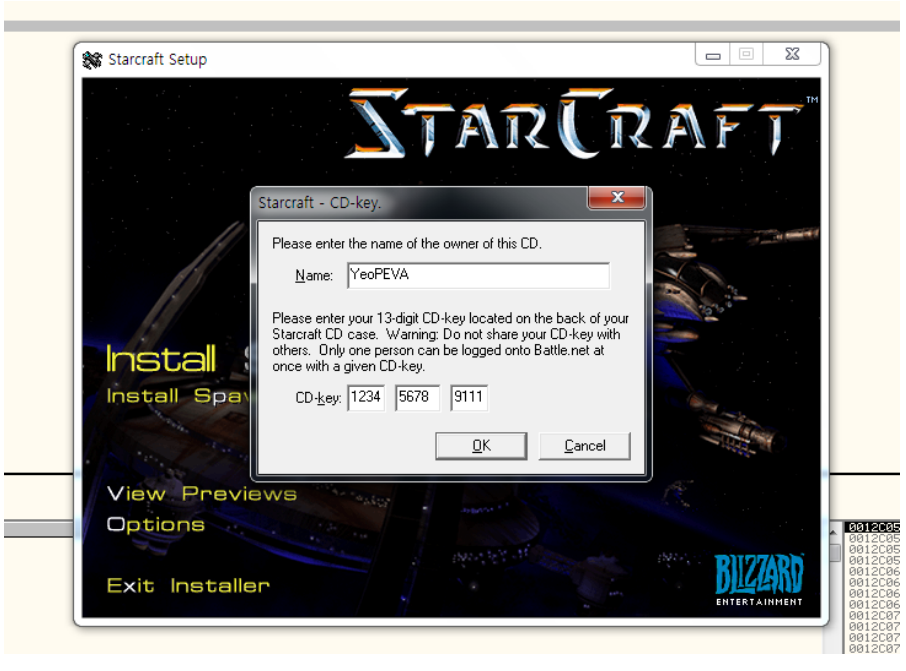
설정 타겟은 스타크래프트 오리지널 버전 기준, 설치 과정에서 시리얼 키 입력을 요구한다.

해당 시리얼 키 부분과 관련하여, 시리얼 키 인증 과정을 확인하고, 그에 따른 시리얼 키를 제작할 수 있는 별도의 프로그램을 제작하는 것을 목표로 진행하였다.



