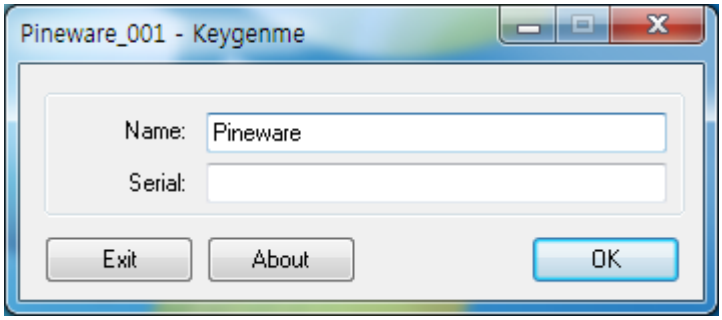
- 문제 -

Serial이 94E7DB1B 일때 Name은 무엇인가

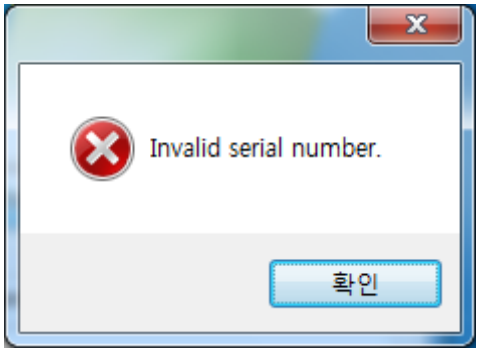
해당 Serial에 대한 정답이 여러개 나오는 문제이며 게시판에 비공개로 글을 남겨주시면 인증 처리해드리겠습니다.
해당 Serial에 대해서 "Serial accepted" 메시지가 나와야 합니다.

- 풀이 -



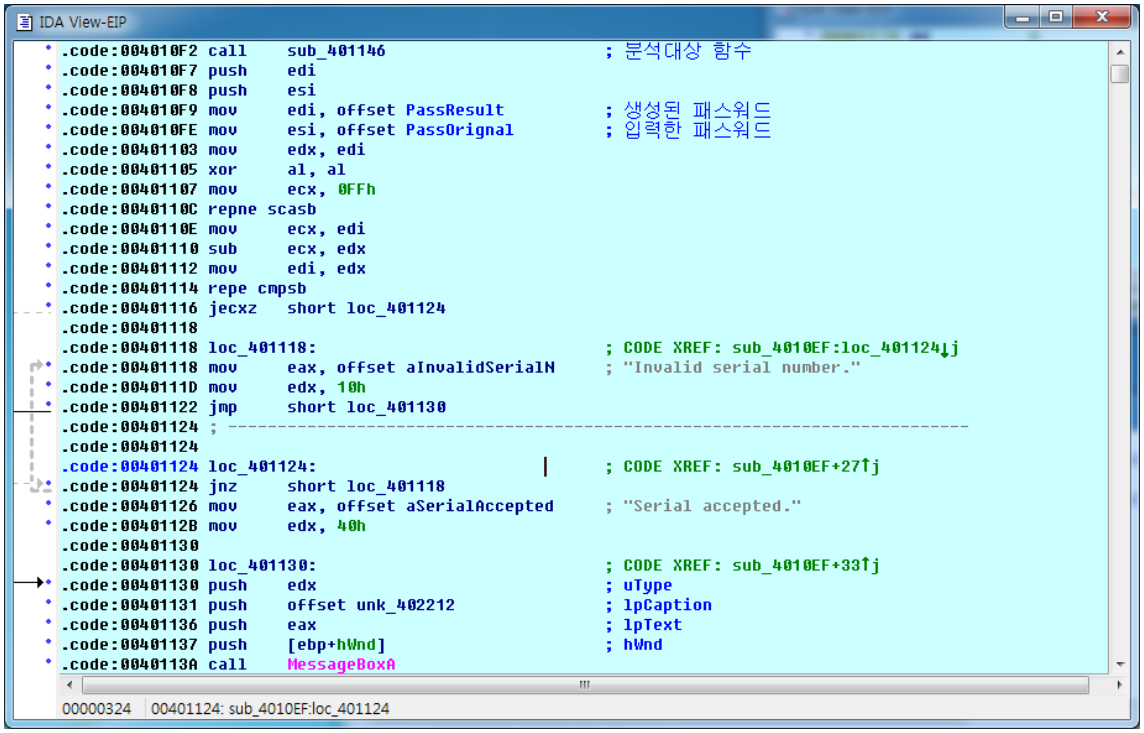
Serial을 보고 Name을 찾는 문제로 Name값을 통해 Serial을 생성할 것이라고 예상할 수 있다.

아무런 정보나 입력하니 경고메시지가 발생한다.

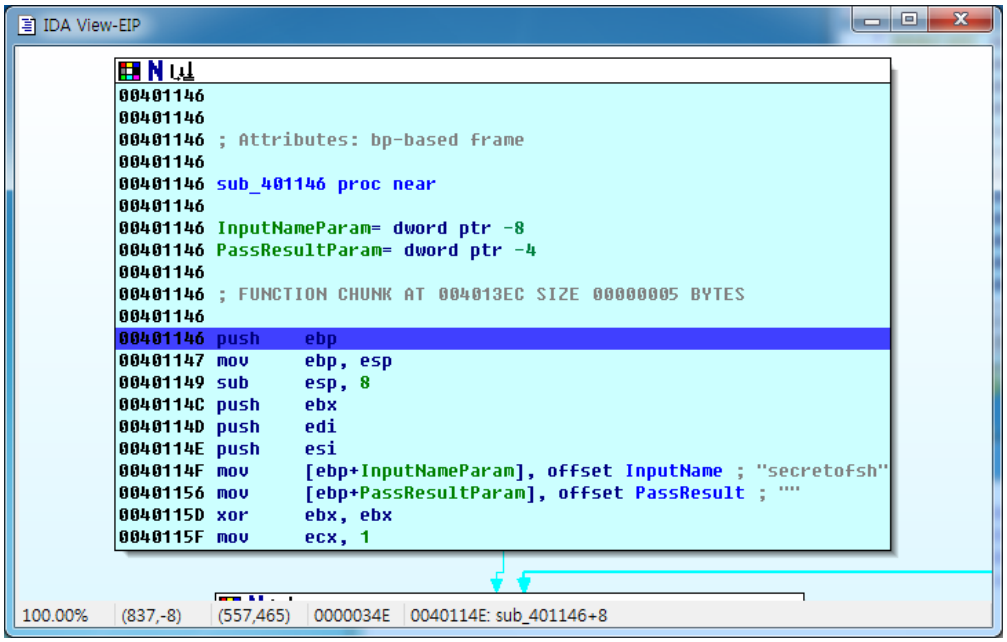


IDA를 통해 발생한 경고메시지를 추적하여 디버깅을 시작한다.

