

SecureSoftware

Đọc file Readme.txt, ta hiểu được đề yêu cầu tạo một key phù hợp bằng flag -i, sau đó chạy chương trình.

```
C:\Users\BILL\Desktop\Release_3\SecureSoftware>.\SecureSoftwarev1.5.exe -i  
Enter the Key: abcd
```

Mở chương trình với IDA.

```

1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     HANDLE hHandle; // [esp+0h] [ebp-10h]
4     CHAR *lpDst; // [esp+4h] [ebp-Ch]
5
6     sub_402480();
7     lpDst = (CHAR *)malloc(0xFFu);
8     ExpandEnvironmentStringsA("%USERPROFILE%", lpDst, 0xFFu);
9     SetCurrentDirectoryA(lpDst);
10    free(lpDst);
11    beginthread(StartAddress, 0, (void *)1);
12    if ( IsDebuggerPresent() )
13        word_404024 = 1;
14    hHandle = (HANDLE)beginthread(sub_401BD4, 0, 0);
15    sub_4021C9(main);
16    if ( argc > 1 )
17    {
18        word_407034 = 1;
19        if ( !strcmp(argv[1], "-i") )
20        {
21            sub_401874();
22        }
23        else if ( !strcmp(argv[1], "-u") )
24        {
25            sub_40174F();
26        }
27        else
28        {
29            sub_401781(off_40400C[0]);
30            MessageBoxA(0, Text, "Error undefined arg", 0x30u);
31        }
32        exit(0);
33    }
34    sub_4017BC();
35    WaitForSingleObject(hHandle, 0xFFFFFFFF);
36    return 0;

```

Đầu tiên ta nhận thấy chương trình đã bật antidebug 2 lớp, lớp thứ nhất là hàm IsDebuggerPresent() và lớp thứ 2 là tại thread StartAddress

Hàm StartAddress này antidebug dựa trên thời gian

```

1 void __cdecl StartAddress(void *a1)
2 {
3     int v1; // eax
4     DWORD v2; // [esp+1Ch] [ebp-Ch]
5
6     if ( a1 == (void *)1 )
7     {
8         dword_407614 = GetTickCount();
9     }
10    else
11    {
12        v2 = GetTickCount();
13        v1 = v2 ^ dword_407614;
14        LOBYTE(v1) = 0;
15        if ( v1 )
16            exit(0);
17        dword_407614 = v2;
18    }
19 }

```

Tiếp theo ta để ý, nếu option -i được gọi thì chương trình sẽ gọi hàm sub_401874()

```

CreateDirectoryA(".\\data", 0);
v4 = time(0);
Stream = fopen(".\\data\\authdata.dat", "wb");
if ( !Stream )
{
    MessageBoxA(0, "File Access Error", "Error: Internal", 0x10u);
    exit(-2);
}
beginthread(StartAddress, 0, 0);
while ( !word_407036 )
;

```

```

sub_401841(off_404008, Buffer, 6);
sub_401841(off_404008, &Buffer[212], 6);
*(_DWORD *)&Buffer[8] = v4;
strcpy(&Buffer[12], "::Buffer");
strcpy(&Buffer[62], word_4076A0);
sub_401781(off_404014[0]);
printf("%s", Text);
scanf("%100[^\n]", &Buffer[112]);

```

Phần trên của hàm sub_401874(), đầu tiên chương trình tạo thư mục data, và sau đó là file authdata.dat, tuy nhiên thư mục này được tạo ở đường dẫn

“C:/User/<username>” do ở hàm main đã gọi hàm SetCurrentDirectoryA để thay đổi đường dẫn.

