# Overview

This is a “Taylor Swift Ticketmaster Fiasco” Themed challenge. The players are going to try to patch a unused function in a obfuscated HTTP client to retrieve a flag.

There are actually TWO flags. The first flag is only worth 100 points, it’s to teach the players about maintenance hooks, basically software and hardware backdoors left behind by the developer who may have forgotten to disable it. For example, default passwords for Zyxel Switches, etc. The first flag is **TPZ{1_L0v3_H00K3R_h3Ad3R2}**

The second flag is worth 10,000 points. Using the clues from the discovery of the maintenance hook, the player will find out there is one more “ticket” to retrieve the 10,000 point flag.[1] The second flag is **TPZ{but_5H3_w34r5_5H0Rt_5KiRT5_I_W34R_t-5hirT5}**

The objective of this exercise, whether or not they succeed, is to show commonly encountered obfuscation techniques used in malware. Other techniques, such as Virtual Machine Obfuscation have not been implemented. But common methods used in APT-level malware have been used

1. Anti-debugging
2. Opaque Predicates
3. Control-Flow Flattening
4. Mixed Boolean Arithmetics
5. Obfuscated strings

[1] It’s important to show the public about maintenance hooks and backdoors. They are commonly found among many popular vendors, which leads to vulnerability findings and likely exploitation.

# Storyline

As a highly successful web developer, Jacob's arrogance knew no bounds. He was convinced that he was the best in the business and had nothing to worry about. His web application “TicketMeister” was the talk of the town, praised for its seamless performance, ease of use, and sleek design. He believed that he had done everything to make sure his application was secure. But he was wrong. Unbeknownst to him, Jacob had accidentally left a back door in his application, a vulnerability that he was completely unaware of. Little did he know that this back door would be the reason for his downfall.

---

Laylor Grift, the best-selling Female Music Artist in the world right now, had just broken the world record for sell out show sales. Her WORLD TOUR had completely sold out in 1 Hour. Every. Single. Show.

Over 1 million Tickets.

So, Laylor has spontaneously decided to do one extra show.

This is taken from her Twitter.

"I'm so overwhelmed with how rich I've just become from all those ticket sales. It's a testament to how talented I am. So as a thank you to you, my fans, I've just decided to do one more show. Tickets will be available from Ticketmeister in exactly 24hrs."

Well, that has come as a shock to everyone …including ticketmeister….as their website isn't showing anything about this new show.

When asked for a comment on how Ticketmeister would handle the surge in traffic because of the incredible demand for tickets, Jacob responded,

“It’s a non-issue”.

---

In the shadows, "Steve" (not his real name) had been watching. This Ticketmeister app had a flaw. He had probed and poked at it over several days. It was so obvious, such a rookie error, that Steve had to check and recheck his findings…..but he was sure he was correct. With this information he could (and probably would) PWN Ticketmeister.

Challenge:

Your target is the "Ticketmeister" application, a popular event ticketing platform. The application has a maintenance hook that allows the developer to perform maintenance on the system without taking it down. However, the developer has left this hook unpatched, leaving a potential vulnerability for attackers to exploit. Your task is to find and exploit this vulnerability to gain access to the system.

Additional Information:

•The unpatched maintenance hook is hidden somewhere within the Ticketmeister application.
•You'll need to reverse engineer the code to find the vulnerability.
•Once you gain access to the system, you'll need to find the flag and capture it before the other teams do.

Can you find it before anyone else does?

# Technical Description

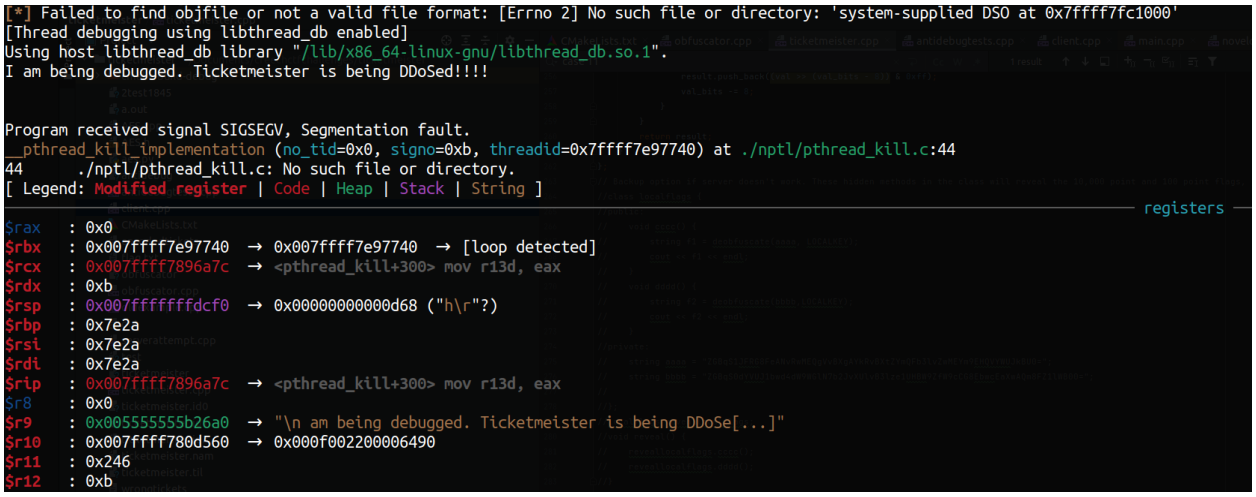
There are three classes, ***antidebug*** for anti-debugging implementation, ***Request*** for the HTTP client, and ***obfuscator***, which contains all of the necessary functions to decode and deobfuscate the “tickets”. These tickets are then sent to a HTTP server as a GET request, the correct “ticket” or “ticket order” retrieves the flag.

The winning “ticket order” is this GET request which returns this flag
TPZ{but_5H3_w34r5_5H0Rt_5KiRT5_I_W34R_t-5hirT5}

GET /be931e23cticketd7ac2f62 HTTP/1.1
Host: angruhtaylorswiftfans.pwn
Accept: text/plain
Accept-Language: en-us
Accept-Encoding: identity
Connection: close
User-Agent: F.U. TicketMeister

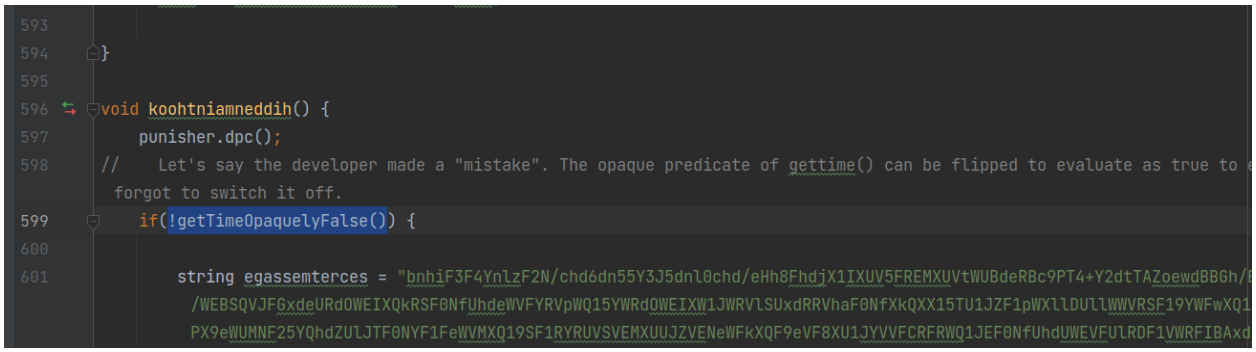
If there are technical issues with networking in the CTF, you can comment out the fourth class, ***localflags***, which has the instances and class methods needed to reveal both the 100 point and 10,000 point flags.

The ***punisher*** instance, is a instance of the ***antidebug*** class, and utilizes three different methods to detect debuggers on Linux, and will segfault and crash the program upon detection.




The reason why I called the class instance “punisher” is because you can do a lot of bad things to the analyst’s machine if they didn’t patch out the anti-debug detections. For example, if the trap was set, and I enumerated you as a root user, I can send a obfuscated command to delete your entire root directory as “punishment” for attempting to debug the application. Obviously I didn’t add that code, but this is a reference from the book Surreptitious Software.

The entire point of the exercise is how a single character that left a maintenance hook (“software backdoor”), open, can lead to full compromise of the application and everything that depends on it. All that was needed was either commenting out the maintenance hook, or removing the “!” character in the opaque predicate.




These vulnerabilities are common, and I can spend hours sending you links of such flaws from news feeds around the world.

If you want to drop more anti-debugging techniques, you have three class methods to choose from

* punisher.ptc() is the ptrace check
* punisher.dpc() checks for common debuggers
* punisher.sdc() attempts to attach itself as a debugger (if it cannot attach to its own process, another debugger is using it)

```
// usage: punisher.method();
class antidebug {
public:
//    ptrace check, segfaults if it finds ptrace
    void ptc() {
        if(ptrace(PTRACE_TRACEME)) {
            printf("Debugger detected! Can't steal my tickets suckah!\n");
            raise(SIGSEGV);
        } else {
            return;
        }
    }
//    Check for common debuggers, segfault if it finds any
    void dpc() {
        if (isUnderDebugger())
        {std::cout << "I am being debugged. Ticketmeister is being DDoSed!!!!\n" << std::endl;
            raise(SIGSEGV);}
        else
        return;
    }
//    Attach self in debugging mode, if cannot attach, segfault. Pretty unreliable, just like timing attacks.
    void sdc() {
        if (ptrace(PTRACE_TRACEME, 0, 1, 0) == -1) {
            std::cout << "YOU SHOULD HAVE PATCHED ME OUT, SCALPER!!!\n" << std::endl;
            raise(SIGSEGV);
        } else {
            return;}



    }
private:
    bool hasEnding (std::string const &fullString, std::string const &ending) {
        if (fullString.length() >= ending.length()) {
            return (0 == fullString.compare (fullString.length() - ending.length(), ending.length(), ending));
        } else {
            return false;
        }
    }
    bool isUnderDebugger()
    {
        bool result = false;
        /*
```