아래 그림에서 보는바와 같이 00401116에서 성공과 실패로 분기하는 것을 알 수 있다. 상단에 존재하는 함수들과 파라미터를 스택을 통해 분석해본 결과 004010FE에서 입력한 패스워드가 보였으며 바로 위 004010F9에서 패스워드로 추정되는 값이 보였다. 8자리의 16진수로 이루어져 있다. 두 개의 값들을 서로 비교하여 성공과 실패로 분기하는 것을 확인할 수 있다.

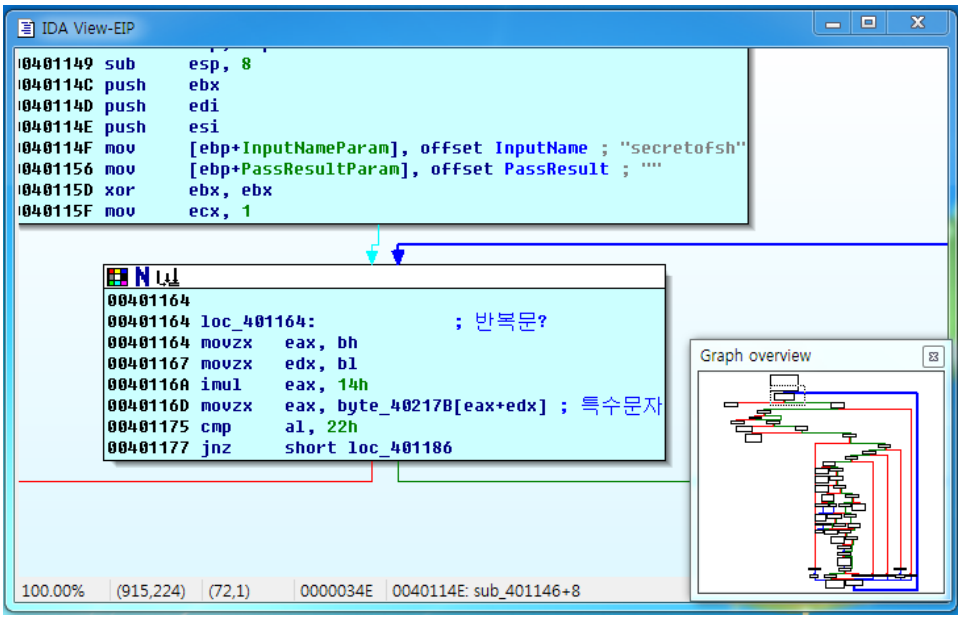


임의로 입력한 Name값에 004010F9에서 보이는 패스워드를 입력하였을 때 인증이 성공하였다. 스크롤을 좀 더 올려보니 GetDlgItemTextA 함수가 보인다. 탐지결과 해당 함수를 통해 입력한 값들을 버퍼로 가져온 뒤 정상적인 입력인지여부를 가볍게 검사한 후 004010F2에서 호출하는 함수로 향하는 것을 확인하였다. 004010F2에서 호출되는 함수가 분석대상임을 알 수 있다.



함수의 시작 부분을 보면 알 수 있듯이 두 개의 파라미터가 존재한다. PassResult는 해당 함를 호출한 영역에서 발견한 패스워드의 결과에 대한 버퍼의 주소를 말하며 InputName은 Name값에 할당한 이름을 말한다. Name에는 "secretofsh", Serial에는 "12341234"를 입력하였다.

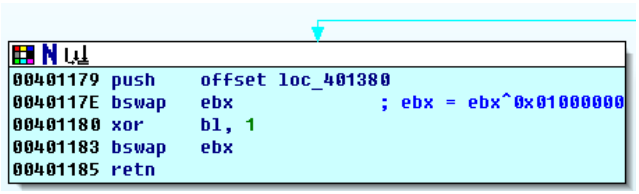
InputName을 통해 변경과정을 거친 후 PassResult에 결과에 대한 패스워드가 대입될 것이다. 차근차근 분석해가며 소스코드를 도출하고자 한다.