Phân tích mảng buffer độ lớn 0xDC, ta nhận thấy mảng buffer bao gồm:

0->6 “cv2pr”

8->12 time(0)

12->62 GetUserNameA

62->112 GetComputerNameA

112->212 Key(Nhập vào)

212->218 “cv2pr”

```
j
v1 = 0;
v6 = Buffer;
for ( j = 0; j <= 0xDB; ++j )
    v1 += (unsigned __int8)*v6++;
fwrite(Buffer, 0xDCu, 1u, Stream);
fclose(Stream);
Stream = fopen(".\\data\\checksum", "wb");
if ( !Stream )
{
    sub_40174F();
    MessageBoxA(0, "File Access Error", "Error: Internal", 0x10u);
    exit(-2);
}
fwrite(&v1, 4u, 1u, Stream);
```

Mảng buffer được lưu vào file authdata.dat, file checksum cũng được ghi tương tự và không quan trọng lắm, vì trong bài này chúng ta không cần đụng đến 2 file đó.

```
sub_40174F();
WaitForSingleObject(hHandle, 0xFFFFFFFF);
return 0;
```

Ở cuối chương trình có một hàm WaitForSingleObject(hHandle), gợi ý có thể luồng hHandle sẽ là nơi kiểm tra flag chứ không phải hàm main. Xem hàm sub_401BD4 gọi ở luồng hHandle

```

1 void __cdecl sub_401BD4()
2 {
3     sub_4021C9(sub_4017BC);
4     sub_4021C9(sub_401874);
5     sub_401B69();
6     if ( word_40703E )
7     {
8         if ( !word_407034 && word_404024 == 2 )
9         {
10            sub_4021C9(0);
11            sub_401F18();
12            if ( !dword_404044 && word_407038 == 1 && word_40703A == 1 && !word_40703C )
13            {
14                sub_401781(off_40401C[0]);
15                MessageBoxA(0, Text, &byte_40524E, 0x40u);
16                exit(0);
17            }
18        }
19    }
20 }

```

Đề ý chỗ MessageBoxA in ra chuỗi ở vị trí Text, mỗi lần chương trình muốn xuất kí tự gì đều gọi hàm sub_401781 để gán chuỗi cần in vào vị trí Text, ta xem xét hàm sub_401781(off_40401C[0])

```

1 int __cdecl sub_401781(unsigned __int8 *a1)
2 {
3     int result; // eax
4     CHAR *i; // [esp+Ch] [ebp-4h]
5
6     for ( i = Text; ; ++i )
7     {
8         result = *a1;
9         if ( !(_BYTE)result )
10            break;
11         *i = *a1++ - 1;
12     }
13     return result;
14 }

```

Đây là một hàm giải mã, với các kí tự đều được giảm 1. Xem xét đoạn chuỗi trong off_40401C[0] và tiến hành decode.

```
.data:0040401C off_40401C dd offset aTvddfftUiFQsph
.data:0040401C ; DATA XREF: sub_401BD4+94↑r
.data:0040401C ; "Tvddfft\"!Uif!qsphsbn!ibt!cffo!vompdl"...
```

Viết một script python để giải mã chuỗi, ta được

```
C:\Users\BILL\Desktop\Release_3\SecureSoftware>python get.py
Tvddfft"!Uif!qsphsbn!ibt!cffo!vompdl
Success!! The program has been unlock
```

Vậy đây chính xác là nơi chúng ta cần nhảy đến. Điều tiếp theo là quan tâm các điều kiện để nhảy đến vị trí đó.

- word_407034 = 0 (nếu không có flag “-i” hay “-u”)
- word_404024 = 2 (mặc định là 2 nếu không bị debug)
- word_40703E != 0 (=1 nếu đã tạo thư mục data và 2 file)

Tiếp theo nhờ cơ chế Xeft Graph To trong IDA, ta biết được các biến toàn cục trong điều kiện if sẽ được thay đổi ở hàm sub_401F18, được gọi ngay phía trên của lệnh if.

