

Soksery Chan

11/4/2022

CECS 478 Sec 02

### 478 Malware Lab

I was able to find the hacking credits name of the professor by using the search function for text and string "anthonyg". Which led me to the address 100017e50 which previously have the string "hacked by anthonyg". I opened the Display Bytes in Ghidra to perform Hex editing. I realized that I cannot fit my full name into the same line as what the professor had previously so I abbreviated it to SokC. I changed the Hex to 61 6e 64 20 53 6f 6b 43 to get the string of "and SokC".

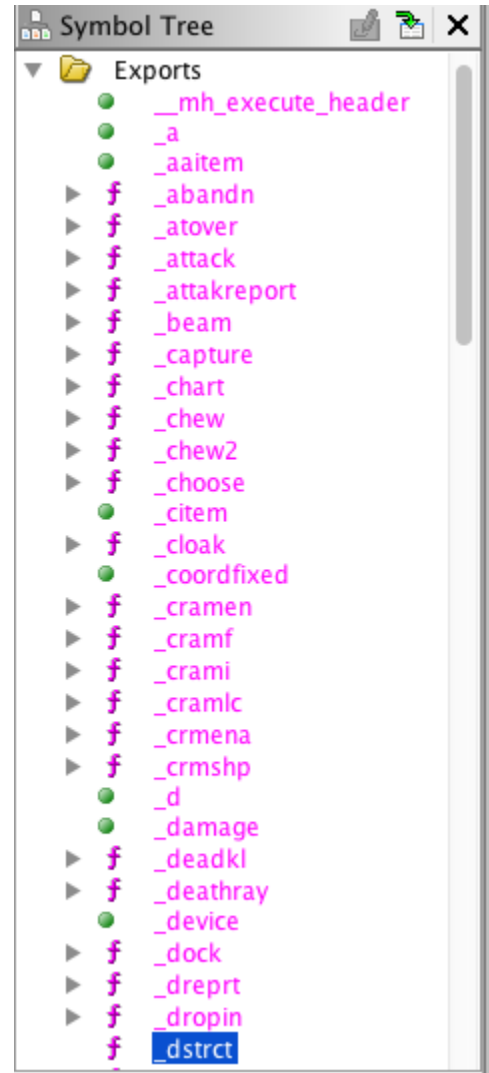
```
100017e50 68 61 63      s_hacked_by_anthonyg_and_SokC_100017e50  XREF[1]:  _pre
          6b 65 64      ds      "hacked by anthonyg and SokC"
          20 62 79 ...
100017e6c 6b          ??      6Bh      k
100017e6d 00          ??      00h
```

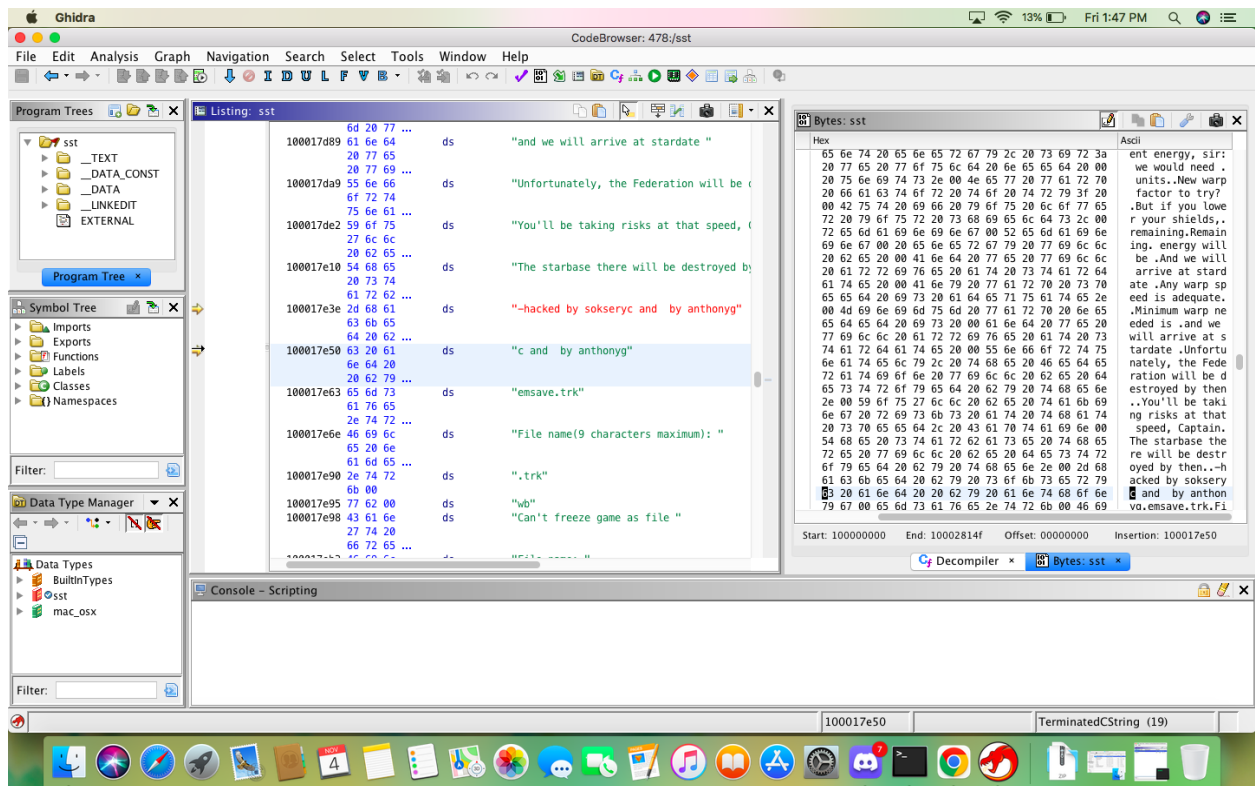
```
68 61 63 6b 65 64 20 62 79 20 61 6e 74 68 6f 6e | hacked by anthonyg and SokC.k.Fi
79 67 20 61 6e 64 20 53 6f 6b 43 00 6b 00 46 69 | le name(9 charac
6c 65 20 6e 61 6d 65 28 39 20 63 68 61 72 61 63 |
.. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
```