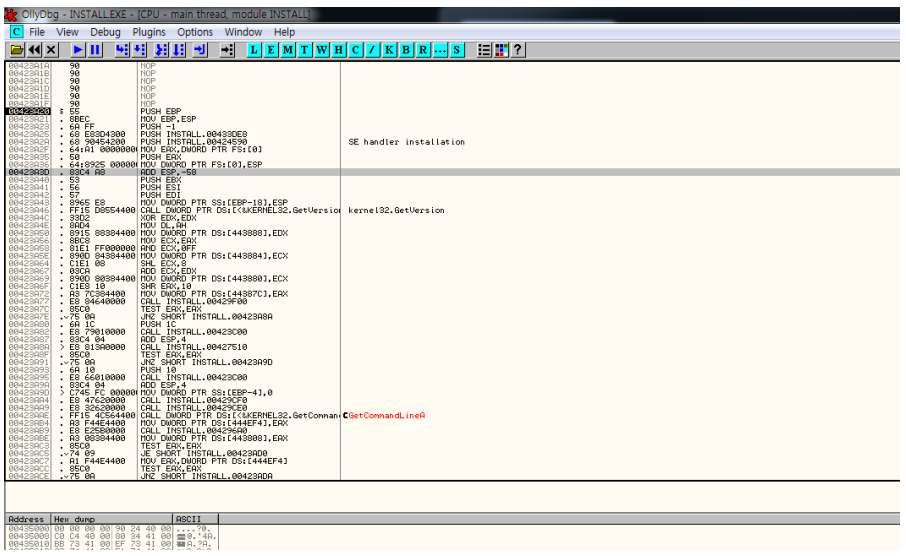
- 분석 대상 : INSTALL.EXE

- 분석 대상 / 실행 화면



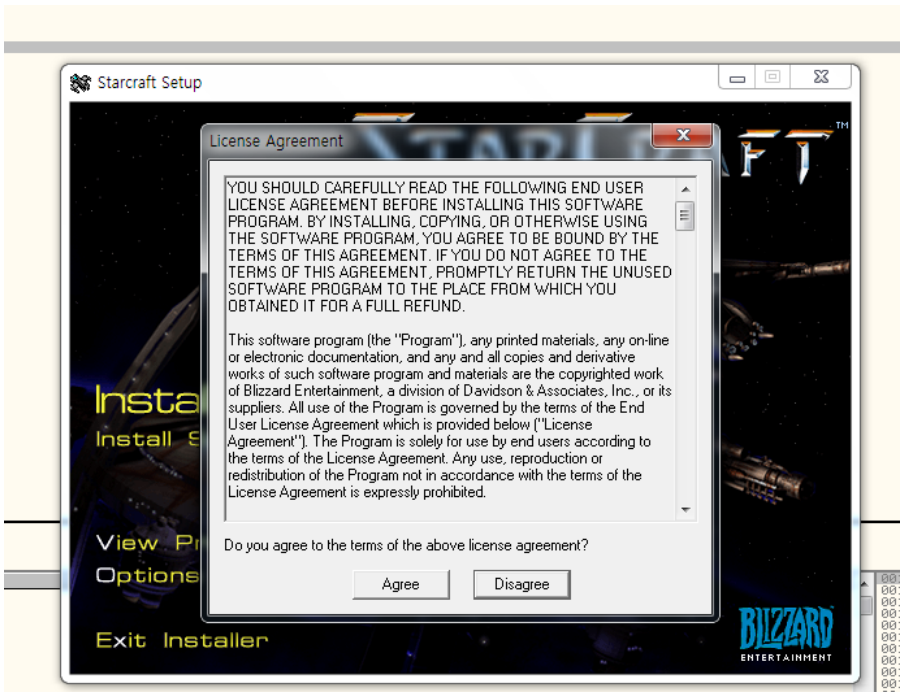
- 분석 대상 목표
  - CD-Key를 인증할 수 있는 키 제작
  - CD-Key 검증 과정 분석

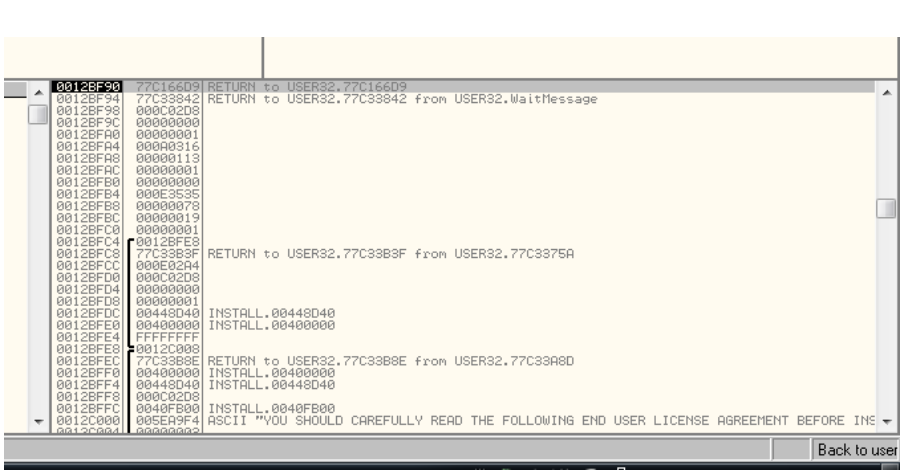
1. 타겟 분석



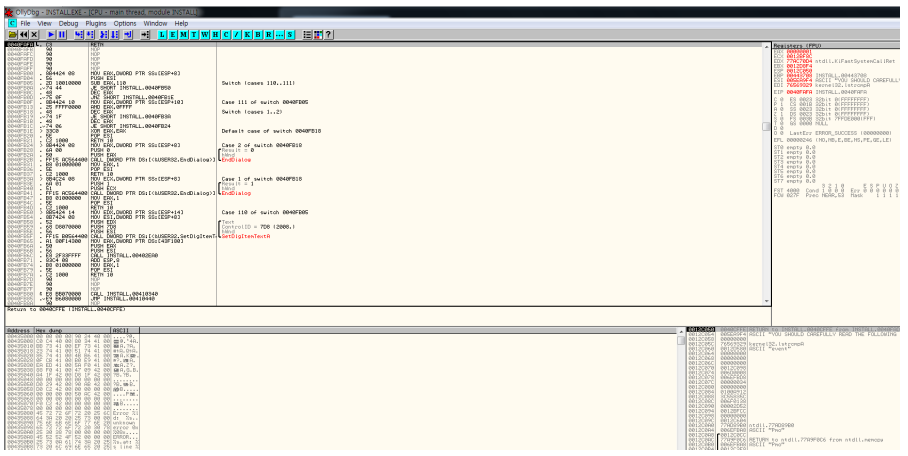
본격적인 타겟 분석을 위해, Ollydbg를 통해 설치 프로그램을 열었다.