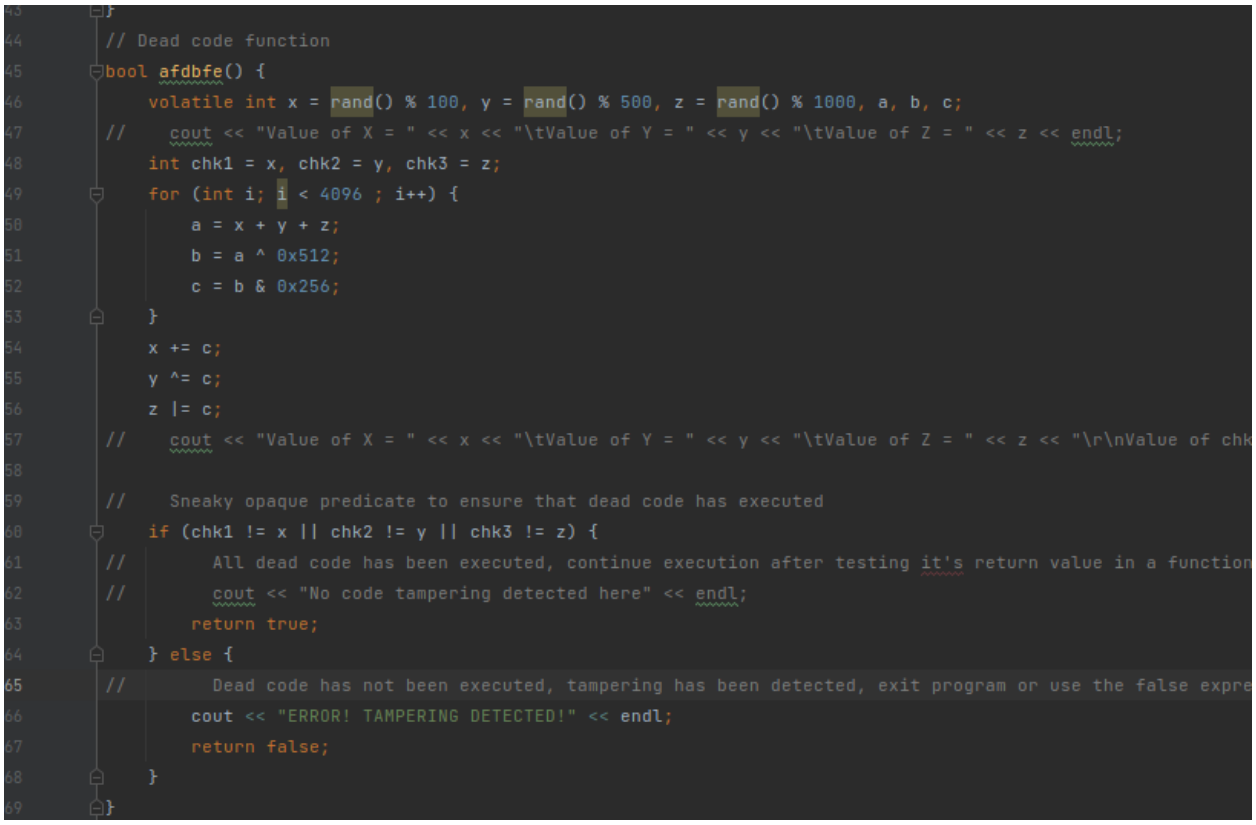
The initCFF() function starts up the Control Flow Flattening Sequence, where arguments are filled into bbebfffcba() function.

```
// Control flow flattening block
int bbebfffcba(int y, int x[], int w, int n) {
    int R, L, k, s;
    int next = 0;
    for(;;)
        switch(next) {
        case 0 : afdbfe(); k = 0; s=1;next=1;break;
        case 1 : afdbfe(); if (k<w) next = 2; else next = 6; break;
        case 2 : afdbfe(); if (x[k]==1) next = 3; else next = 4; break;
        case 3 : afdbfe(); R = (s*y)%n; next = 5; break;
        case 4 : afdbfe(); R=s; next=5;break;
        case 5 : afdbfe(); s=R*R%n; L=R;k++;next=1;break;
        case 6 : afdbfe(); return L;
    }
}
// After we are done here deobfuscating the tickets, we need to make th
int initCFF(){
//
//
    int y = rand()%10;
    int x[] = {5, 10, 20, 40};
    int w = rand()%100;
    int n = rand()%5;
    bbebfffcba(y, x, w, n);
    return 1;
}
```

The variable “next” is the Control Flow dispatcher, and each prepends another function that executes dead-code,  afdbfe() before moving to the next function. It is inserted into specific functions so even if anti-debugging checks are patched, they have to get past the dead code. It will execute useless computations 4096 times. Reverse-engineers can enter the initCFF() function and patch a RET to stop the Control Flow Flattening Sequence from starting up. In this case, it will be line #6, for the first call the the rand() function. Patch it out with a RET and you’ll kill the functionality of Control Flow Flattening. However, notice that on the bottom that it returns a integer of 1, or 0x1, you should always consider patching a RET with the same value, that is RET 0x1 in GHIDRA, because that value may be tested to detect tampering.

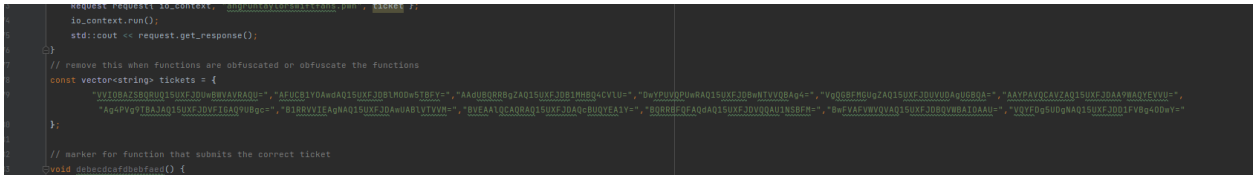
There is a unused opaque predicate (I only used it to ensure that the “maintenance hook” is going to execute) that you may use to predefine logic and control flow if you want to modify the challenge, and it can be used to detect tampering (if the program was patched before the calculations are done, then the tested variables are the same, you can choose to exit the program if you’d like). Or you can do a switcheroo and test it for opaquely true to force players to patch it to reveal the hook. But what I am trying to do is explain common software flaws to the players.

Also, in GHIDRA, while it normally strips away dead code, you can make it reappear again on the decompilation section by evaluating the values of the variables holding them, and returning a value, like true or false, after evaluating the contents.



```
}
// Dead code function
bool afdbfe() {
    volatile int x = rand() % 100, y = rand() % 500, z = rand() % 1000, a, b, c;
//    cout << "Value of X = " << x << "\tValue of Y = " << y << "\tValue of Z = " << z << endl;
    int chk1 = x, chk2 = y, chk3 = z;
    for (int i; i < 4096 ; i++) {
        a = x + y + z;
        b = a ^ 0x512;
        c = b & 0x256;
    }
    x += c;
    y ^= c;
    z |= c;
//    cout << "Value of X = " << x << "\tValue of Y = " << y << "\tValue of Z = " << z << "\r\nValue of chk

//    Sneaky opaque predicate to ensure that dead code has executed
    if (chk1 != x || chk2 != y || chk3 != z) {
//        All dead code has been executed, continue execution after testing it's return value in a function
//        cout << "No code tampering detected here" << endl;
        return true;
    } else {
//        Dead code has not been executed, tampering has been detected, exit program or use the false expre
        cout << "ERROR! TAMPERING DETECTED!" << endl;
        return false;
    }
}
```

The most important part of the challenge is the array (actually a vector<string> type), called “tickets”. These are the obfuscated tickets that will be sent through the HTTP client code.



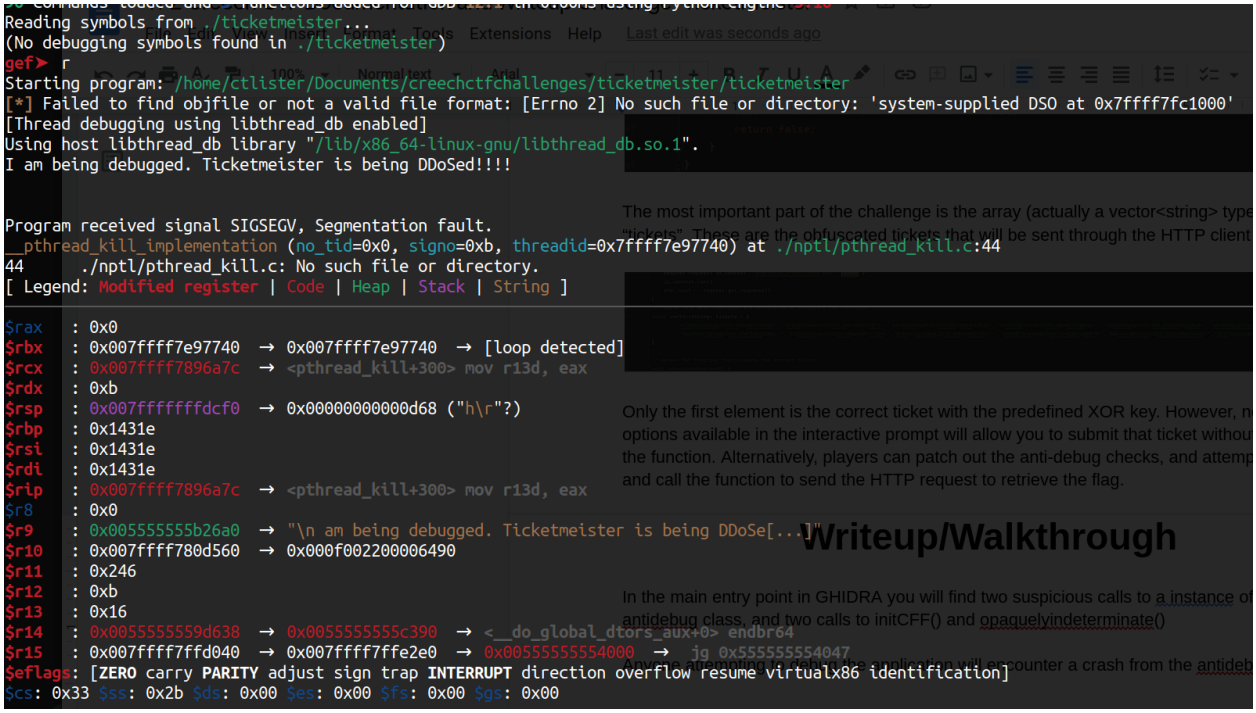
```
    io_context.run();
    std::cout << request.get_response();
}
// remove this when functions are obfuscated or obfuscate the functions
const vector<string> tickets = {
        "VVI08AZ5BQRUQ15UXFJDUwBWVAVPAQU=","AFUCB1Y0AwdAQ15UXFJDBlMODw5TBFY=","AAdUBQRRBgZAQ15UXFJDB1MHBQ4CVlU=","DwYPUVQPUwRAQ15UXFJDBwNTVVQBAg4=","VgQGBFMGUgZAQ15UXFJDUVUDAgUGBQA=","AAYPAVQCAVZAQ15UXFJDAA9WAQYEVVU=",
        "Ag4PVgYIBAJAQ15UXFJDVFIGAQ9UBgc=","B1RRVVIEAgNAQ15UXFJDAwUABlYTVVN=","BVEAAlQCAQRAQ15UXFJDAQcBUQYEA1Y=","BQRRBFQFAQdAQ15UXFJDVQQAU1NSBFM=","BwFVAFVWVQVAQ15UXFJDBQVWBAIOAAU=","VQYFDg5UDgNAQ15UXFJDD1FVBg4ODwY="
};

// marker for function that submits the correct ticket
void debecdcafdbebfaed() {
```

Only the first element is the correct ticket with the predefined XOR key. However, none of the options available in the interactive prompt will allow you to submit that ticket without patching the function. Alternatively, players can patch out the anti-debug checks, and attempt to identify and call the function to send the HTTP request to retrieve the flag.