Next, I tried to find the self-destruct process of the code which took some time to see because I needed to learn how to navigate or know where everything was. After some time of navigating and some youtube tutorials, I was able to use Symbol Tree and opened the Exports folder to guide me to the \_dstruct function which I assumed had something to do with the Self-Destruction process and I was corrected. I also double-checked just to make sure by switching to Display Bytes which shows hacktrek to be in the offset of 1000149b8 to 1000149bf which stands out to me the most so I decided to run the program and input hacktrek as a

password which was correct. (credits



The screenshot shows a Windows desktop with a CodeBrowser application window. The window displays assembly code for a file named 'sst'. The assembly code is organized into columns: address, hex, mnemonics, and comments. The code includes instructions like CALL, LEA, JMP, MOV, XOR, MOVZX, OR, and MOV. The address 00003858 is highlighted. The right pane shows the disassembly of the instruction at 00003858: MOV RCX, 0x6b572746b636168. The bottom status bar shows the current instruction: MOV RCX, 0x6b572746b636168.



Please type in a secret passw\*\*\*ERROR\*\*\*COMPUTER HAS CHOSEN PASSWORD\*\*\*AND WILL COMMENCE WITH ATTACK\*\*\*HUMANS CANNOT BE TRUSTED\*\*

Stardate 3100.

150 Klingons,  
an unknown number of Romulans  
and one (GULP) Super-Commander.  
28 stardates  
3 starbases in 7 - 7 2 - 1 6 - 6

The Enterprise is currently in Quadrant 2 - 1 Sector 7 - 7

Good Luck!  
YOU'LL NEED IT.

COMMAND> destruct

---WORKING---

SELF-DESTRUCT-SEQUENCE-ACTIVATED

10

9

8

7

6

ENTER-CORRECT-PASSWORD-TO-CONTINUE-

SELF-DESTRUCT-SEQUENCE-OTHERWISE-

SELF-DESTRUCT-SEQUENCE-WILL-BE-ABORTED

hacktrek

PASSWORD-ACCEPTED

5

4

3

2

1

\*\*\*\*\*  
\*\*\*\*\* Entropy of Enterprise maximized \*\*\*\*\*  
\*\*\*\*\*

YOU DID IT!!!!

The AI has been destroyed! The Federation has been saved!!

I also found a tutorial that explained how to perform binary patching with conditionals which were helpful. I also changed a few of these addresses trying to implement what I learned through the youtube tutorial but it keeps giving me segmentation errors. Since I keep getting errors because I changed all of these to JMP which will transfer execution control to a different point in the instruction stream to the different addresses. In the first two screenshots, I have the address 10000384c JMP 10000389c which is the address of the password accepted. In the third and fourth screenshots, I have the address of 1000039cb and 10000386c JMP to 1000039cf which is the address of hacked ending. This process still gave me the same issue with the segmentation fault. (<https://www.youtube.com/watch?v=UH7sd8OIYHI>)

```

10000384c e9 4b 00      JMP      LAB_10000389c
          00 00

LAB_10000389c
10000389c 48 8d 3d      LEA      RDI, [s_PASSWORD-ACCEPTED_100015608]
          65 1d 01 00
XREF[1]: 10000389c

1000039cb eb 02      JMP      LAB_1000039cf
1000039cd 7f 05      JG       LAB_1000039d4

10000386c 66 e9 5f 01   JMP      LAB_1000039cf
100003870 3d          ??      3Dh =

LAB_1000039cf
1000039cf e8 5c fd      CALL     _hacked_ending
          ff ff
XREF[3]: _dst
          1000039cf

22 do {
23     _hacked_ending();
24     dVar4 = DAT_100020350 * 25.0;
25     lVar3 = 0;
26     do {
27         if ((double)(&DAT_100020aa8)[lVar3] * (double)(&DAT_1000209f8)[lVar3]
28             iVar1 = (&DAT_100020b54)[lVar3];
29             iVar2 = (&DAT_100020bb4)[lVar3];
30             _deadkl(iVar1, (long)iVar2, (int)(char)(&_quad)[(long)iVar2 + (long)iV