그 후 실행 과정을 통해, 프로그램을 실행하는 과정을 거친 이후,

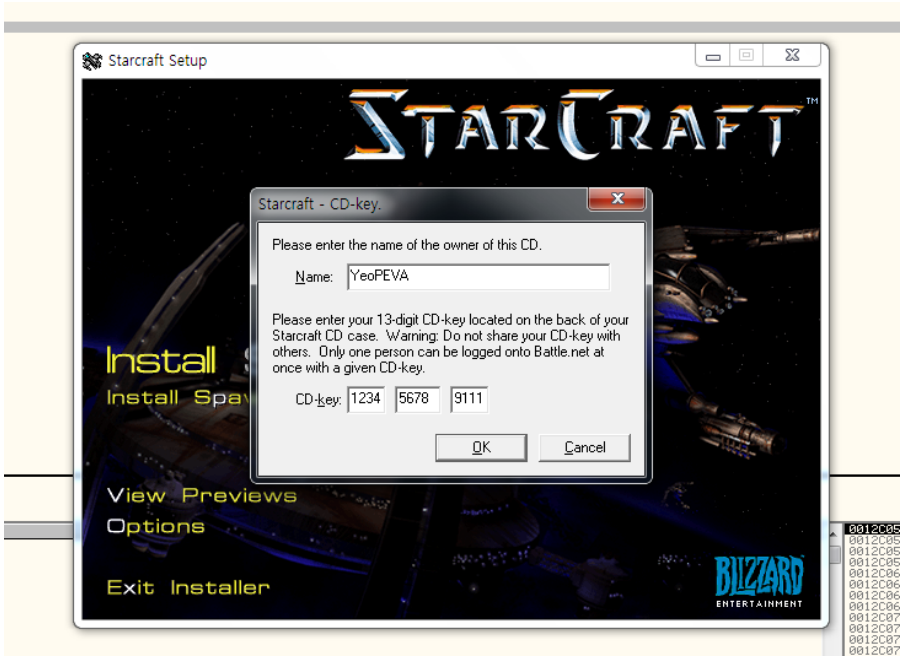




Back to User를 위와 같이 설정한 후,



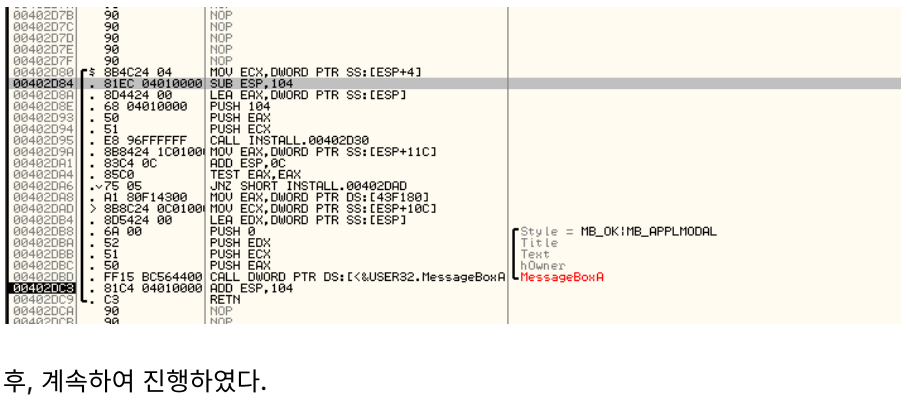
위에서 나온 이용 약관을 클릭하면, 상단과 같이 Call 명령이 일어난 바로 다음 위치로 이동할 수 있었다.



상단 방법으로 Back to User 모드의 동작을 확인하였으며,

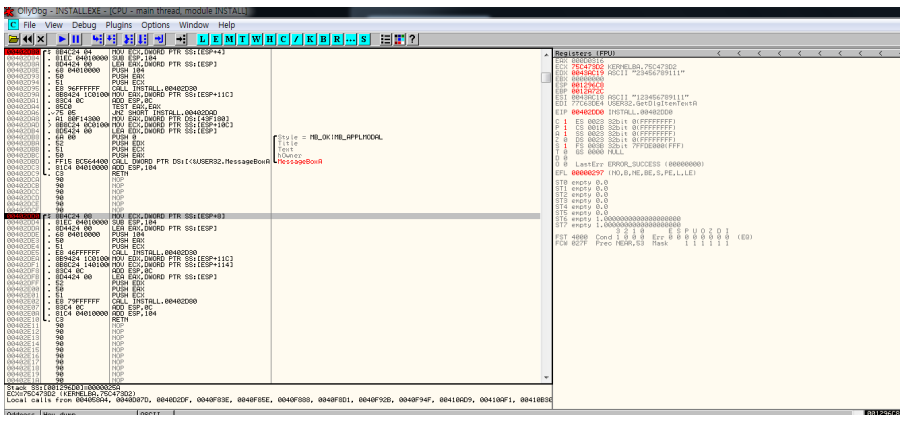
동일한 방식으로 키 인증 과정에 접근하기 위하여, CD-Key를 입력하고, 진행하였다.