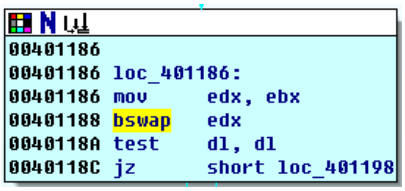


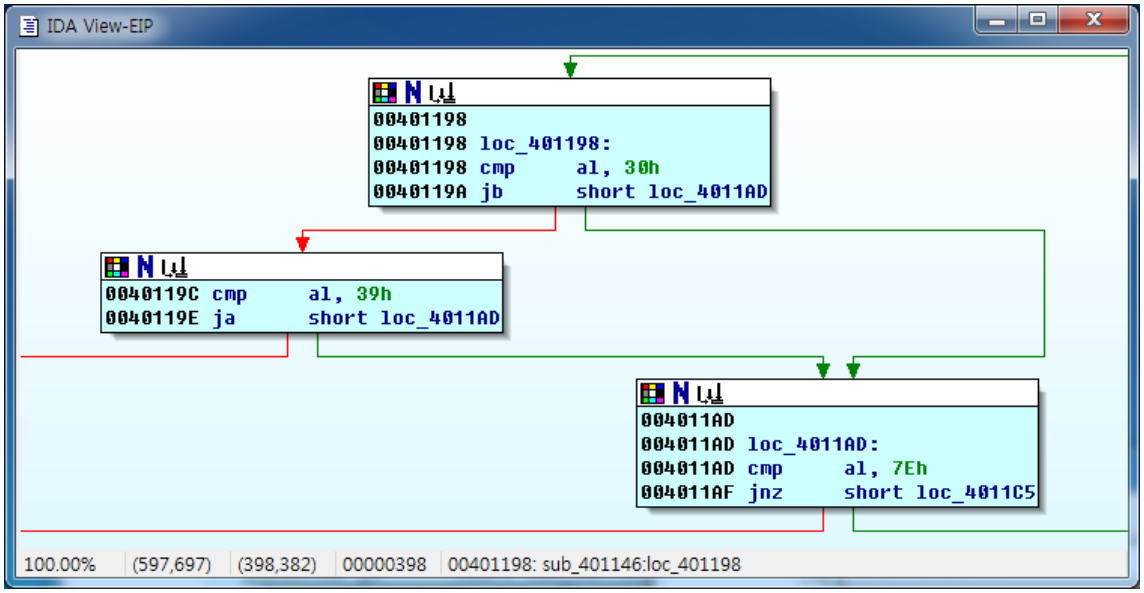
0040116D에서는 40217B위치의 암호화에 활용할 특정문자의 위치를 참조하여 eax로 가져오고 있는데 해당 문자열은 다음과 같다.

```
$5%$#wv#_"as"*3$$v+8v*"~x"<\^`\"9":<;x"->~:!4#v$_"md8"G+*%*+I:!#@_>87+/\"0"^v>,+
```

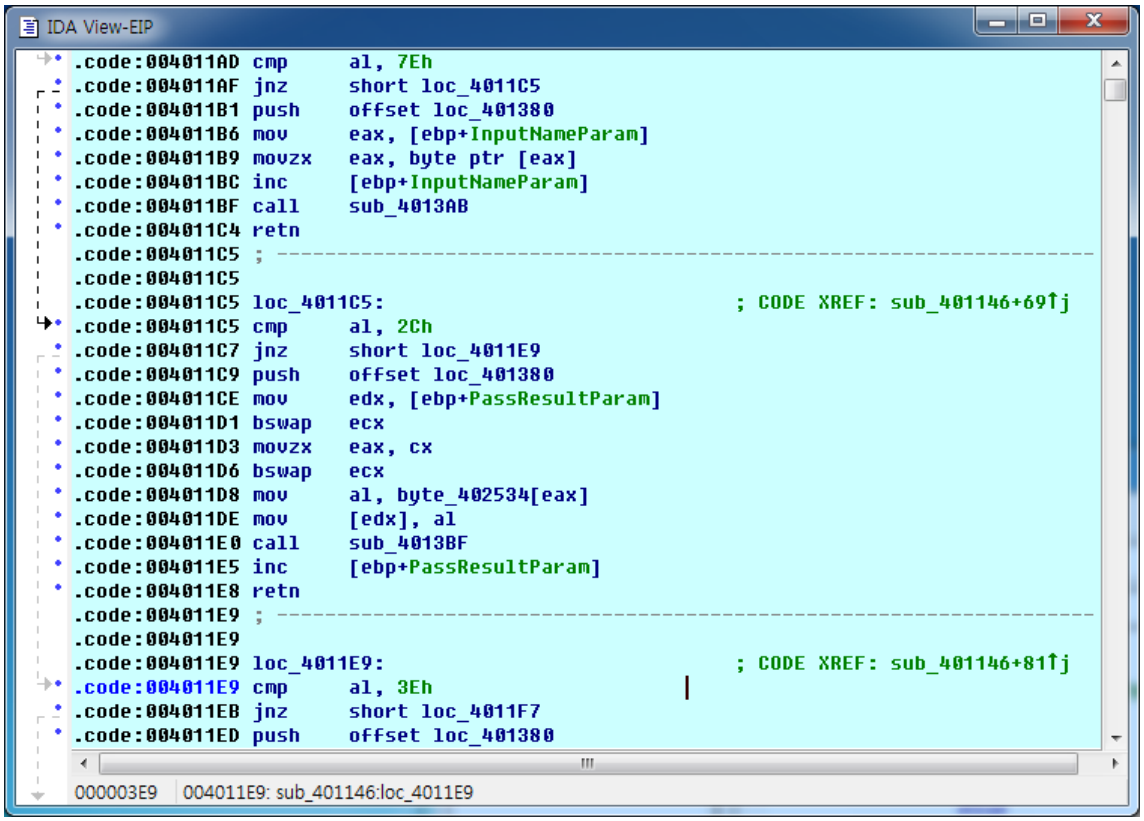
가져온 하나의 문자의 값을 비교하여 분기하는 코드는 다음과 같다.