Xem xét hàm sub_401F18, đầu tiên chương trình đọc dữ liệu từ file authdata.dat và đưa vào chuỗi buffer, sau đó dựa vào các điều kiện cần như sau:

- dword_404044 == 0 (gán trong sub_4021C9 nếu chương trình không bị patch hay debug đồng thời sẽ gán luôn dword_404040 = 0)
- word_407038 == 1 (gán trong sub_401CB0 nếu buffer[0:6] == buffer[212:218] == "cv2pr" và dword_404040 != -1 (không patch hay debug))
- word_40703A == 1 (nếu username và computename đều khác “x”, nếu debug hay patch thì dword_404040 = -1 và username và computename sẽ bị đổi thành “x” trong hàm sub_401B69())
- word_40703C == 0 (dựa vào khóa sau hàm sub_401D0C)

3 điều kiện đầu chúng ta không cần tác động vì nếu chương trình chạy bình thường và file authdata.dat và checksum không bị sửa đổi thì các giá trị sẽ tự động được gán, chỉ duy nhất điều kiện word_40703C == 0 là chúng ta cần quan tâm.

Sau khi hàm sub_401D0C(Buffer) chạy sẽ thay đổi các giá trị của khóa, và chúng ta cần tất cả các giá trị đều = -1 sau hàm sub_401D0C để gán word_40703C = 0

```

sub_401D0C(Buffer);
v7 = 0;
for ( i = 0; i < v4; ++i )
{
    if ( Str[i] != -1 )
    {
        sub_401781((unsigned __int8 *)off_404018[0]);
        MessageBoxA(0, Text, &byte_40524E, 0x10u);
        v7 = 1;
        break;
    }
}
if ( v7 == 1 )
{
    word_407038 = 0;
    word_40703A = 0;
}
else
{
    word_40703C = 0;
}
}

```

Phần đầu của hàm sub_401D0C

```

if ( dword_407040 == 5 )
    strrev(String);
v4 = strlen(String);
for ( i = 0; i < v7; ++i )
    Buffer[i] %= 16;
for ( j = 0; j < v6; ++j )
    word_4076A0[j] %= 16;

```

Đầu tiên hàm sẽ chuyển tất cả các kí tự chuỗi Buffer và word_4076A0 thành các kí tự hex, vị trí các kí tự khóa sẽ bị đảo ngược nếu dword_407040 == 5, chúng ta có thể lần lại để tìm giá trị chính xác của dword_407040, tuy nhiên chúng ta nên chọn cách nhẹ hơn là thử cả 2 trường hợp thuận và đảo.

Buffer và word_4076A0 lần lượt là username và computename, được gán từ hàm sub_401F18

```
pcbBuffer = 100;  
GetUserNameA(::Buffer, &pcbBuffer);  
pcbBuffer = 100;  
GetComputerNameA(word_4076A0, &pcbBuffer);
```

Phần sau của hàm sub_401D0C


```

word_4076A0[] ^= 10;
for ( k = 0; ; ++k )
{
    result = k;
    if ( k >= v4 )
        break;
    v3 = sub_402190((unsigned __int8)String[k]);
    if ( k >= v6 )
    {
        if ( k >= v7 + v6 )
        {
            String[k] = 0;
        }
        else if ( v3 == Buffer[v7 - (k - v6) - 1] )
        {
            String[k] = -1;
        }
        else
        {
            String[k] = 0;
        }
    }
    else if ( v3 == word_4076A0[v6 - k - 1] )
    {
        String[k] = -1;
    }
    else
    {
        String[k] = 0;
    }
}