그 결과 위와 같이 이동할 수 있었다.

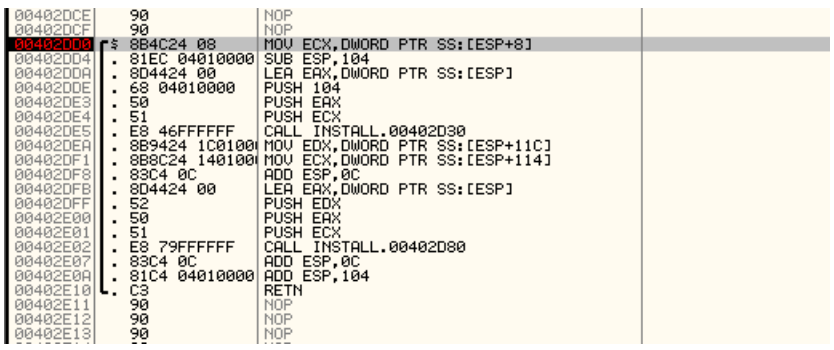


그 후, 이동한 측에 브레이크 포인트를 설정한 후, 계속하여 진행하였다.

해당 부분은 메시지 박스 출력을 위해 활용되는 구문으로 확인 되었다.

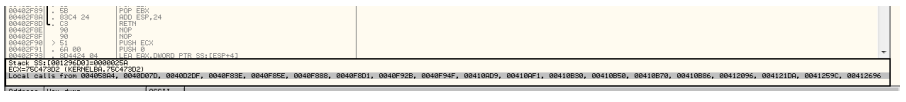


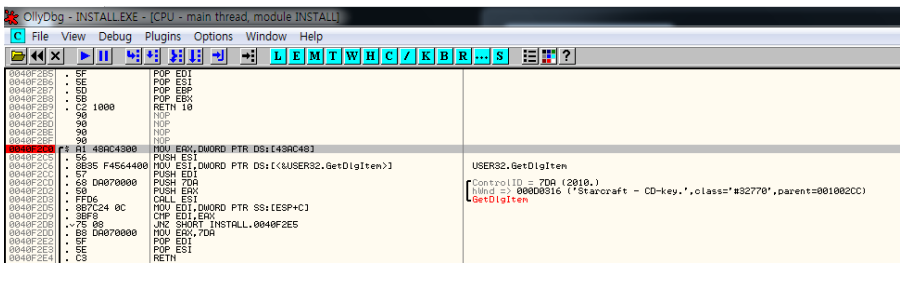
이후, 분석을 이어서 진행하는 과정에서,



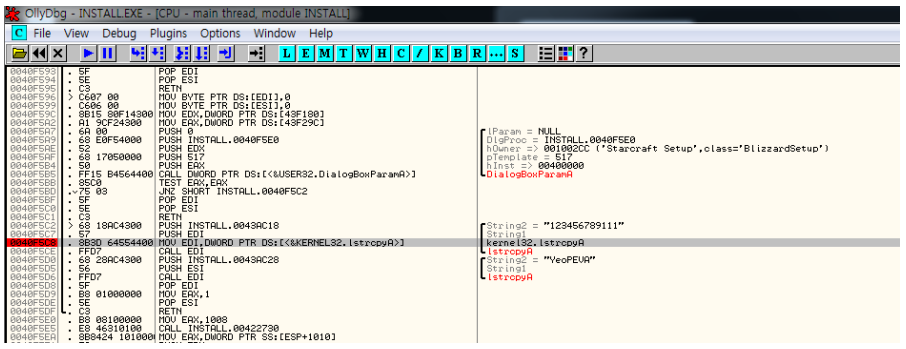
MOV ECX, DWORD PTR SS:[ESP+8] 내 다양한 주소에서 불려지는 명령어들이 존재하는 것을 확인하였으며,

해당 주소들을 확인해나가기 시작하였다.