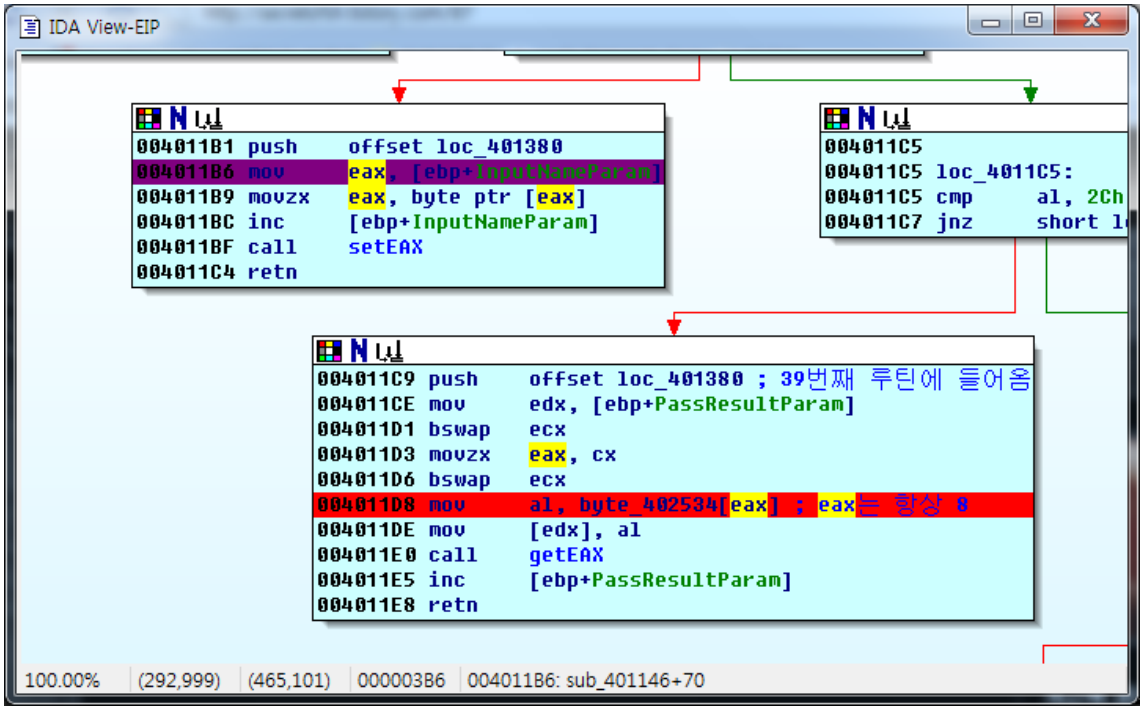


4011AD를 기점으로 거짓으로 갈때는 대부분 401380을 push하고 retn하여 401380으로 이동하는것을 확인하였다. 참으로 갈때에는 4011AD를 시작으로 al값에 의해 지속적으로 비교하며 분기하는것을 확인할 수 있다. 즉 switch과 같은 분기를 말한다.



어셈블러로 작성된 것인지 ah, al, bh, bl 요런 레지스터의 사용 빈도가 높았다.

반복분의 시작부분에 존재하는 40116D에서 결정한 eax값에 의해 다양한 방향으로 분기하는 구조를 갖고 있다.



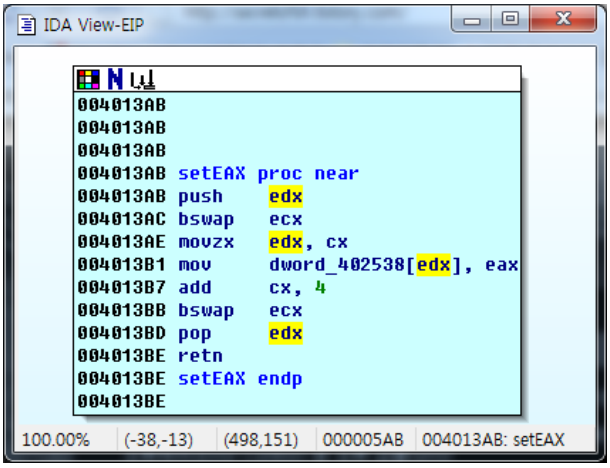
004011B9에서 입력한 Name의 한글자를 가져오는 것을 확인할 수 있다.
004011D8에서는 어떤 과정을 거쳐 생성된 값을 패스워드로 가져오는 것을 확인할 수 있다.

두 위치에 브레이크문을 걸고 run을 해보았다. 004011B9 구문을 입력한 Name의 길이+1만큼 지나갔으며 이후 004011D8구문은 패스워드의 길이인 8+1만큼 지나갔다. 이름을 통해 생성된 특정 값을 생성한 후에 패스워드를 생성한다는 것을 알 수 있다.

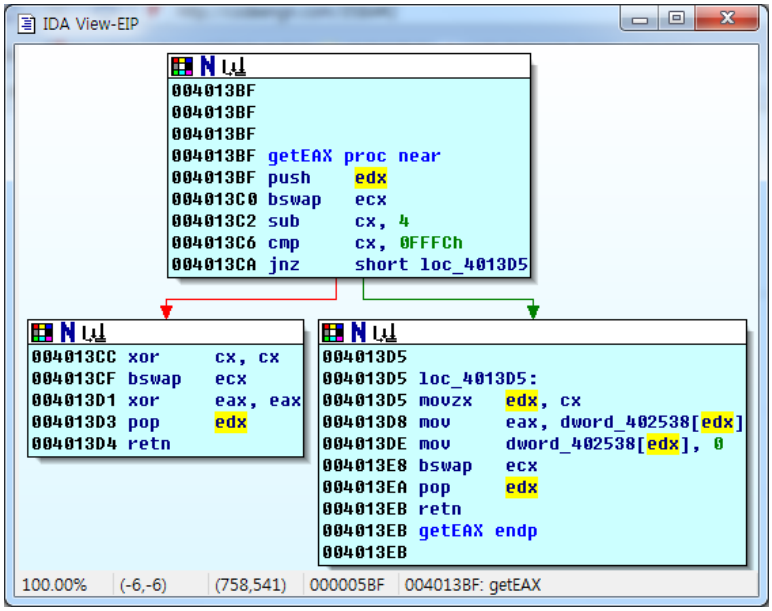
eax값에 의해 분기하는 4011677에 브레이크를 추가로 걸고 확인해본 결과 일정한 패턴으로 004011B1과 004011C9를 진입하는 것을 확인할 수 있었다. 이를 통해 4011677에서 eax값이 일정하게 변화된다는 것을 예측할 수 있었다.

모든 분기문에서 setEAX(004013AB), getEAX(004013BF) 함수를 주기적으로 호출하는 것을 확인할 수 있다. setEAX함수에서는 402538의 메모리의 특정 위치에 eax값을 삽입하며, getEAX에서는 402538의 메모리공간 특정 위치의 값을 eax로 가져오는 역할을 한다.

분석과정중 알아낸 사실은 setEAX는 eax값을 스택에 push를 하며, getEAX에서는 스택에서 eax로 pop을 하는 역할을 한다. 여기서 스택 메모리 시작 주소는 아래 그림에서 보이는 바와 같이 402538을 말한다.