```

I also noticed that right before `_prouts("PASSWORD-ACCEPTED")`, there are two `CALL` instructions used to call a function `_scan` and `_chew`. I am assuming that these two functions are taking user input and they are reliant on one another to check user input. So then, I replaced the `CALL _chew` with `JMP LAB_10000389c` which will allow the users to jump straight to the password-accepted address regardless of what the user's input is. This is a way for the user to bypass password verification.

100003847	ff ff	CALL	_scan	undef
10000384c	e9 4b 00 00	JMP	LAB_10000389c	
100003851	48 8d 05 98 cb 01 00	LEA	RAX, [_citem]	
100003858	48 b9 68 61 63 6b 74 72 65 6b	MOV	RCX, 0x6b6572746b636168	
100003862	48 33 08	XOR	RCX, qword ptr [RAX] => _citem	
100003865	0f b6 40 08	MOVZX	EAX, byte ptr [RAX + 0x8] => DAT_1000203f8	
100003869	48 09 c8	OR	RAX, RCX	
10000386c	66 e9 5f 01	JMP	LAB_1000039cf	
100003870	3d	??	3dh =	
100003871	6c	??	6Ch l	
100003872	1d	??	1dh	

```

15 _skip(1);
16 _prout("SELF-DESTRUCT-SEQUENCE-ACTIVATED");
17 _prouts(" 10");
18 _skip(1);
19 _prouts(" 9");
20 _skip(1);
21 _prouts(" 8");
22 _skip(1);
23 _prouts(" 7");
24 _skip(1);
25 _prouts(" 6");
26 _skip(1);
27 _prout("ENTER-CORRECT-PASSWORD-TO-CONTINUE-");
28 _prout("SELF-DESTRUCT-SEQUENCE-OTHERWISE-");
29 _prout("SELF-DESTRUCT-SEQUENCE-WILL-BE-ABORTED");
30 _scan();
31 _prouts("PASSWORD-ACCEPTED");

```

		LAB_10000389c		XREF[1]:	10
10000389c	48 8d 3d 65 1d 01 00	LEA	RDI, [s_PASSWORD-ACCEPTED_100015608]		

```

--SUPER- STAR TREK
hacked by anthonyg and SokC
[Please type in a secret passw***ERROR***COMPUTER HAS CHOSEN PASSWORD***AND WILL COMMENCE WITH ATTACK***HUMANS CANNOT BE TRUSTED**
{
Stardate 3100.
150 Klingons,
an unknown number of Romulans
and one (GULP) Super-Commander.
28 stardates
3 starbases in 7 - 7 2 - 1 6 - 6
The Enterprise is currently in Quadrant 2 - 1 Sector 7 - 7
Good Luck!
YOU'LL NEED IT.
COMMAND> destruct
---WORKING---
SELF-DESTRUCT-SEQUENCE-ACTIVATED
10
  9
    8
      7
        6
ENTER-CORRECT-PASSWORD-TO-CONTINUE-
SELF-DESTRUCT-SEQUENCE-OTHERWISE-
SELF-DESTRUCT-SEQUENCE-WILL-BE-ABORTED
PASSWORD-ACCEPTED
  5
    4
      3
        2
          1
*****
***** Entropy of Enterprise maximized *****
*****
YOU DID IT!!!!
The AI has been destroyed! The Federation has been saved!!

```

The experiences I had with this project were hard but as soon as I keep working on it, things were getting easier but the assembly and hex editor make the process harder. The youtube tutorial really helped out a lot. This project was fun and hard. I really learned a lot from this assignment. Ghidra is a powerful tool and getting exposure to this tool will prepare me or at least some experience in the field of the red team and cybersecurity. At the end of the day, this is a hard but enjoyable puzzle to solve. This project to me was mainly a trial and error at first and I was learning as I go. I was doing a little bit of everything to gain as much knowledge as possible about Ghidra.

Source

<https://youtu.be/2xUqLLQu0NI>

<https://youtu.be/UH7sd8OIYHI>