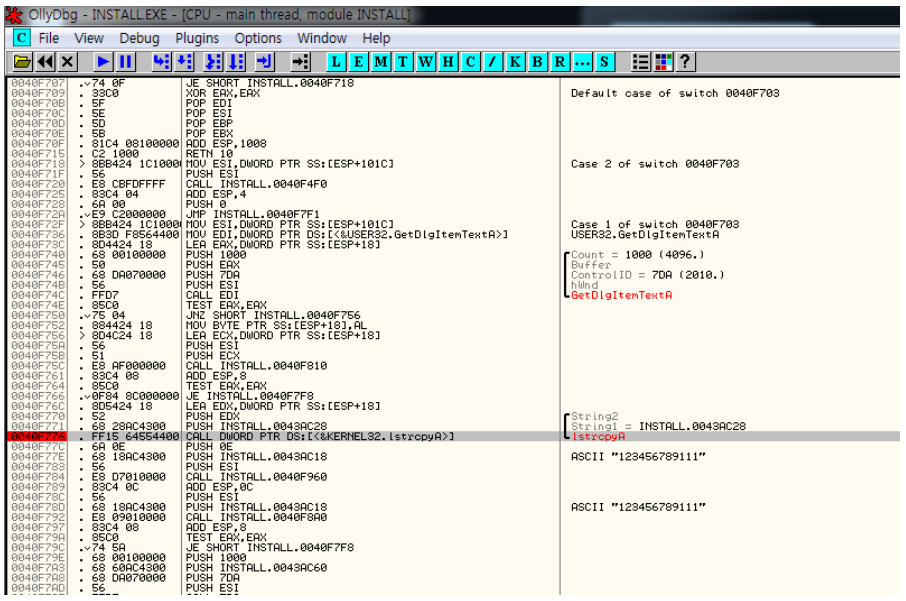




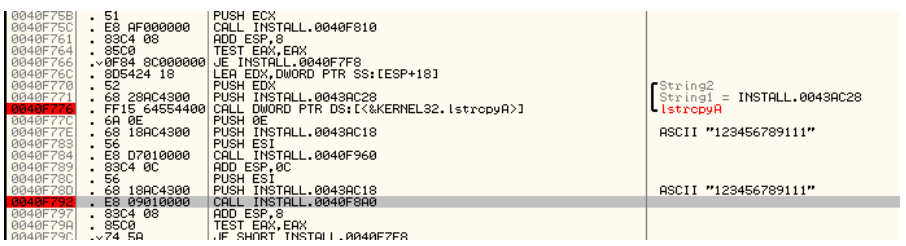
해당 주소들을 확인하는 과정에서, CD-key와 연계된 것으로 확인되는 부분을 확인하였으며,
해당 측으로 브레이크 포인트를 걸어준 이후, 분석을 이어나갔다.



그러는 과정에서, 키 값 및 유저 이름과 관련된 구문을 확인할 수 있었으며, 검증을 위해 준비하는 과정 중 하나로 파악되었다.



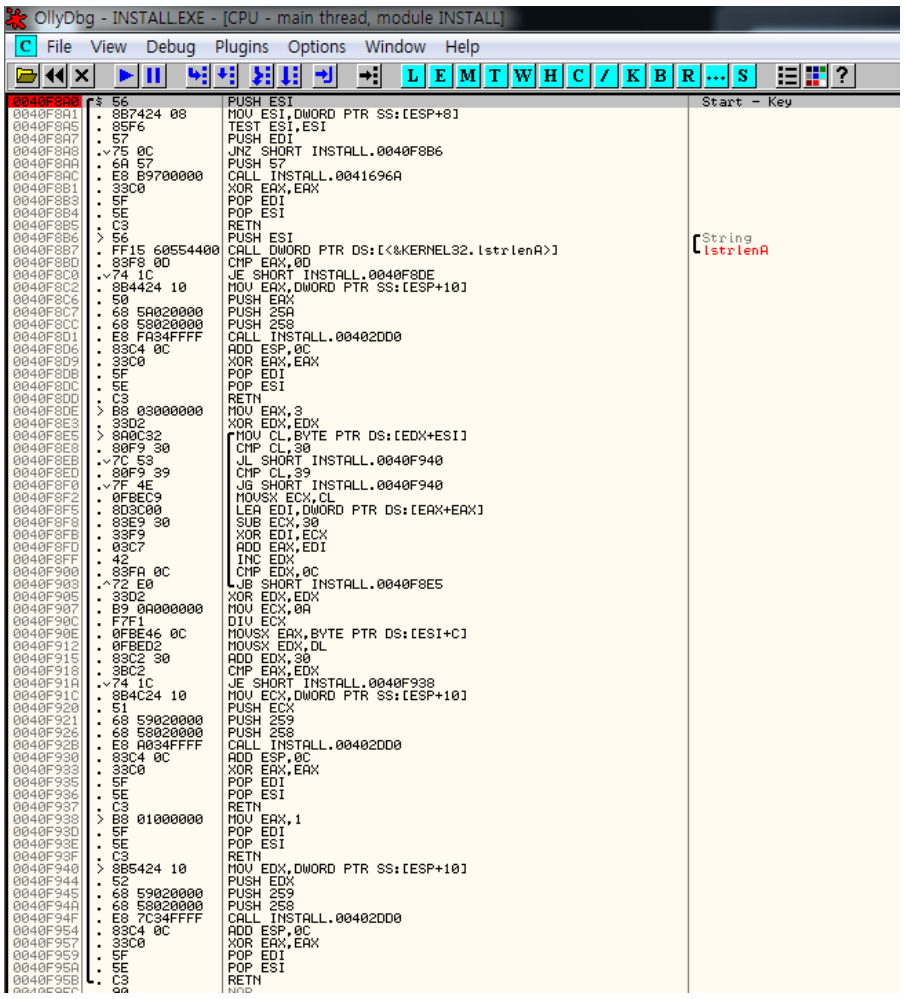
이후 추가적으로 확인하던 도중, 시리얼 키를 PUSH하고, 함수를 호출하는 구문들을 더 살펴보았다.
첫 번째 콜에 브레이크 포인트를 걸고, 분석을 진행하였으나 별다른 결과는 없었으며, 다음 호출 구문을 좀 더 상세히 분석하기로 하였다.