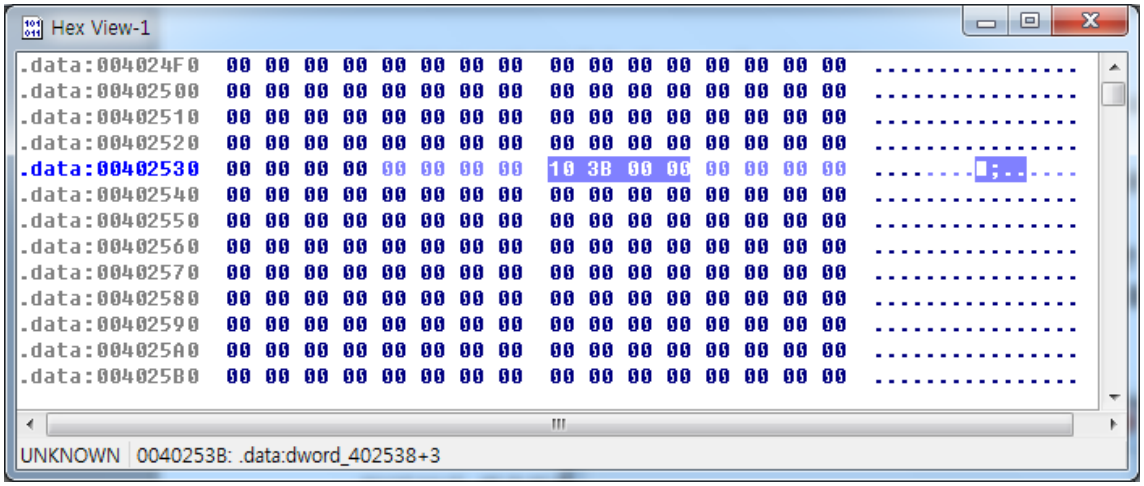
[그림 11] setEAX 함수



[그림 12] getEAX 함수

앞서 확인한 00401177분기점, 004011B9 Name값 가져오는 지점, 004011D8 Password 삽입 지점 3부분에 브레이크 포인트를 걸고 디버깅을 시작하였다.

00401177에서 eax값을 기록하며 getEAX, setEAX에 의해 변화시키는 402538메모리를 HexView를 통해 변화를 살펴보면서 디버깅을 하였다.
최초 004011B9에 도달할 때 메모리는 아래 그림과 같이 항상 일정했다. 즉, 3B10의 값을 할당한 상태에서 이름값을 통해 지속적으로 변화시킨다.



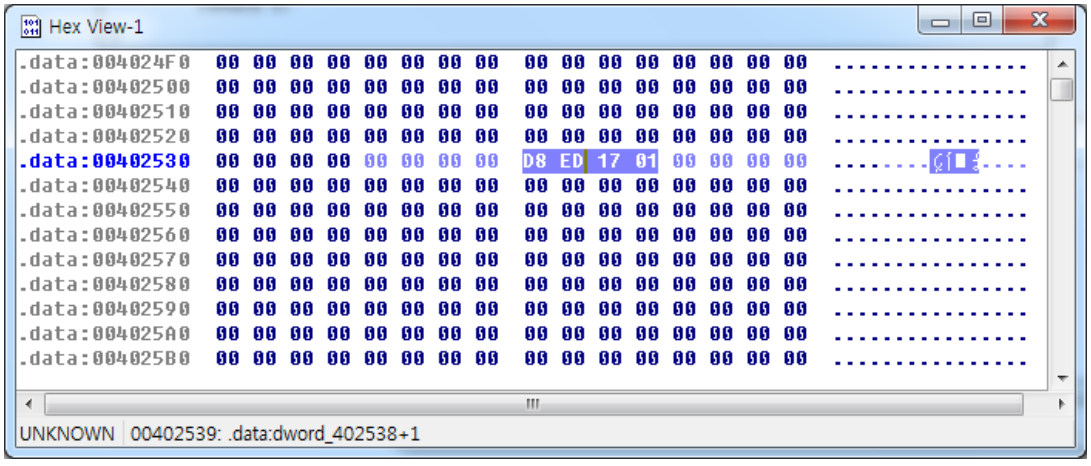
3B10이 삽입된 이후 eax의 값의 변화를 살펴보며 동작하는 기능을 기록하였다. 기록하는 도중 getEAX함수에서는 402538 메모리에 최근 setEAX함수를 통해 set한 값이 없으면 0을 반환하고 최근에 setEAX함수를 통해 set한 값이 있으면 해당 값을 eax로 반환하는 것을 확인하였다. setEAX함수에서는 402538메모리 영역에 스택처럼 dword단위로 값을 삽입하는 루틴을 갖는다.