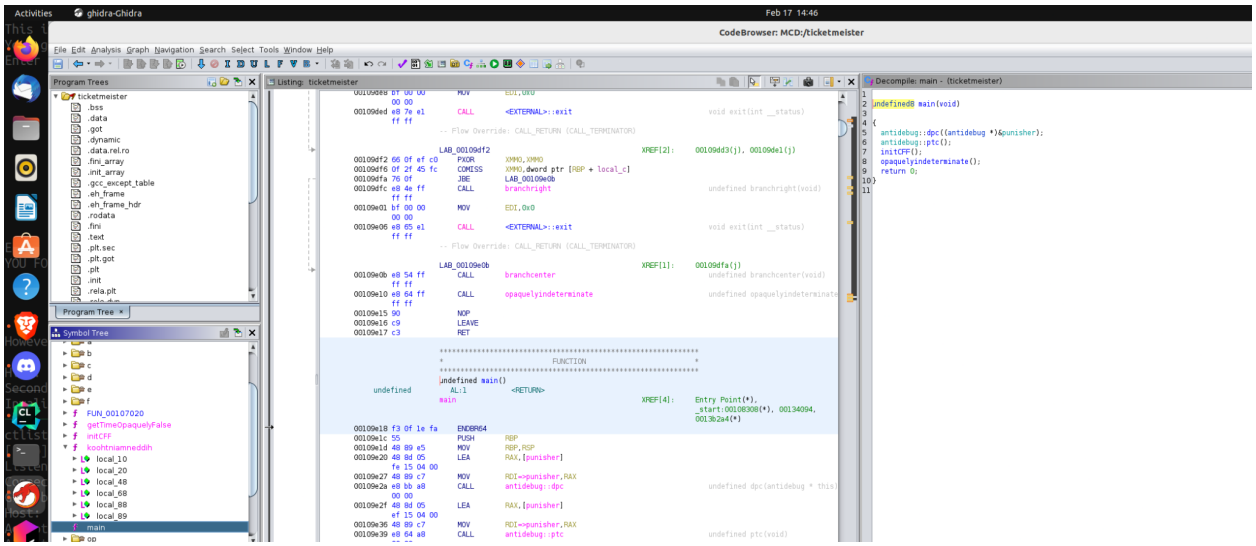
# Writeup/Walkthrough

In the main entry point in GHIDRA you will find two suspicious calls to a instance of the antidebug class, and two calls to initCFF() and opaquelyindeterminate()

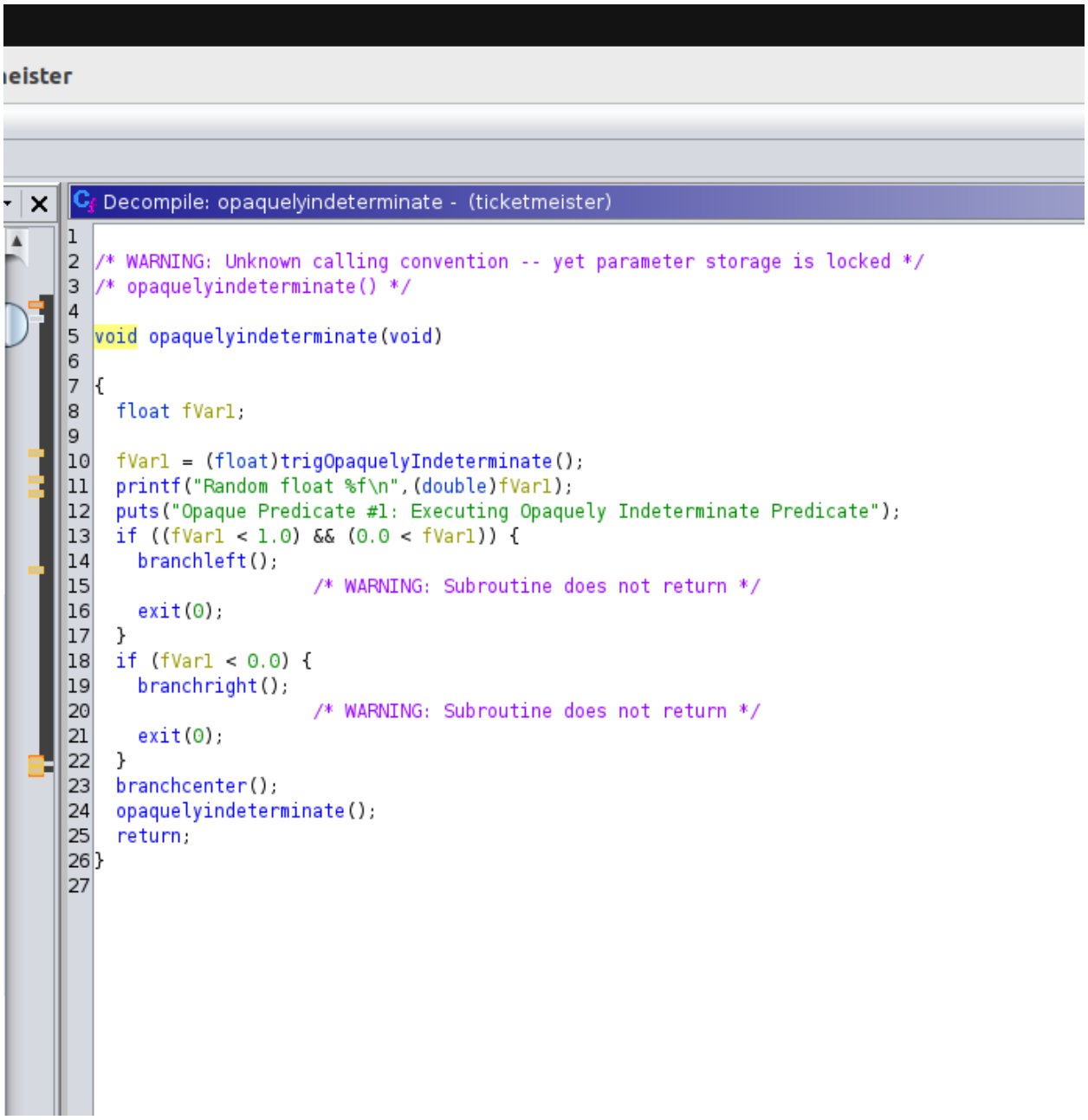
Anyone attempting to debug the application will encounter a crash from the antidebug class.*There is a alternative method of identifying the anti-debugging methods, patching them out to prevent it from functioning.*



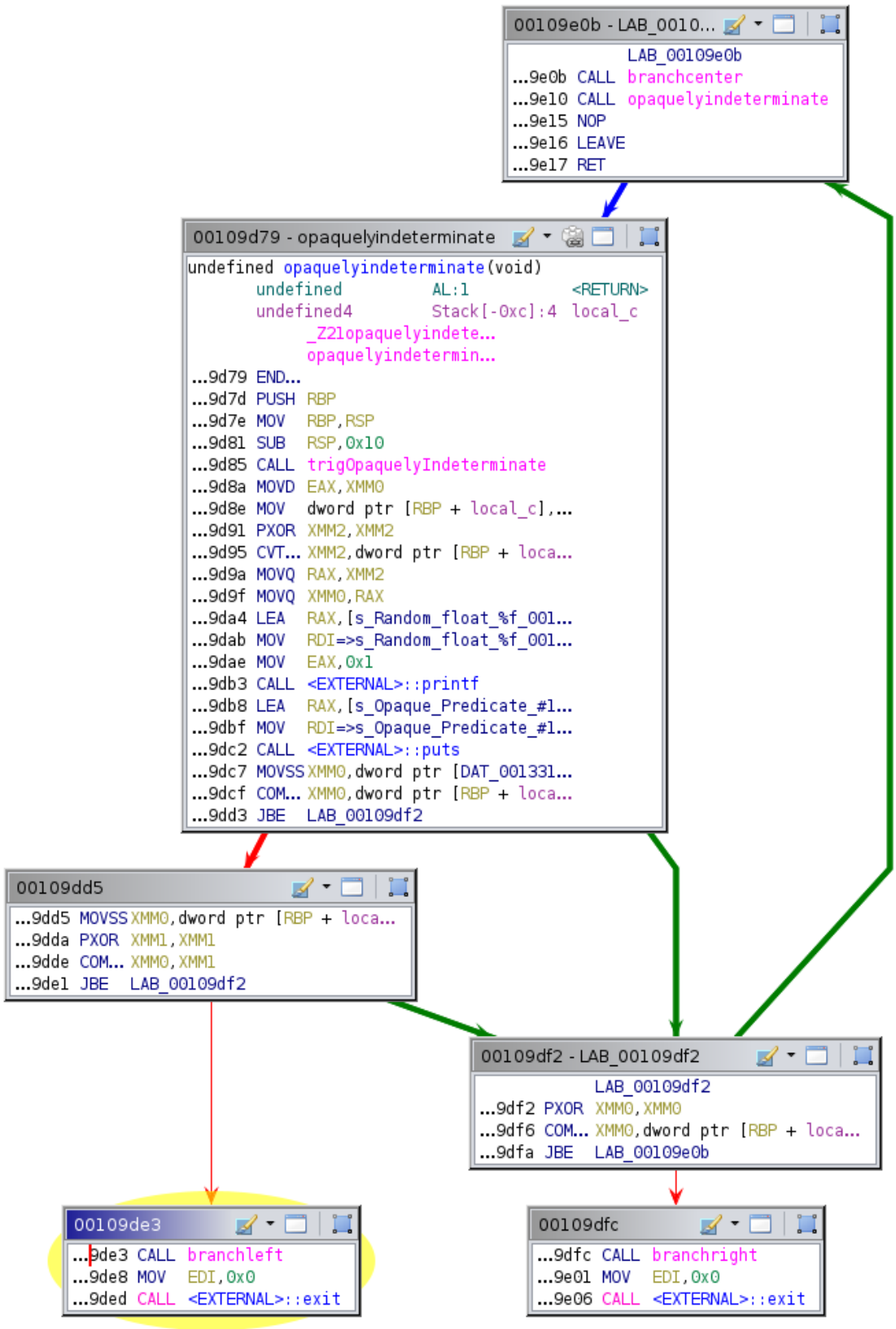
Instead, players are encouraged to look for the hidden maintenance hook, a intentional backdoor for developers that was improperly implemented and not disabled. Instead of the opaque predicate protecting the first flag from functioning, it actually reveals both the 100 point flag and the hint to the 10,000 point flag.