좀 더 상세한 확인을 위해, 브레이크 포인트 및 f7를 통해 step into를 진행하였다.

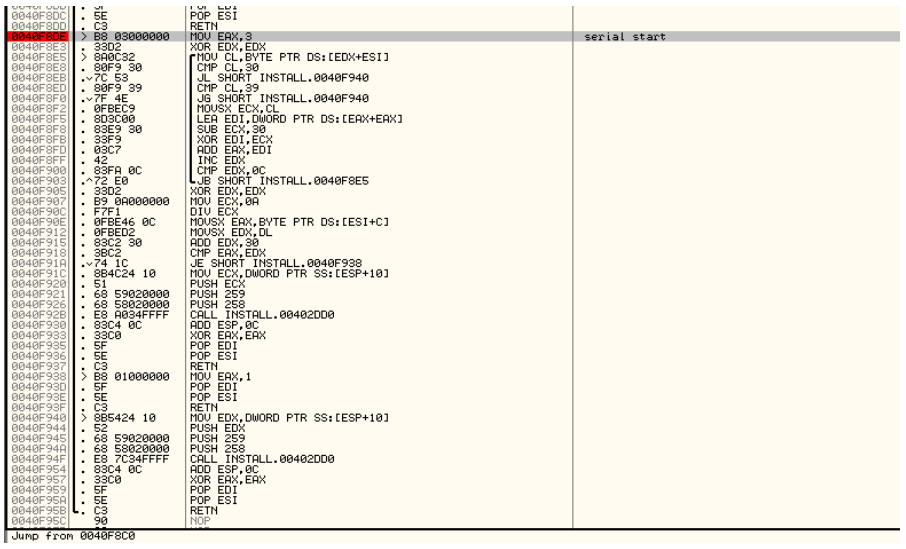
그 결과, 해당 함수 내에서 진행되는 부분이 키 시리얼 인증과 관련된 구문임을 확인할 수 있었으며,
해당 구문들을 상세히 분석하기 시작하였다.

2. 분석 시작위치 결정



해당 코드들을 확인하고, 브레이크 포인트를 활용하여 검증한 결과, lstrlenA를 통해 키의 길이가 13자리가 맞는지를 확인하는 구문이였다.

그렇기에 본격적인 시리얼 검증 과정과 관련된 부분은 mov eax,3 이라는 것을 확인할 수 있었다.



위에서 말했듯이, 앞에서 다룬 부분은 키 시리얼이 13자리가 맞는지를 검증하는 구문이였으며, 시리얼 키가 그에 맞지 않다면, 다시 시리얼 키 입력 로직으로 돌아가는 구문이였다.

그렇기에 본격적인 코드 분석은 해당 구문을 분석해야한다고 판단하게 되었다.