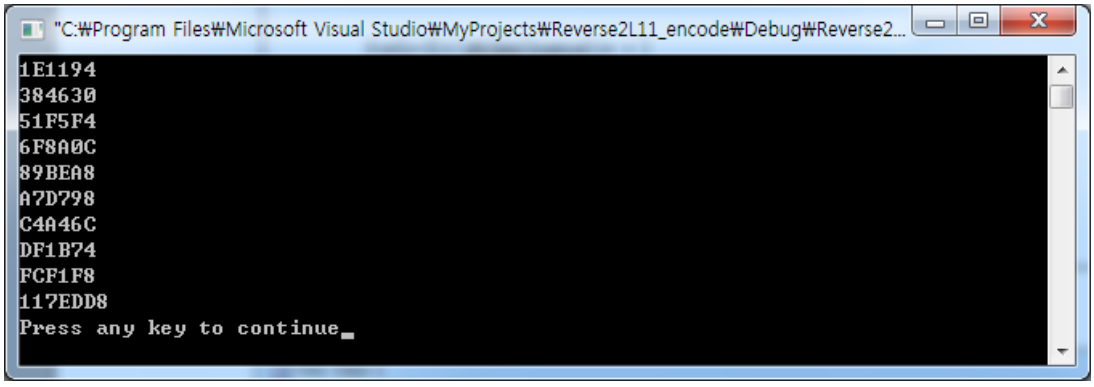
eax ; 설명
[3A] ;Name의 첫글자 삽입(secretofsh입력한 글자중 s의 패턴), Name주소1 증가
[21] ;[3A]에서 삽입한 값이 NULL인지 확인하며 EDX에 삽입:특정영역에 0일경우 1 삽입;
[34] ;al에서 30을 제외, 04의 값 삽입
[23] ;bx값만 바꿔 흐름만 제어
[24] ;[34]에서 할당한 값(04)을 가져옴
[5F] ;[21]에서 삽입한 값(0)을 가져옴, cx를 1로 초기화
[22] ;ebx후미의 1bit xor
[6D] ;6D를 삽입함
[64] ;64를 삽입함
[38] ;38를 삽입함
[22] ;ebx후미의 1bit xor
[47] ;동작없음
[2B] ;[38]에서 삽입한 0x38가져옴, [64]에서 삽입한 0x64가져옴, 0x38과 0x64를 더한 0x9C 삽입
[2A] ;삽입된 값 0x9C를 가져옴, 삽입된 값 0x6D를 가져옴 0x6D에 0x9C를 곱한뒤 삽입(set:0x426C, 메모리:0x6C42)
[25] ;동작없음
[2A] ;0x426C 가져옴, 0x426C앞에 삽입된 Name의 첫글자 값(0x73) 가져옴, 0x426C에 이름의 첫 글자를 곱한 뒤 삽입(set:0x1DD684, 메모리:0x84D61D)
[2B] ;삽입된 값 0x1DD684 가져옴, 최초 삽입되어있던 0x3B10(메모리상103B)값 가져옴, 두 개의 값을 더한 뒤 삽입(set:0x1E1194, 메모리:0x94111E)
[3E] ;cx = 1
[7E] ;//이름의 한글자를 가져와 set함-----이후 다시 [3A]로 반복

위 절차에서 불필요한 부분을 제거하여 코드로 구현하면 다음과 같다.

```
#include <stdio.h>
#include <string.h>

int main()
{
	char name[]="secretofsh";
	char pass[10];
	int buf=0x3B10;
	int i;

	for(i=0;i<strlen(name);i++)
	{
		buf = buf + name[i] * 0x426C;
		printf("%.0X\n",buf);
	}
	return 0;
}
```

이름값을 통해 구해진 402538 메모리의 상태는 다음과 같다.





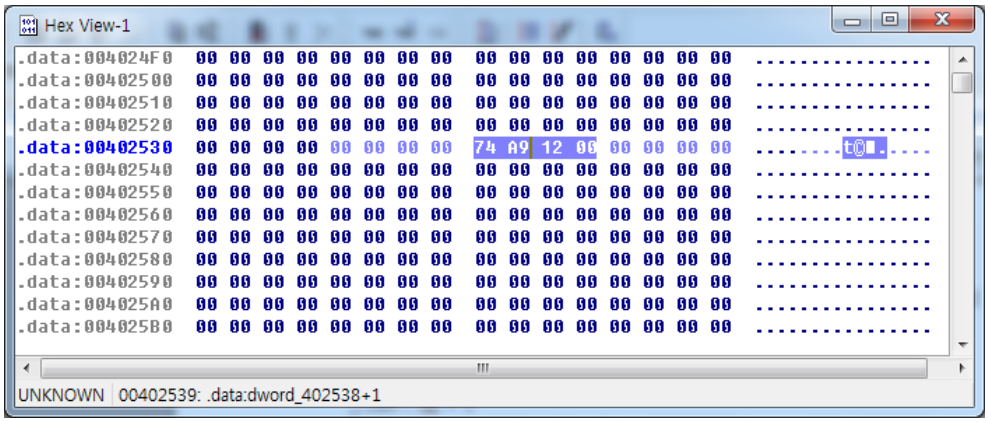
이후 004011DE에서 패스워드로 삽입될 때까지의 패턴을 분석하여 다음과 같이 정리하였다. 004011D8에서는 항상 402534메모리의 8번째 해당하는 byte값을 패스워드에 할당하는 것을 확인하였다. 402534메모리의 8번째 주소인 40253C의 값을 유심히 관찰하도록 한다.

Name값의 NULL문자 이후 첫 번째 패스워드를 입력할 때 까지의 패턴은 아래와 같다.

eax ; 설명
[3A] ;Name의 끝 NULL문자 삽입
[21] ;[3A]의 문자가 NULL이기 때문에 1삽입
[34] ;al에서 0x30을 제외, 특정위치에 항상 0x04가 대입됨
[23] ;bx 값 바꿔 흐름만 제어
[24] ;[34]에서 할당한 값을 가져옴
[5F] ;[21]에서 삽입한 값을 가져옴, cx =0xFF
[24] ;[3A]에서 삽입한 Name의 NULL문자를 가져옴
[76] ;cx = 0x100
[3E] ;cx = 1
[38] ;al -=30, eax값 8을 삽입
[37] ;al -=30, eax값 7을 삽입
[2B] ;[37]에서 삽입한 7과 [38]에서 삽입한 8을 가져와 더한 0xF값 삽입
[2F] ;[2B]에서 삽입한 0xF값 가져옴, 가져온 값이 0이 아니라면 [3A]전에 메모리에 삽입되어 있는 값을 가져온 값으로 나눈다.(117EDD8/F = 12A974) 나머지인 edx값(0xC)를 삽입, 앞서나눈 값(0x12A974) 삽입
[5C] ; [2F]에서 삽입한 두 개의 값의 위치를 바꿈
[22] ; ebx의 상위바이트를 1과 xor
[30] ; 30삽입
[22] ; ebx의 상위바이트를 1과 xor
[5E] ; cx = 0xFF00
[2B] ; [30]에서 삽입된 30과 [2F]에서 삽입된 0xC 값을 가져와 더한뒤 삽입(0x3C)
[3C] ; cx = 0xFF;
[3A] ; [2B]에서 삽입된 값(0x3C)을 한번 더 삽입
[22] ; ebx의 상위바이트를 1과 xor
[39] ; 0x39 삽입
[22] ; ebx의 상위바이트를 1과 xor
[5C] ; 삽입된 두 개의 값을 바꿈([39]와 [3A]에서 삽입한 0x39와 0x3C의 위치를 바꿈)
[60] ; 0x3C를 가져옴, 0x39를 가져옴, 비교하여 앞의 값이 클경우 eax=1, 작거나 같을경우 eax=0 eax삽입
[5E] ; cx=0xff00
[5F] ; [60] 에서 삽입한 값(1) 가져옴, [5C]에서 위치가 바뀌면서 삽입된 값(0xC) 가져옴, 0값과 비교후분기, cx=0xFF;
[23] ; BX값 셋팅
[77] ; ..al과 40비교후 점프
[23] ; BX값 셋팅
[25] ; ..al과 40비교후 점프
[35] ; 35에서 30을 뺀 값(0x5)을 삽입
[24] ; 35에서 삽입한 0x5값 가져옴
[38] ; 38에서 30을 뺀 값(0x8) 삽입
[2B] ; 상위의 [2B]에서 삽입된 0x3C값과 [38]에서 삽입된 0x8값을 가져와 더한 값(0x44)을 삽입
[76] ; cx= 0x100
[78] ; al과 40 비교
[25] ; al과 40 비교
[3E] ; cx = 1