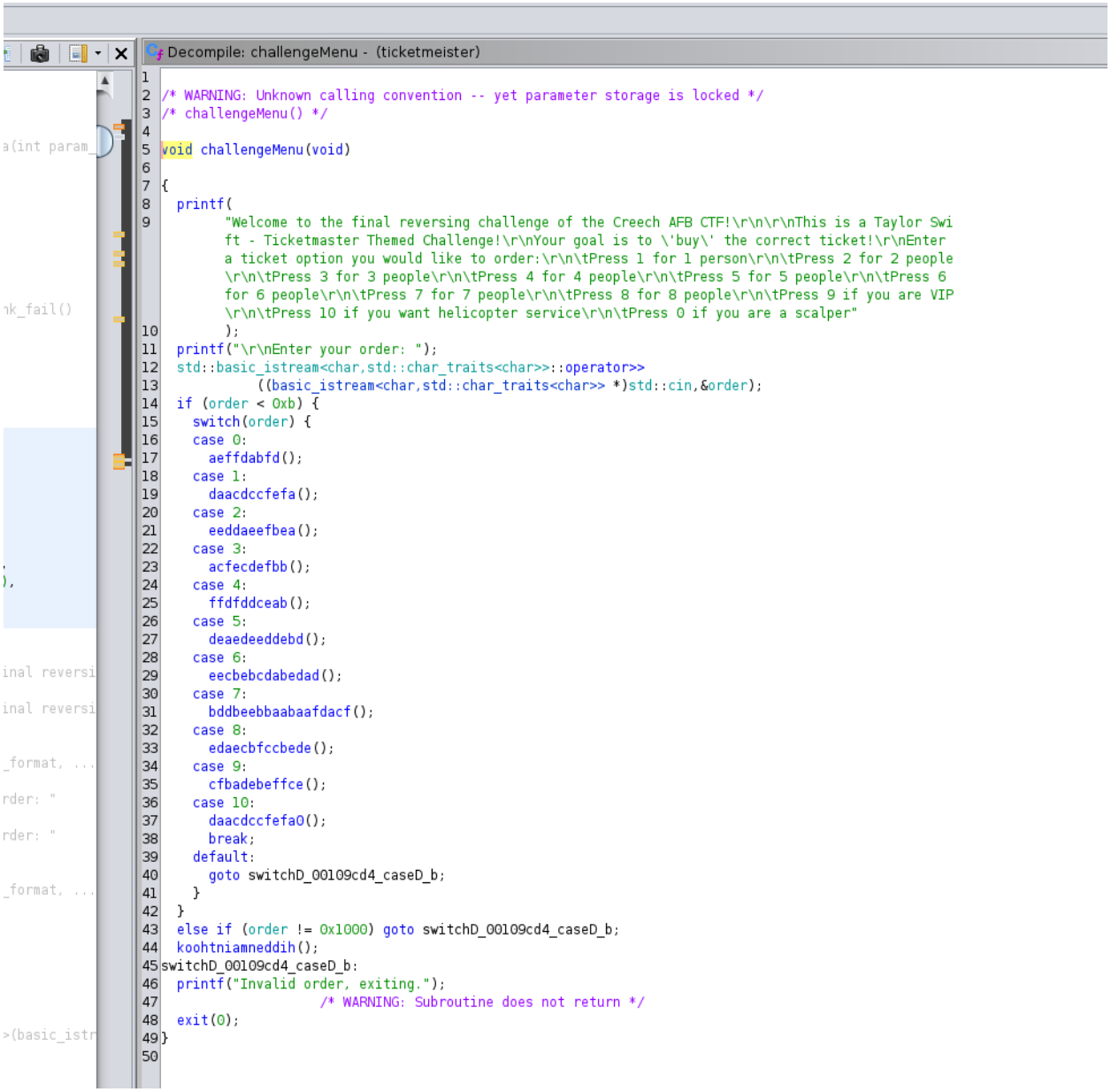
Following the opaquelyindeterminate() call gives you three control flow patterns as shown on this graph



Graph view


00109e0b - LAB_0010...
LAB_00109e0b
...9e0b CALL branchcenter
...9e10 CALL opaquelyindeterminate
...9e15 NOP
...9e16 LEAVE
...9e17 RET
00109d79 - opaquelyindeterminate
undefined opaquelyindeterminate(void)
undefined AL:1 <RETURN>
undefined4 Stack[-0xc]:4 local_c
_Z21opaquelyindete...
opaquelyindetermin...
...9d79 END...
...9d7d PUSH RBP
...9d7e MOV RBP,RSP
...9d81 SUB RSP,0x10
...9d85 CALL trigOpaquelyIndeterminate
...9d8a MOVD EAX,XMM0
...9d8e MOV dword ptr [RBP + local_c],...
...9d91 PXOR XMM2,XMM2
...9d95 CVT... XMM2,dword ptr [RBP + loca...
...9d9a MOVQ RAX,XMM2
...9d9f MOVQ XMM0,RAX
...9da4 LEA RAX,[s_Random_float_%f_001...
...9dab MOV RDI=>s_Random_float_%f_001...
...9dae MOV EAX,0x1
...9db3 CALL <EXTERNAL>::printf
...9db8 LEA RAX,[s_Opaque_Predicate_#1...
...9dbf MOV RDI=>s_Opaque_Predicate_#1...
...9dc2 CALL <EXTERNAL>::puts
...9dc7 MOVSS XMM0,dword ptr [DAT_001331...
...9dcf COM... XMM0,dword ptr [RBP + loca...
...9dd3 JBE LAB_00109df2
00109dd5
...9dd5 MOVSS XMM0,dword ptr [RBP + loca...
...9dda PXOR XMM1,XMM1
...9dde COM... XMM0,XMM1
...9de1 JBE LAB_00109df2
00109df2 - LAB_00109df2
LAB_00109df2
...9df2 PXOR XMM0,XMM0
...9df6 COM... XMM0,dword ptr [RBP + loca...
...9dfa JBE LAB_00109e0b
00109de3
...9de3 CALL branchleft
...9de8 MOV EDI,0x0
...9ded CALL <EXTERNAL>::exit
00109dfc
...9dfc CALL branchright
...9e01 MOV EDI,0x0
...9e06 CALL <EXTERNAL>::exit

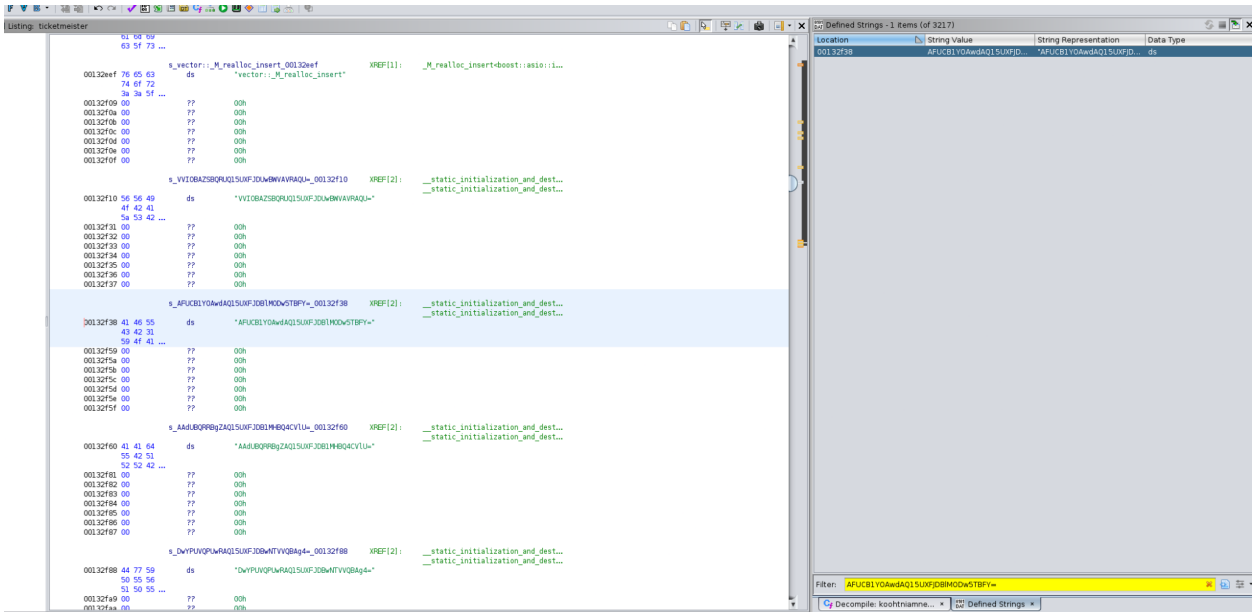
This was all meant to throw the player off their game. Even though control flow splits between branch left, right, and center, they actually lead to the presentation of the challenge menu.




Notice the hidden command for the variable 0x1000, which is 4096 in decimal. If the player enters 4096 as a option and presses enter, they get the first 100 point flag and a clue to where the other flags are stored. The clue is the encoded “ticket”, **AFUCB1YOAwdAQ15UXFJDBIMODw5TBFY=**

```
This is a Taylor Swift - Ticketmaster Themed Challenge!
Your goal is to 'buy' the correct ticket!
Enter a ticket option you would like to order:
       Press 1 for 1 person
       Press 2 for 2 people
       Press 3 for 3 people
       Press 4 for 4 people
       Press 5 for 5 people
       Press 6 for 6 people
       Press 7 for 7 people
       Press 8 for 8 people
       Press 9 if you are VIP
       Press 10 if you want helicopter service
       Press 0 if you are a scalper
Enter your order: 4096
YOU FOUND THE MAINTENANCE HOOK! The first flag is

            TPZ{1_L0v3_H00K3R_h3Ad3R2}


However, if you use the information you learned from this hidden maintenance hook to submit the real request, you get the full 10,000 points.

Hint: You need to find the correct function which deobfuscates the correct base64 encoded and XORed ticket string, and patch that into ANY of the selectable tickets. There are ELEVEN tickets, not TEN!
Second Hint: In GHIDRA the obfuscated strings look like this. AFUCB1YOAwdAQ15UXFJDBlMODw5TBFY=. That is one of the strings
```

Search by defined strings, and you'll find that there is a vector<string> data type holding all of the encoded strings.



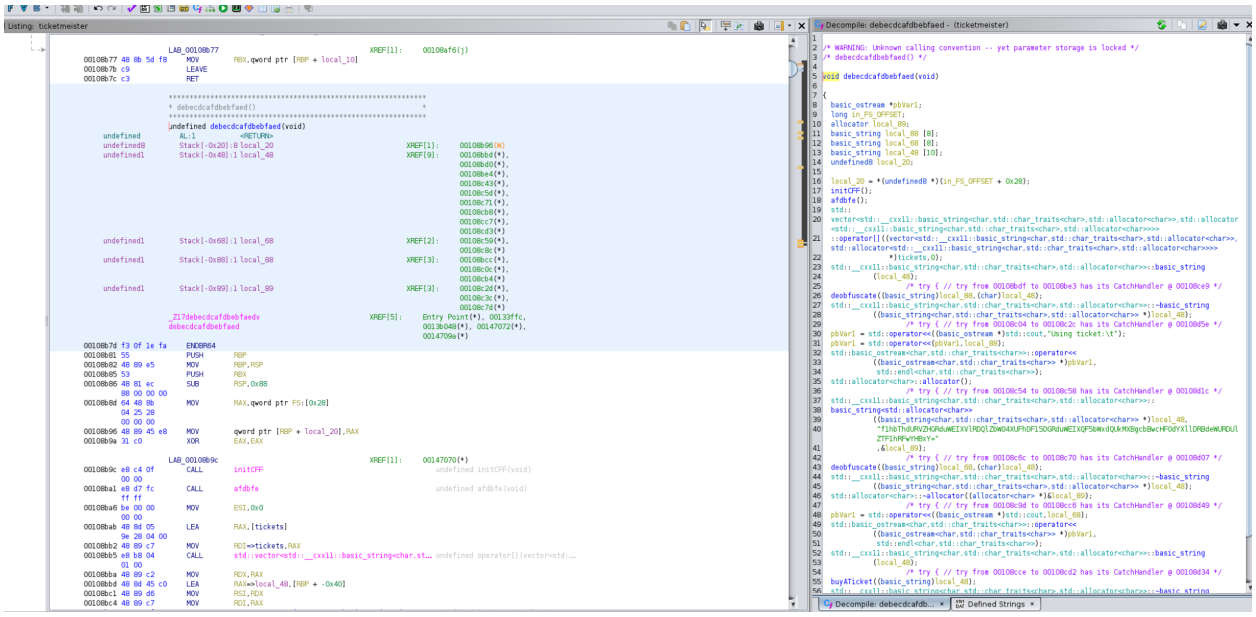

Look one array element above and take note of the encoded string **VVIOBAZSBQRUQ15UXFJDUwBWVAVRAQU=** which is the correct "ticket". There actually is a function that uses it but it's not in the options menu.

