Analysis of Looking For Tokens (LFT) Anti – Cheat

deinero#1641
eternablue#9117
https://discord.gg/VGZ7euMD37

Introduction

This analysis explores the depths and procedures that are deployed by LFT in order to stop cheaters from winning money in their “tokens”, which is just another way of wording wagering money in Fortnite games. In this article I will describe the methods deployed by this anti-cheat in order to prevent cheaters, and explain the ethics behind deploying this anti cheat itself.

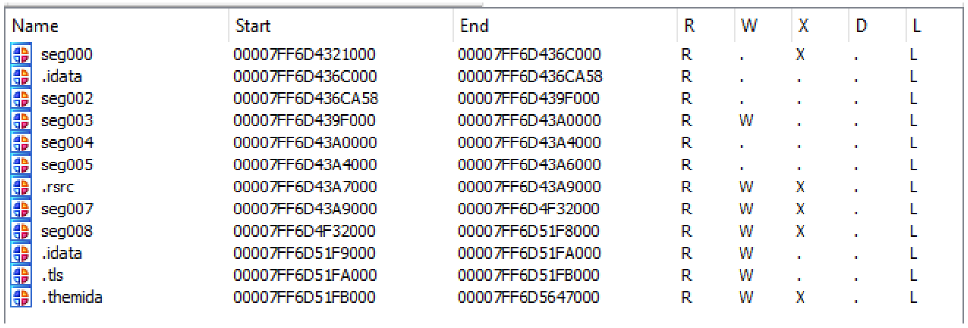
Anti Cheat Overview

To start off, this anti cheat is not heuristic based, or well they “attempt” to make it heuristic based anyway. The main way they attempt to make it heuristic based is by sending discord webhook messages to some discord channel they have, which stores some information about the player, followed with a screenshot of their current screen. These are then manually reviewed, and then a decision is made whether it was a false positive or not, and then you may get banned from the site.

Anti Cheat Components

1) Segmentations

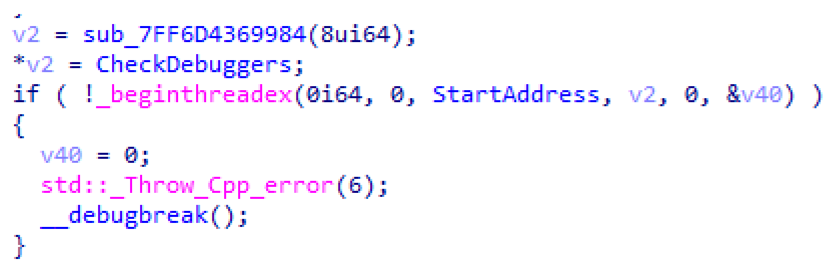
The screenshot attached below shows the segments that the anti cheat is composed of. As the screenshot shows, there are multiple segments with RWX flags (Read Write Execute), which can allow for DLL injection into the actual process and being unnoticed. This will allow you to then hook functions within the actual anti cheat and work your way around it.



Name	Start	End	R	W	X	D	L
seg000	00007FF6D4321000	00007FF6D436C000	R	.	X	.	L
.idata	00007FF6D436C000	00007FF6D436CA58	R	.	.	.	L
seg002	00007FF6D436CA58	00007FF6D439F000	R	.	.	.	L
seg003	00007FF6D439F000	00007FF6D43A0000	R	W	.	.	L
seg004	00007FF6D43A0000	00007FF6D43A4000	R	.	.	.	L
seg005	00007FF6D43A4000	00007FF6D43A6000	R	.	.	.	L
.rsrc	00007FF6D43A7000	00007FF6D43A9000	R	W	X	.	L
seg007	00007FF6D43A9000	00007FF6D4F32000	R	W	X	.	L
seg008	00007FF6D4F32000	00007FF6D51F8000	R	W	X	.	L
.idata	00007FF6D51F9000	00007FF6D51FA000	R	W	.	.	L
.tls	00007FF6D51FA000	00007FF6D51FB000	R	W	.	.	L
.themida	00007FF6D51FB000	00007FF6D5647000	R	W	X	.	L