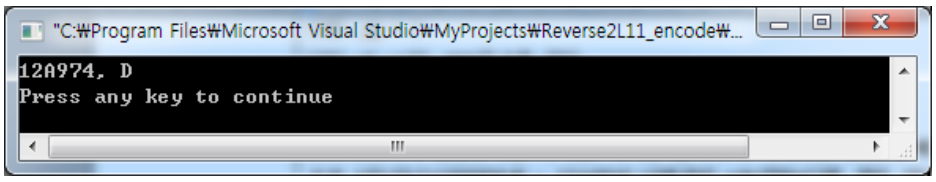
(2C) : [2B]에서 생성된 0x44값을 가져와 패스워드 버퍼에 등록

위 과정이 수행된 이후의 402538메모리의 값은 다음과 같다.

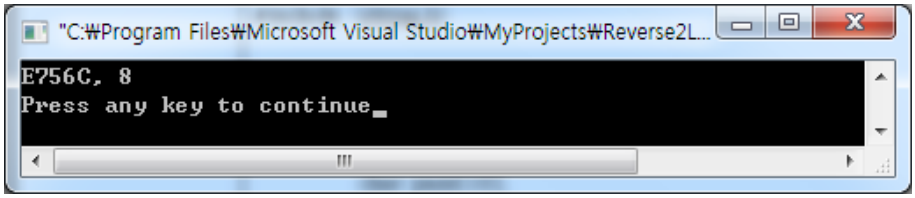


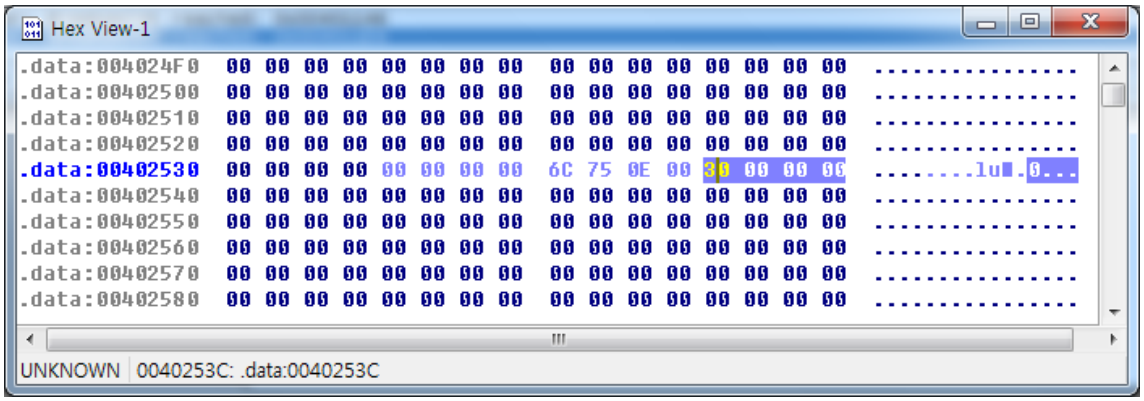
분석된 부분을 추가하여 불필요한 부분을 제거하여 코드로 구현하면 다음과 같다.

```
#include <stdio.h>
#include <string.h>

int main()
{
	char name[]="secretofsh";
	char pass[10];
	int buf=0x3B10;
	int i, mod=0;

	for(i=0;i<strlen(name);i++)
	{
		buf = buf + name[i] * 0x426C;
	}

	mod = buf%0xF;
	buf/=0xF;
	pass[0] = 0x30+mod+0x8;

	printf("%.0X, %c\n",buf, pass[0]);

	return 0;
}
```



검증을 위해 이름을 codeengn으로 변경하여 동일한 결과가 나오는지 비교해 보았다.





첫 번째 메모리 영역에 삽입된 0x0E756C값은 동일하였으나 패스워드에 삽입되는 다음 메모리영역 값은 0x30('0')으로 동일하지 않았다.
앞서 분석한 결과에서 붉은색으로 표시한 부분이 2가지의 패턴으로 갈라지는 것을 확인하였다.

붉은색 부분 상위 [2B]에서 가져온 값(0x3C)이 0x39보다 클 경우 조사한 패턴대로 흘러가며 작거타 같을 경우 붉은색 부분은 아래의 내용으로 흘러간다. eax값이 0x60일때의 값에 의해 영향을 받기 때문에 분석된 내용의 붉은색 부분의 흐름을 결정하는 [60]의 위치부터 조사하였다.