If the player continues tracing the call from the options available in the **challengeMenu()** they will find a hidden function at memory address 0x00108b7d named **debecdcafdbebfaed(),** which calls the deobfuscate function

```
                    * deobfuscate(std::__cxx11::basic_string<char, std::char_... *
                    ****************************************************************
                    undefined __stdcall deobfuscate(basic_string param_1, ch...
     undefined          AL:1           <RETURN>
     basic_string       EDI:4          param_1
     char               SIL:1          param_2
     undefined8         Stack[-0x10]:8 local_10                          XREF[1]:     00108877(R)
     undefined8         Stack[-0x20]:8 local_20                          XREF[2]:     00108743(W),
                                                                                      001087f4(R)
     undefined1         Stack[-0x48]:1 local_48                          XREF[6]:     00108750(*),
                                                                                      00108763(*),
                                                                                      00108779(*),
                                                                                      00108790(*),
                                                                                      001087a3(*),
                                                                                      001087b8(*)
     undefined1         Stack[-0x68]:1 local_68                          XREF[3]:     0010879f(*),
                                                                                      001087c4(*),
                                                                                      001087db(*)
     undefined1         Stack[-0x88]:1 local_88                          XREF[3]:     0010875f(*),
                                                                                      0010878c(*),
                                                                                      001087e7(*)
     undefined8         Stack[-0x90]:8 local_90                          XREF[3]:     00108724(W),
                                                                                      001087c8(R),
                                                                                      00108870(R)
     undefined8         Stack[-0x98]:8 local_98                          XREF[2]:     0010872b(W),
                                                                                      00108749(R)
     undefined1         Stack[-0x9c]:1 local_9c                          XREF[2]:     00108734(W),
                                                                                      00108785(R)
                    _Z11deobfuscateNSt7__cxx1112basic_stringIcStll...  XREF[19]:    Entry Point(*),
                    deobfuscate                                                      debecdcafdbebfaed:00108bdf(c),
                                                                                     debecdcafdbebfaed:00108c6c(c),
                                                                                     daacdccfefa:00108dd1(c),
                                                                                     eeddaeefbea:00108efa(c),
                                                                                     acfecdefbb:00109023(c),
                                                                                     ffdfddceab:0010914c(c),
                                                                                     deaedeeddebd:00109275(c),
                                                                                     eecbebcdabedad:0010939e(c),
                                                                                     bddbeebbaabaafdacf:001094c7(c),
                                                                                     edaecbfccbede:001095f0(c),
                                                                                     cfbadebeffce:00109719(c),
                                                                                     daacdccfefa0:00109842(c),
                                                                                     aeffdabfd:0010996b(c),
                                                                                     koohtniamneddih:00109eee(c),
                                                                                     00133fe4, 0013afb8(*),
                                                                                     0014702b(*), 00147047(*)
00108714 f3 0f 1e fa     ENDBR64
00108718 55              PUSH        RBP
```

The first deobfuscate() call 0x00108bdf loads the obfuscated ticket and on 0x00108cce, passes the information to the buyATicket() function, which sends the HTTP GET request to the server.

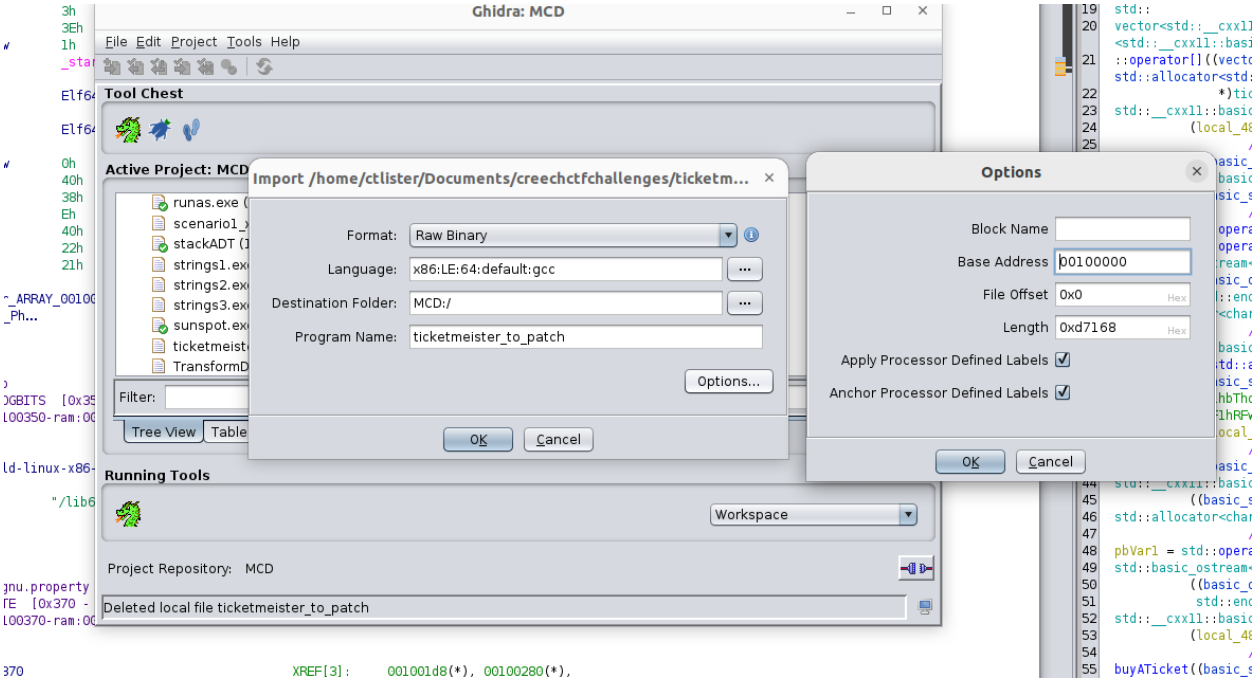

If the player continues tracing the call they will find a hidden function at memory address 0x00108b7d named **debecdcafdbebfaed()** by moving up in the disassembly. It calls vector<string> element 0, the correct “ticket” to submit.

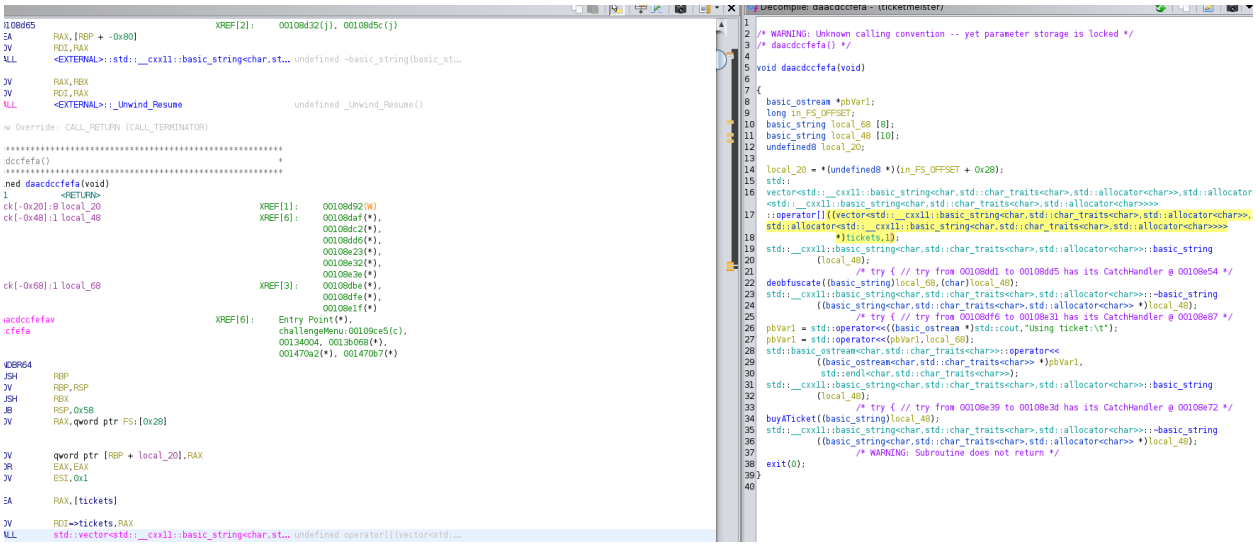


```
                        * debecdcafdbebfaed()                                 *
                        *************************************************************
                        undefined debecdcafdbebfaed(void)
undefined                 AL:1           <RETURN>
undefined8                Stack[-0x20]:8 local_20                          XREF[1]:     00108b96(W)
undefined1                Stack[-0x48]:1 local_48                          XREF[9]:     00108bbd(*),
                                                                                        00108bd0(*),
                                                                                        00108be4(*),
                                                                                        00108c43(*),
                                                                                        00108c5d(*),
                                                                                        00108c71(*),
                                                                                        00108cb8(*),
                                                                                        00108cc7(*),
                                                                                        00108cd3(*)
undefined1                Stack[-0x68]:1 local_68                          XREF[2]:     00108c59(*),
                                                                                        00108c8c(*)
undefined1                Stack[-0x88]:1 local_88                          XREF[3]:     00108bcc(*),
                                                                                        00108c0c(*),
                                                                                        00108cb4(*)
undefined1                Stack[-0x89]:1 local_89                          XREF[3]:     00108c2d(*),
                                                                                        00108c3c(*),
                                                                                        00108c7d(*)
                        _Z17debecdcafdbebfaedv                             XREF[5]:     Entry Point(*), 00133ffc,
                        debecdcafdbebfaed                                               0013b048(*), 00147072(*),
                                                                                        0014709a(*)
00108b7d f3 0f 1e fa      ENDBR64
```

The player is to patch a CALL from any of the usable functions to point to array element 0 instead. To do this they must reopen the binary as raw as GHIDRA's patching functions is naturally bugged (due to a 3-year old linker issue).

Set 00100000 as the base address option to align it because in raw format it's difficult to read, but we can pick any of the options from the menu to patch the binary.