3. 코드 분석

```
0040F8DE  |> B8 03000000    MOV EAX,3                ; EAX 초기화 (시작값 3)
0040F8E3  |. 33D2           XOR EDX,EDX              ; EDX = 0, 문자열의 현재 위치 추적

0040F8E5  |> 8A0C32         /MOV CL,BYTE PTR DS:[EDX+ESI]  ; 문자열에서 현재 문자 가져옴
0040F8E8  |. 80F9 30         |CMP CL,30                     ; 문자 값이 '0'(ASCII 0x30)보다 작은지 확인
0040F8EB  |. 7C 53           |JL SHORT INSTALL.0040F940     ; 작으면 실패 처리로 점프
0040F8ED  |. 80F9 39         |CMP CL,39                     ; 문자 값이 '9'(ASCII 0x39)보다 큰지 확인
0040F8F0  |. 7F 4E           |JG SHORT INSTALL.0040F940     ; 크면 실패 처리로 점프

0040F8F2  |. 0FBEC9         |MOVSX ECX,CL                  ; CL(문자) 값을 정수형으로 변환
0040F8F5  |. 8D3C00         |LEA EDI,DWORD PTR DS:[EAX+EAX] ; EDI = EAX * 2
0040F8F8  |. 83E9 30         |SUB ECX,30                   ; 숫자로 변환 (ASCII 값에서 0x30을 뺌)
0040F8FB  |. 33F9           |XOR EDI,ECX                   ; EDI = EDI XOR ECX
0040F8FD  |. 03C7           |ADD EAX,EDI                   ; EAX = EAX + EDI
0040F8FF  |. 42             |INC EDX                       ; 다음 문자로 이동 (EDX += 1)
0040F900  |. 83FA 0C        |CMP EDX,0C                    ; 12번째 문자까지 반복 확인
0040F903  |.^72 E0          \JB SHORT INSTALL.0040F8E5     ; 아직 12개 문자 처리 전이면 반복

0040F905  |. 33D2           XOR EDX,EDX                  ; 나머지 계산 준비 (EDX = 0)
0040F907  |. B9 0A000000    MOV ECX,0A                   ; 나눌 값 10 설정
0040F90C  |. F7F1           DIV ECX                      ; EAX를 10으로 나눔. 몫(EAX), 나머지(EDX)

0040F90E  |. 0FBE46 0C      MOVSX EAX,BYTE PTR DS:[ESI+C]  ; 13번째 문자 읽기
0040F912  |. 0FBED2         MOVSX EDX,DL                 ; 나머지(EDX)를 정수로 변환
0040F915  |. 83C2 30        ADD EDX,30                   ; 나머지 값에 ASCII 변환(0x30 추가)
0040F918  |. 3BC2           CMP EAX,EDX                  ; 13번째 문자와 나머지가 일치하는지 비교
0040F91A  |. 74 1C          JE SHORT INSTALL.0040F938    ; 같으면 인증 성공
```

- **루틴 흐름**
  - 입력된 시리얼 키는 13자리이다.
  - 첫 12자리 문자는 특정 계산 과정을 거쳐 최종 결과를 도출한다.(`EAX`).
  - 마지막 13번째 문자는 계산 결과(`EAX % 10`)와 비교된다.
- **인증 성공 조건**
  - 첫 12자리의 계산 결과에서 도출된 나머지 값(0~9)이 13번째 문자와 정확히 일치해야 한다.

- 결론
  - **입력**: 13자리 시리얼 키
  - **검증**:
    1. 첫 12자리의 ASCII 값을 숫자로 변환 후 계산.
    2. 계산 결과를 10으로 나눈 나머지(`EDX`)가 13번째 문자와 일치해야 성공.
  - **결과**: 일치하면 성공(`EAX = 1`), 그렇지 않으면 실패.