[60] ; 삽입된 0x30과 0x39를 가져온뒤 비교하여 앞의 값이 클경우 eax=1, 작거나 같을경우 eax=0 eax삽입
[5E] ; cx = 0xFF00
[5F] ; [60] 에서 삽입한 값(0) 가져옴, 5C에서 위치가 바뀌면서 삽입된 값(0x0) 가져옴, 0값과 비교후분기, cx=1;
[22] ; ebx의 상위바이트를 1과 xor
[61] ; 0x61삽입
[73] ; 0x73삽입
[22] ; ebx의 상위바이트를 1과 xor
[2A] ; [73]에서 삽입된 0x73과 [61]에서 삽입된 0x61을 가져와 곱한 값(0x2B93)을 다시 삽입
[33] ; al값 0x33에서 0x30을뺀 0x3값을 삽입
[24] ; [33]에서 삽입된 0x3값 가져옴
[24] ; [2A]에서 삽입된 0x2B93 가져옴
[2B] ; 상위의 [2B]에서 삽입된 0x30값과 0x0값을 가져와 더한 값(0x30)을 삽입
[76] ; cx= 0x100

```
[78] ; al과 40 비교
[25] ; al과 40 비교
[3E] ; cx = 1
[2C] ; [2B]에서 생성된 0x30값을 가져와 패스워드 버퍼에 등록
```

60에서 결정된 eax값에 의해 붉은색 부분이 [22]와 [23]두가지 유형으로 갈렸다. 불필요한 부분을 제거하고 두가지 유형을 고려하여 코드를 구현하면 다음과 같다.

```
#include <stdio.h>
#include <string.h>

int main()
{
	char name[]="secretofsh";
	char pass[10];
	int buf=0x3B10;
	int i, mod=0;

	for(i=0;i<strlen(name);i++)
	{
		buf = buf + name[i] * 0x426C;
	}

	mod = buf%0xF;
	buf/=0xF;

	pass[0] = 0x30+mod;
	if(pass[0]>0x39)
	{
		pass[0]+=0x8;
	}

	printf("%.0X, %c\n",buf, pass[0]);

	return 0;
}
```

패스워드 8자리중 첫 번째 자리는 구해냈다. 이제 나머지 7자리의 패턴을 분석하여 코드를 완성시키도록 한다. 분석 결과 첫 번째 자리를 삽입한 뒤의 패턴은 첫 번째 패스워드를 구하는 패턴을 분석해 놓은 자료의 노란색 음영으로 표시해 놓은 부분은 동일하였으며 상단 일부만 달랐다.

아래 분석된 부분 이후 노란색 음영부분을 연결하면 나머지 7자리의 패스워드가 완성된다.
현재 위 소스코드의 buf에 해당되는 값만 402538메모리에 삽입되어 있는 상태이다.

```
[2B] ;삽입된 값(buf값)을 가져와 0을 더한뒤 다시 삽입
[6C] ;삽입된 값(buf값)을 가져와 좌측으로 1bit 회전시킨뒤 다시 삽입
후미 1byte가 0x60이면 삽입된 값을 가져와 0과 비교한 뒤 0보다 크면 1삽입, 0보다 작거나 같으면 0삽입,
```

```
이후 al이 0x24이면 삽입된 값을 가져오고 0x24가 아니면 분기
[3A] ;삽입된 값을 가져온 뒤 다시 두 번 삽입
[21] ;삽입된 값을 가져와 0이면 1삽입, 아니면 0삽입
[23] ;BX 갱신
[5F] ;21에서 삽입된 값을 가져와 비교, 0이아니면 cx=1, 0이면 cx=0xFF
```

위 분석된 내용과 앞서 분석한 노란색 음영부분을 연결하여 코드를 완성하면 다음과 같다.

```
#include <stdio.h>
#include <string.h>

int main()
{
	char name[]="secretofsh";
	char pass[10]={0,};
	int buf=0x3B10;
	int i, mod=0;

	for(i=0;i<strlen(name);i++)
	{
		buf = buf + name[i] * 0x426C;
	}

	for(i=0;i<8;i++)
	{
		mod = buf%0xF;
		buf/=0xF;

		pass[i] = 0x30+mod;
		if(pass[i]>0x39)
		{
			pass[i]+=0x8;
		}
		buf<<=1;
	}
	printf("%.0X, %c, %s\n",buf, pass[0],pass);

	return 0;
}
```

테스트 결과 올바른 패스워드가 도출되는 것을 확인하였다.

위 결과를 분석하여 패스워드를 통한 Name값을 가져올 수 있도록 다음과 같이 코드를 작성하였다.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
int main()
{
```

```
	char name[20];
	char pass[10]="94E7DB1B";
	int i, ch, modmod, mod=0;

	int sbuf = 0;
	int temp;

	srand(time(0));
	for(i=7;i>=0;i--)
	{
		//ror
		temp=sbuf&1;
		temp<<=31;
		sbuf>>=1;
		sbuf|=temp;

		if(pass[i]>0x41)
		{
			mod = pass[i]-0x38;
		}
		else
		{
			mod = pass[i]-0x30;
		}

		sbuf*=0xF;
		sbuf+=mod;
	}

	//최초 buf=0x3B10
	//for(i=7;i>0;i--)
	//{
	//	buf = buf + name[i] * 0x426C;
	//}
	//sbuf = 0x3B10+name[0]*0x426C +name[1]*0x426C +name[2]*0x426C
+name[3]*0x426C +name[4]*0x426C +name[5]*0x426C;
	//(sbuf - 0x3B10)/0x426C = name[1]+name[2]+name[3];

	temp = (sbuf-0x3B10)/0x426C;//모든 문자의 총 합이 temp값이면 인증 성공
	//10글자인 패스워드 추출

	for(i='a';i<='z'+1;i++)
	{
		if(temp/i<=14&&temp/i>=5)
		{
			mod = temp%i;
			break;
		}
	}
	if(i=='z'+1)
	{
```

```
                printf("Name 생성 실패");
                return -1;
        }
        ch = i;
        modmod = mod/(temp/ch-rand()%5);

        for(i=0;i<temp/ch;i++)
        {
                if(mod!=0)
                {
                        if(mod < modmod)
                        {
                                name[i] = ch+mod;
                                mod = 0;
                        }
                        else
                        {
                                name[i] = ch+modmod;
                                mod-=modmod;
                        }
                }
                else
                {
                        name[i] = ch;
                }
        }
        name[i]=0;

        printf("Name: %s\n",name);

        return 0;
}
```

- 결과 -