If you look at the function that corresponds to input number 1, notice the highlighted section where tickets,1 points to the array element…[2]

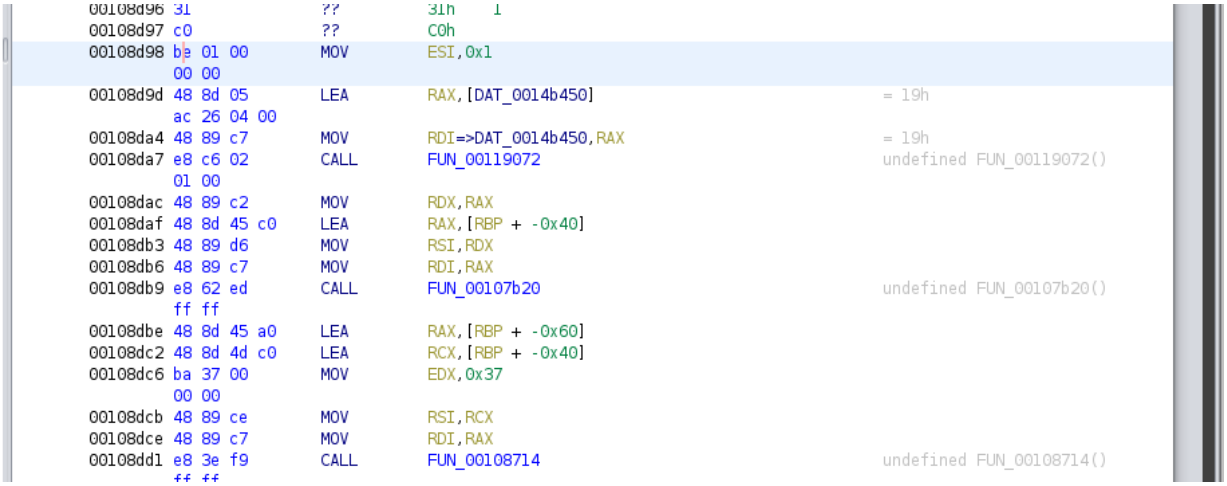
[2] If you observe on the right, the number 1 is the last item on the statement. Due to assembly's FIFO nature, this means that the number 1 or hex 0x1 will be at the TOP of the section of the stack, and not the bottom. This is how quickly I learned that the ESI register holds the value of 0x1, or the index of the vector of strings.

```
5  void daacdccfefa(void)
6
7  {
8    basic_ostream *pbVar1;
9    long in_FS_OFFSET;
10   basic_string local_68 [8];
11   basic_string local_48 [10];
12   undefined8 local_20;
13
14   local_20 = *(undefined8 *)(in_FS_OFFSET + 0x28);
15   std::
16   vector<std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>,std::allocator
     <std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>>>
17   ::operator[]((vector<std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>,
     std::allocator<std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>>>
18                 *)tickets,1);
19   std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string
20             (local_48);
21                     /* try { // try from 00108dd1 to 00108dd5 has its CatchHandler @ 00108e54 */
22   deobfuscate((basic_string)local_68,(char)local_48);
23   std::  cxx11::basic string<char,std::char traits<char>,std::allocator<char>>::~basic string
```

Now look at the disassembly section, the ESI register is holding the value of the array. We need to patch out 0x1 with 0x0 to point to the “correct ticket”.

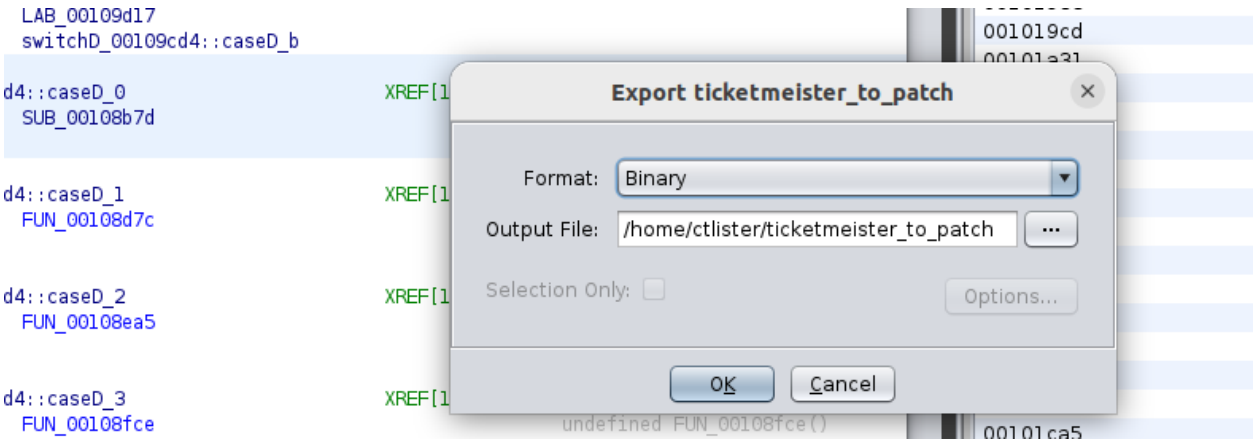
```
                                                                    00134004, 0013b068(*),
                                                                    001470a2(*), 001470b7(*)
00108d7c f3 0f 1e fa     ENDBR64
00108d80 55              PUSH        RBP
00108d81 48 89 e5        MOV         RBP,RSP
00108d84 53              PUSH        RBX
00108d85 48 83 ec 58     SUB         RSP,0x58
00108d89 64 48 8b        MOV         RAX,qword ptr FS:[0x28]
         04 25 28
         00 00 00
00108d92 48 89 45 e8     MOV         qword ptr [RBP + local_20],RAX
00108d96 31 c0           XOR         EAX,EAX
00108d98 be 01 00        MOV         ESI,0x1
         00 00
00108d9d 48 8d 05        LEA         RAX,[tickets]
         ac 26 04 00
00108da4 48 89 c7        MOV         RDI=>tickets,RAX
00108da7 e8 c6 02        CALL        std::vector<std::__cxx11::basic_string<char,st... undefined operator[](vector<std:...
         01 00
00108dac 48 89 c2        MOV         RDX,RAX
00108daf 48 8d 45 c0     LEA         RAX=>local_48,[RBP + -0x40]
00108db3 48 89 d6        MOV         RSI,RDX
```

On your raw disassembly and go to your aligned memory address 0x00108d98 Press Ctrl + Shift + G to change the value of 0x1 (#1) to 0x0 (#0) to point the function to the winning ticket.
*To make it simpler, press ‘d’ to disassemble to easily see the opcodes to patch*

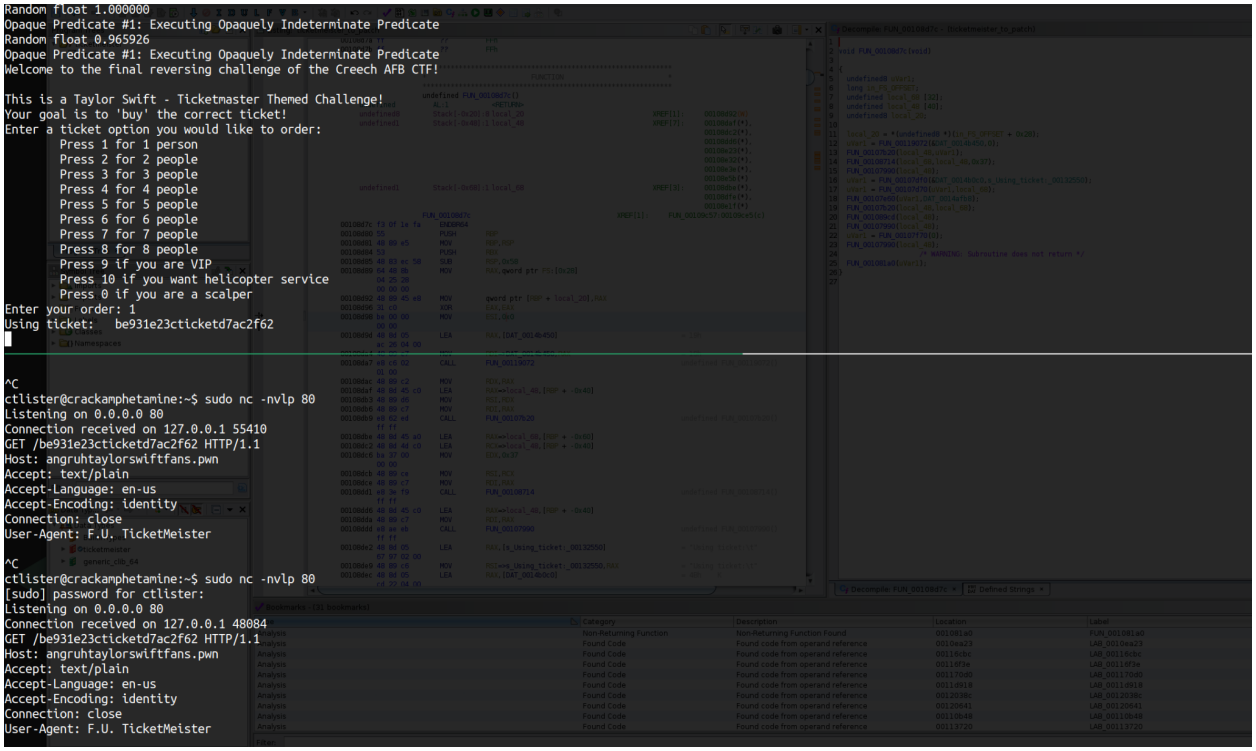



Then patch the MOV ESI,0x1 instruction to MOV ESI 0x0 so the vector<string> points to element 0, the correct ticket to retrieve the flag.