```

Dựa vào điều kiện so sánh, ta có thể viết script python để tìm ra giá trị khóa thích hợp, hàm sub_402190 chuyển các kí tự từ hex sang giá trị, yêu cầu các kí tự a->f viết hoa, chuỗi Buffer là username, word_4076A0 là computername

```

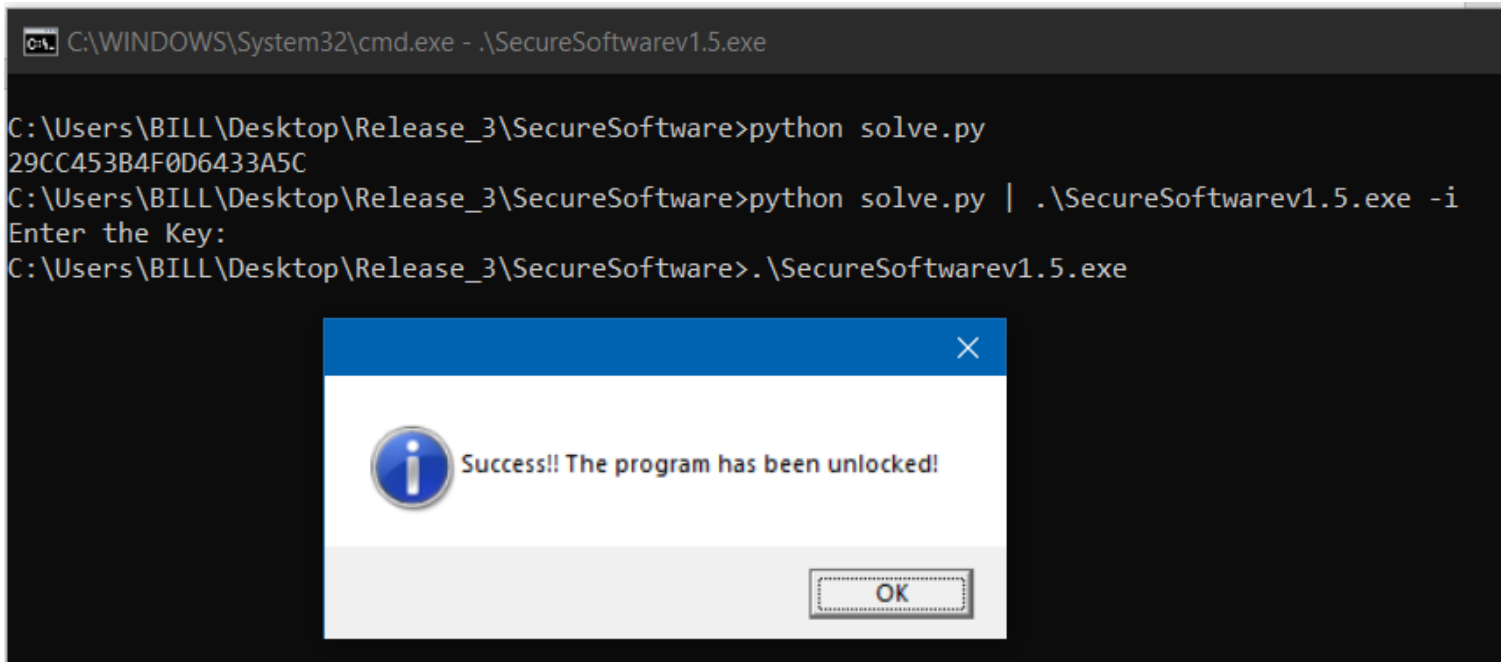
1 int __cdecl sub_402190(unsigned __int8 a1)
2 {
3     int v2; // [esp+10h] [ebp-4h]
4
5     if ( a1 > 0x46u )
6         return -1;
7     v2 = a1 - 48;
8     if ( v2 > 9 )
9         v2 = a1 - 55;
10    return v2;
11 }

```

Code python

```
username = 'BILL'
computername = 'DESKTOP-6TCCJEL'
a = [ord(i) % 16 for i in username + computername]
for i in a:
    print("%X" % i, end=' ')
```

Và kết quả



The screenshot shows a Windows command prompt window with the title bar "C:\WINDOWS\System32\cmd.exe - .\SecureSoftwarev1.5.exe". The command prompt shows the following commands and output:

```
C:\Users\BILL\Desktop\Release_3\SecureSoftware>python solve.py
29CC453B4F0D6433A5C
C:\Users\BILL\Desktop\Release_3\SecureSoftware>python solve.py | .\SecureSoftwarev1.5.exe -i
Enter the Key:
C:\Users\BILL\Desktop\Release_3\SecureSoftware>.\SecureSoftwarev1.5.exe
```

Overlaid on the command prompt is a Windows message box with a blue header bar and a close button (X). The message box contains an information icon (i) and the text "Success!! The program has been unlocked!". At the bottom right of the message box is an "OK" button.

Flag: 29CC453B4F0D6433A5C (với username = 'BILL' và computername = 'DESKTOP-6TCCJEL')