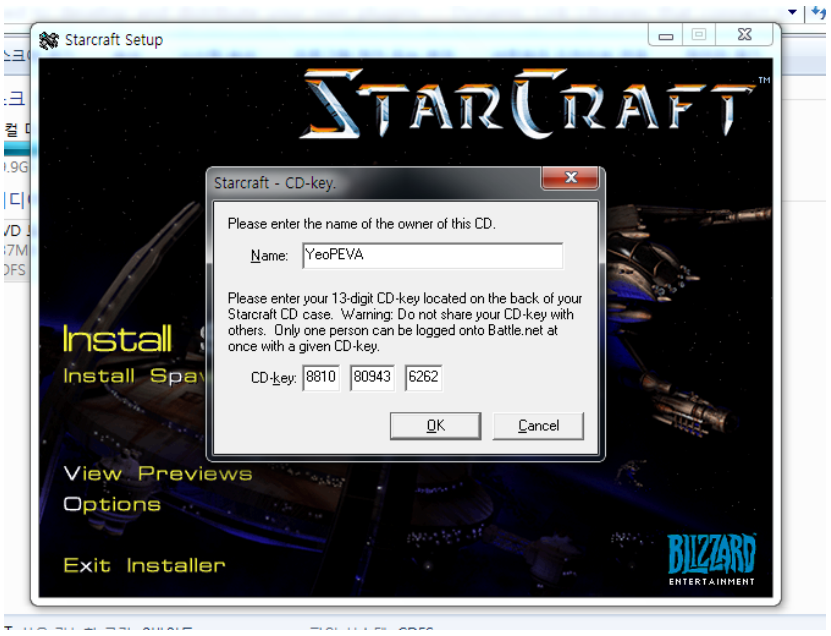
4. 취약점 검증

취약점을 검증하기 위하여, 위에서 분석된 내용을 기반으로, python 코드를 작성하여 시리얼 키 인증을 통과할 수 있는 시리얼 키를 제작하는 프로그램을 만들었다.

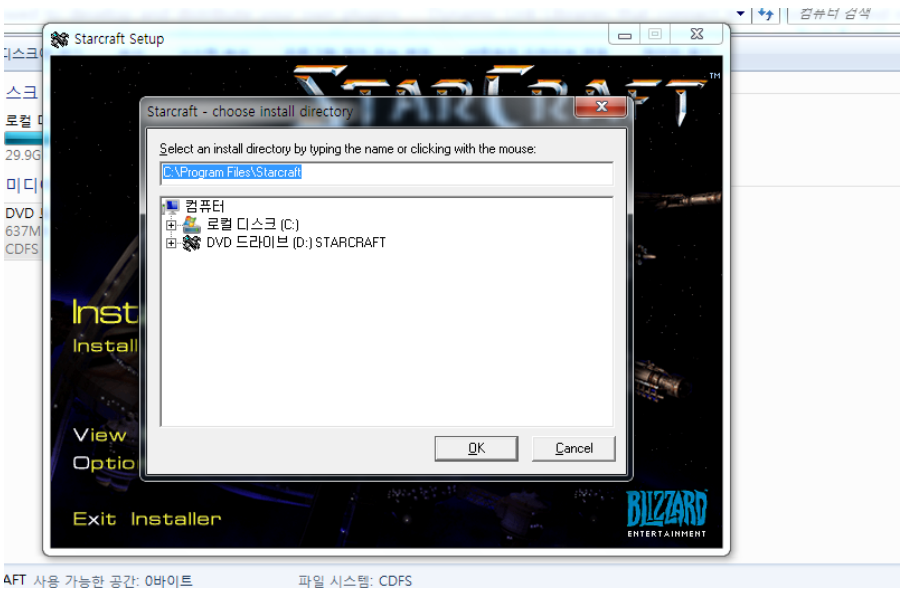
```
def generate_serial():
    import random

    # 12자리 숫자를 랜덤으로 생성
    serial_base = [random.randint(0, 9) for _ in range(12)]

    eax = 3  # 초기화
    for num in serial_base:
        ecx = num
        edi = eax + eax
        eax += edi ^ ecx

    # 나머지 계산
    remainder = eax % 10

    # 13번째 자리값 추가
    serial_base.append(remainder)

    # 시리얼 키 문자열로 반환
    return ''.join(map(str, serial_base))


if __name__ == "__main__":
    serial_key = generate_serial()
    print(f"Generated Serial Key: {serial_key}")
```

❯ python solve.py
Generated Serial Key: 4994854882323
❯ python
solve.py
Generated Serial Key: 8810809436262
❯ python
solve.py
Generated Serial Key: 5451556667087

그 결과, 위와 같은 시리얼 키를 확보할 수 있었으며, 해당 키 구문을 통해 인증을 시도하였다.



- 8810-80943-6262 (설치 시도)



- 인증 성공

5. 결론

타겟 분석 결과, 시리얼 키 인증 과정에서 별도로 인터넷 연결이 없어도 설치가 가능한 프로그램이기에, 시리얼 키 인증 로직을 프로그램 내에서 진행한다고 파악하였으며, 이를 분석하기 위해 back-to-user-mode를 활용하여 키 로직 근처로 이동 후, 분석을 함으로써 시리얼 키 검증 로직을 찾을 수 있었다.

찾아낸 키 검증 로직을 분석하여, 별도의 파이썬 코드를 작성한 후, 시리얼 키를 제작하고 이를 설치 프로그램에 실제로 활용함으로써 취약점을 검증할 수 있었으며, 이를 통해 시리얼 키를 생성하여 프로그램을 지속하여 인증할 수 있기에 해당 취약점 검증 결과는 활용이 가능하다.