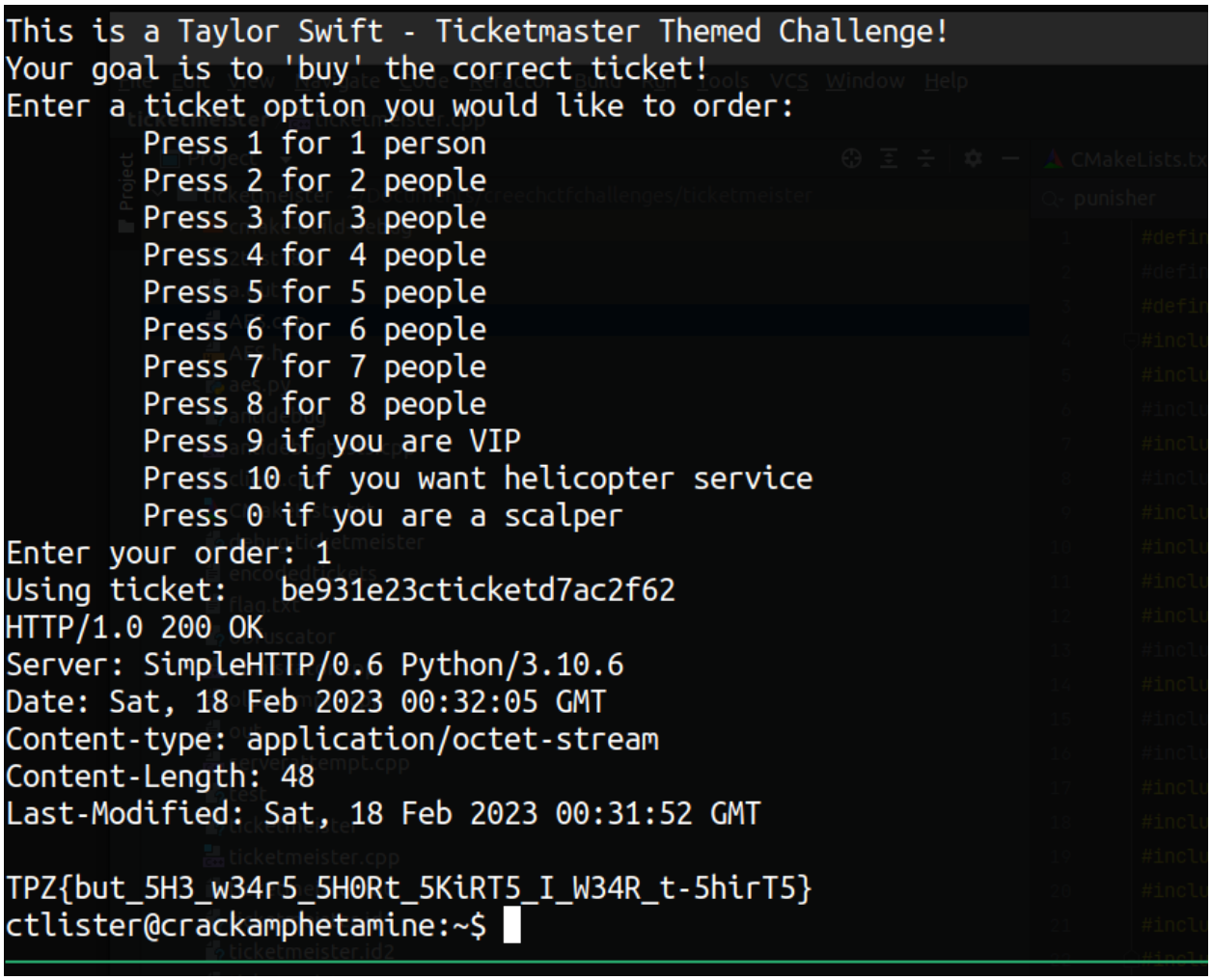
Press O to export the binary and make it executable, chmod +x ticketmeister_to_patch.bin



Run the menu again and press 1 and you will retrieve the winning ticket and the 10,000 point flag.



Run this against the server. You will retrieve the flag.

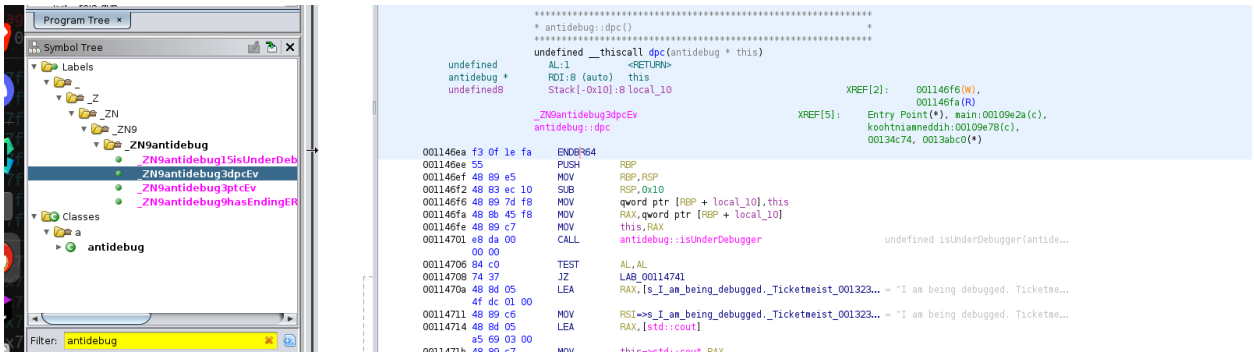


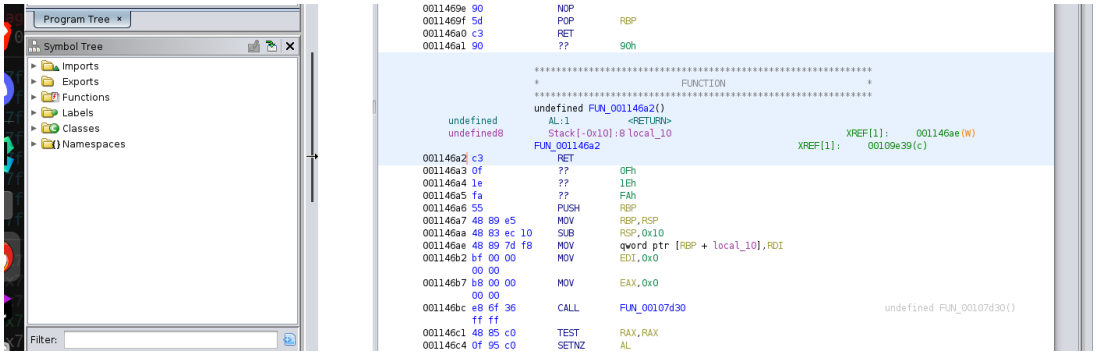
# Optional Pathway: Killing the Anti-Debug Functions

To make the application debuggable there are two checks for debuggers in the main() entry point before it enters the control flow flattening sequence.

You can patch a RET in the raw import of the binary of these functions to prevent the anti-debugging tools from functioning. This includes ANY calls and any instances of the antidebug class. It will also disable any class instances attempting to use the anti-debugging tricks.

001146ea
001146a2



```
[*] Failed to find objfile or not a valid file format: [Errno 2] No such file or directory: 'system-supplied DSO at 0x7ffff7fc1000'
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Random float 1.000000
Opaque Predicate #1: Executing Opaquely Indeterminate Predicate
Random float 0.690252
Opaque Predicate #1: Executing Opaquely Indeterminate Predicate
Welcome to the final reversing challenge of the Creech AFB CTF!

This is a Taylor Swift - Ticketmaster Themed Challenge!
Your goal is to 'buy' the correct ticket!
Enter a ticket option you would like to order:
        Press 1 for 1 person
        Press 2 for 2 people
        Press 3 for 3 people
        Press 4 for 4 people
        Press 5 for 5 people
        Press 6 for 6 people
        Press 7 for 7 people
        Press 8 for 8 people
        Press 9 if you are VIP
        Press 10 if you want helicopter service
        Press 0 if you are a scalper
Enter your order: ^C
Program received signal SIGINT, Interrupt.
0x00007ffff7914992 in __GI___libc_read (fd=0x0, buf=0x5555555b28d0, nbytes=0x400) at ../sysdeps/unix/sysv/linux/read.c:26
26      ../sysdeps/unix/sysv/linux/read.c: No such file or directory.
[ Legend: Modified register | Code | Heap | Stack | String ]
```

Now you can see that the application is debuggable. Beforehand it was not because the anti-debugging protections crashed the application before you were allowed to debug it.

Run the command **p (void) debecdcafdbebfaed ()** and you can retrieve the flag.

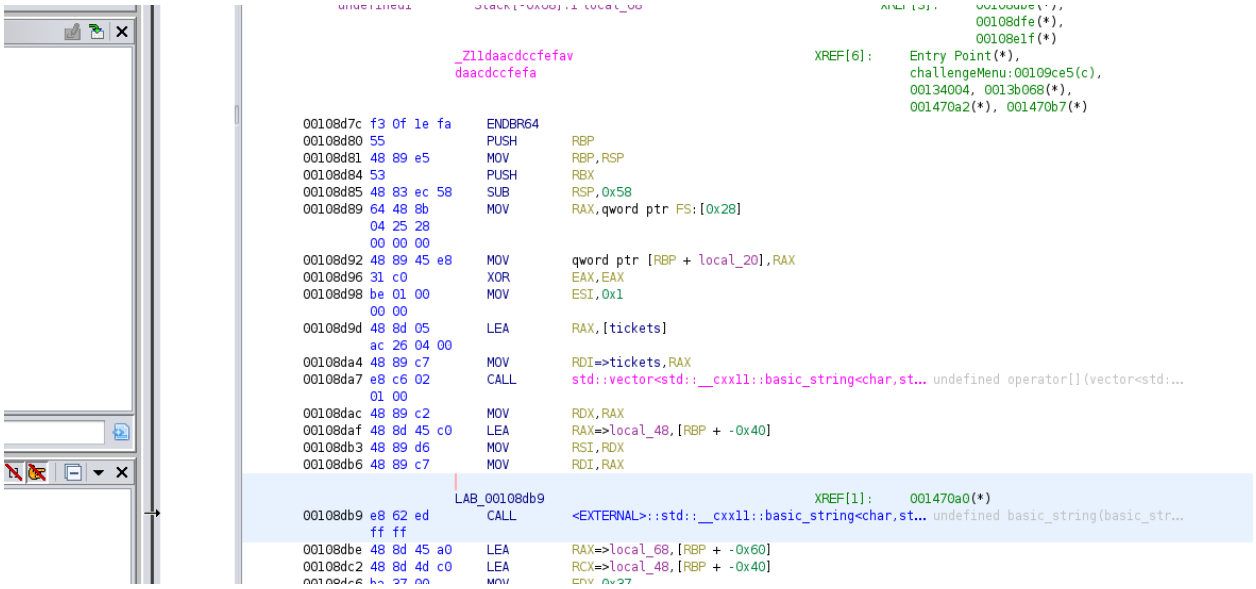
```
ERROR! TAMPERING DETECTED!
Using ticket:   be931e23cticketd7ac2f62
Holy crap. You actually got it. You will get 10,000 points instead of 100!
[New Thread 0x7ffff77ff640 (LWP 144322)]
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.10.6
Date: Sat, 18 Feb 2023 22:26:58 GMT
Content-type: application/octet-stream
Content-Length: 48
Last-Modified: Sat, 18 Feb 2023 22:26:28 GMT

TPZ{but_5H3_w34r5_5H0Rt_5KiRT5_I_W34R_t-5hirT5}
[Thread 0x7ffff77ff640 (LWP 144322) exited]
[Inferior 1 (process 144315) exited normally]
gef➤
```

```
[sudo] password for ctlister:
sudo: python: command not found
ctlister@crackamphetamine:~/Documents/creechctfchallenges/ticketmeister$ sudo python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
127.0.0.1 - - [18/Feb/2023 14:26:58] "GET /be931e23cticketd7ac2f62 HTTP/1.1" 200 -
```

# Possible Bugs

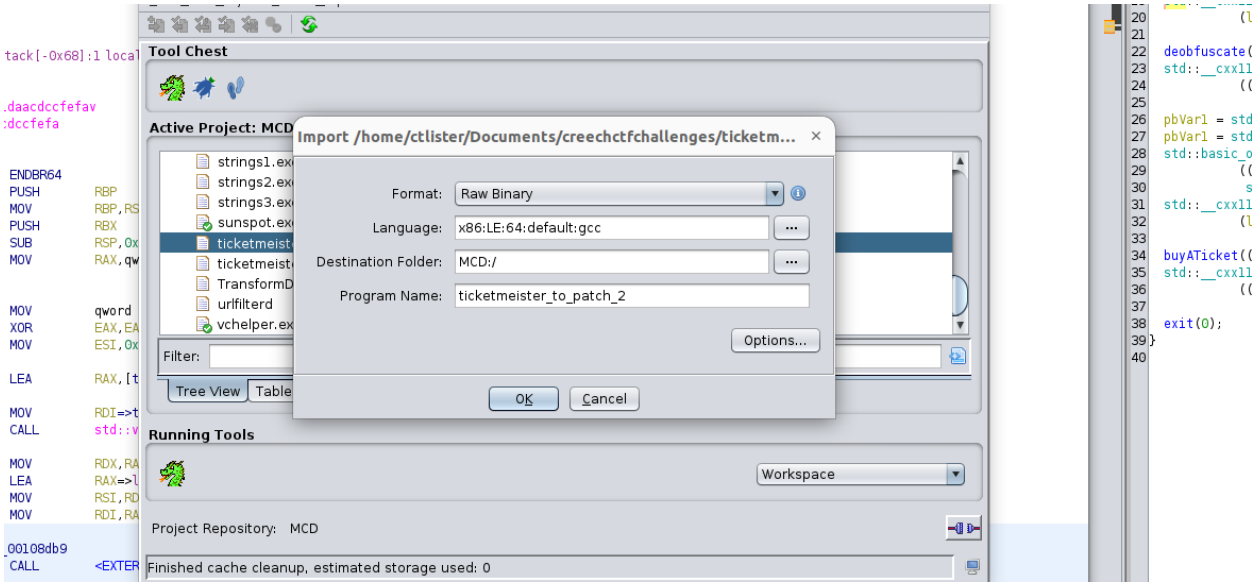
I have been exclusively using GHIDRA since 10.1.5 on three different platforms, Windows, Ubuntu, and Kali Linux. I noticed there were issues in earlier versions (before 10.2.2) where the base address is different (required for binary patching), in that event, if it happens, there is a workaround.

This is a aligned version of the binary with the function we want to patch, from the auto-analyze version.

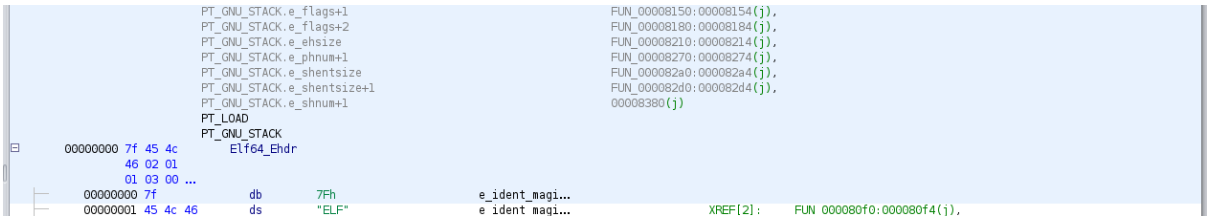



Let me load a unaligned version of the binary.[3] You do the same exact thing, load as raw, specify little-endian, 64-bit, gcc.

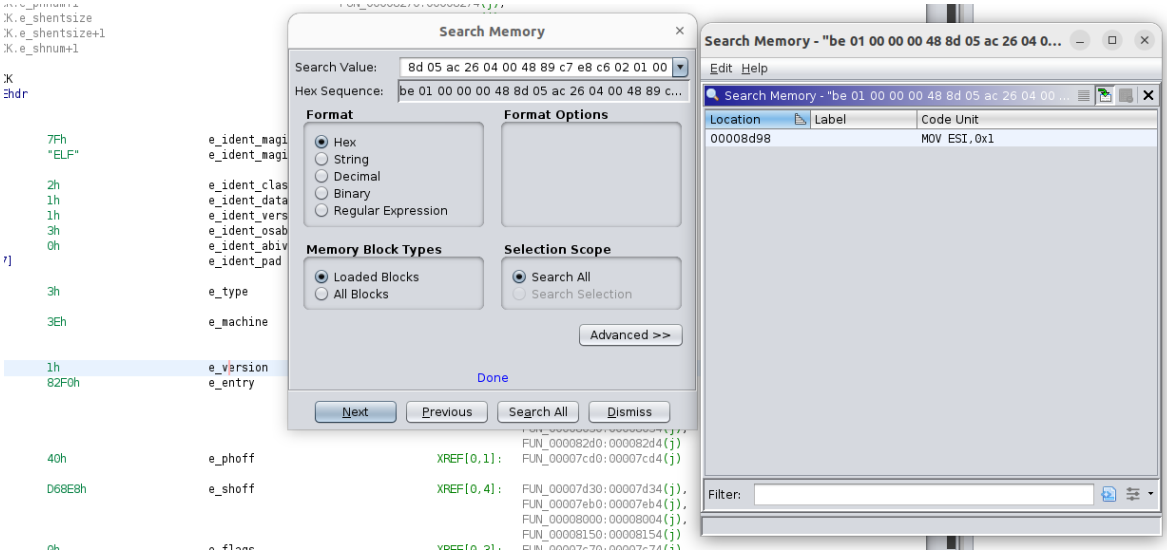
[3] I just tested GHIDRA versions 10.2.2 on Kali Linux VMs and 10.2.3 on Ubuntu 22.04, they show the same base address.



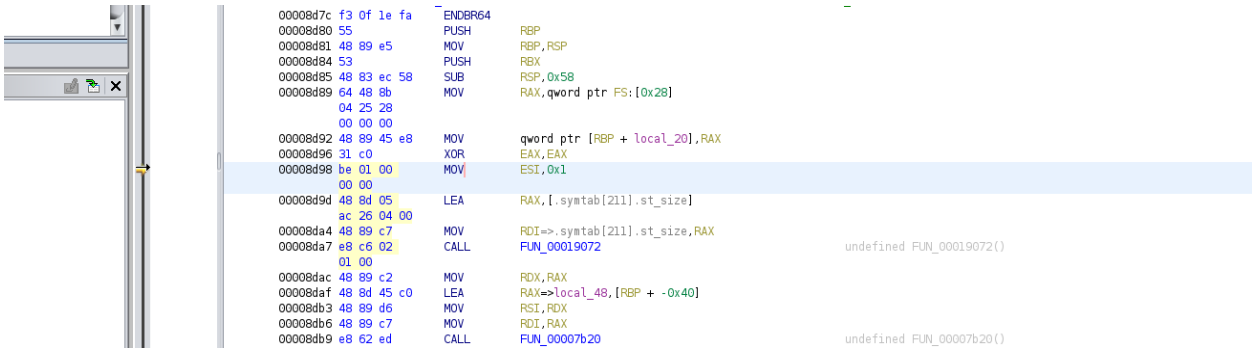

See how the base address is unaligned? Every opcode is a offset from the base address. In the event this happens during the CTF, you can use the opcode search method instead



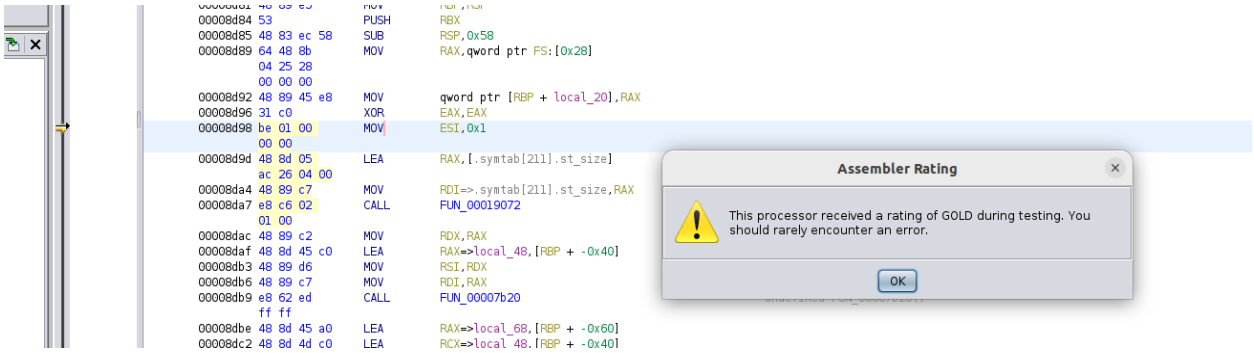

The opcodes you want to search for (the more exact the better) is **be 01 00 00 00 48 8d 05 ac 26 04 00 48 89 c7 e8 c6 02 01 00**, so press **s**, and search for it in the raw disassembly section.




Double click to navigate back to the memory address




Press Ctrl + Shift + G to start the disassembler/assembler for patching. Click OK and change MOV ESI, 0x1 to MOV ESI 0x0 to point the function to the correct array element.




Start up the app and press 1 and you get the flag.


00008d84 53 PUSH RBX
00008d85 48 83 ec 58 SUB RSP,0x58
00008d89 64 48 8b 04 25 28 00 00 00 MOV RAX,qword ptr FS:[0x28]
00008d92 48 89 45 e8 MOV qword ptr [RBP + local_20],RAX
00008d96 31 c0 XOR EAX,EAX
00008d98 be 00 00 00 00 MOV ESI,0x0
00008d9d 48 8d 05 ac 26 04 00 LEA RAX,[.symtab[211].st_size]
00008da4 48 89 c7 MOV RDI=>.symtab[211].st_size,RAX
00008da7 e8 c6 02 01 00 CALL FUN_00019072 undefined FUN_00019072()
00008dac 48 89 c2 MOV RDX,RAX
00008daf 48 8d 45 c0 LEA RAX=>local_48,[RBP + -0x40]
00008db3 48 89 d6 MOV RSI,RDX
00008db6 48 89 c7 MOV RDI,RAX
00008db9 e8 62 ed ff ff CALL FUN_00007b20 undefined FUN_00007b20()
00008dbe 48 8d 45 a0 LEA RAX=>local_68,[RBP + -0x60]
00008dc2 48 8d 4d c0 LEA RCX=>local_48,[RBP + -0x40]

# GHIDRA Hotkeys Reference/Cheat-Sheet

| | |
|---|---|
| g | Opens the go to address menu |
| c | Clear code bytes, useful for opcode searching |
| d | Disassemble raw assembly to make it more readable |
| s | Open the binary search menu |
| Ctrl +Shift + G | Attempt to apply a binary patch |
| o | Save output to file, options include raw binary, GHIDRA Zip File, etc. |
| Ctrl + T | Open the Symbols Table. Very useful for attempting to find vulnerabilities in higher level languages like Golang, Nim, Rust, etc. For example, I solved a few challenges from Red Team CTF in DEFCON 30 in a ransomware binary written in Nim by looking at all of the C Standard Library Calls |
| ; | Add comment menu |
| Ctrl + B | Open bookmarks Menu (normally opened on bottom) |
| Ctrl + D | Add bookmark |
| Alt + Left Arrow | Move to previous screen (for backtracking progress) |
| Alt + Right Arrow | Move to next screen |
| [Tools] -> [Program Differences] | GHIDRA's built-in "bin-diffing" feature. Useful for comparing two binaries or verifying binary patches. |
| I | Import file in GHIDRA main menu |
| Enter | Use default tool in GHIDRA main menu |
| Ctrl + Shift + F | Show cross-references "xrefs" of a specific function |