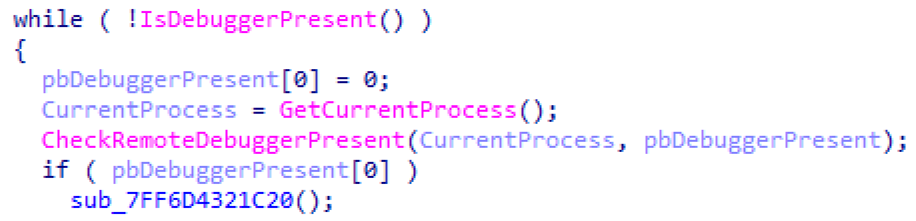
2) Debugger Checks [Security]

Now we can explore the actual security of the anti cheat itself to prevent reversing. The current PE I have loaded into IDA is the anti cheat dumped by scylla on my VM.



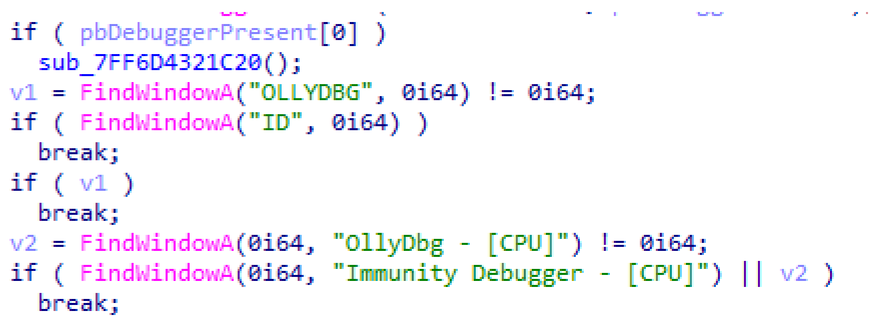
```
v2 = sub_7FF6D4369984(8ui64);
*v2 = CheckDebuggers;
if ( !_beginthreadex(0i64, 0, StartAddress, v2, 0, &v40) )
{
  v40 = 0;
  std::_Throw_Cpp_error(6);
  __debugbreak();
}
```

As this screenshot above shows, they start a thread to check for debuggers. The major flaw in this is that the actual thread itself can just be nop'd. If you set the create thread to non operating code, you can then attach any debuggers / do any memory options.



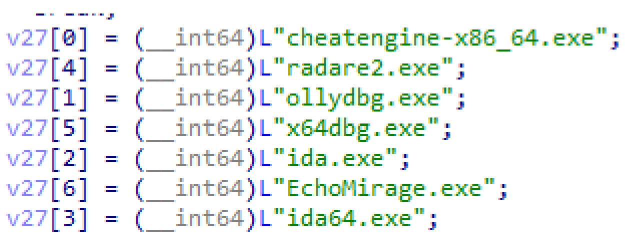
```
while ( !IsDebuggerPresent() )
{
  pbDebuggerPresent[0] = 0;
  CurrentProcess = GetCurrentProcess();
  CheckRemoteDebuggerPresent(CurrentProcess, pbDebuggerPresent);
  if ( pbDebuggerPresent[0] )
    sub_7FF6D4321C20();
```

This is the actual while loop in which the thread runs in, on the condition that while there is no debugger present, this code will run. If the condition of a debugger present ever returns true, it will run a function which will output a message box with the contents “SOFTWARE MEMORY ACCESSED BY THIRD-PARTY SOFTWARE”. It also important to mention that IsDebuggerPresent() is a function from the windows API, and the function is very inefficient and by far not good enough to be used on an actual “anti cheat”.



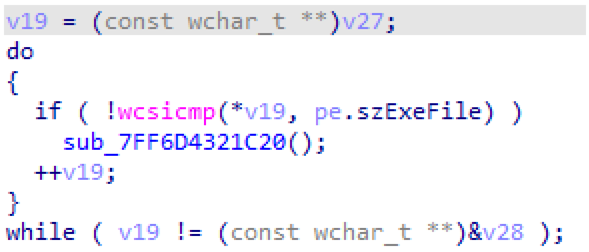
```
if ( pbDebuggerPresent[0] )
  sub_7FF6D4321C20();
v1 = FindWindowA("OLLYDBG", 0i64) != 0i64;
if ( FindWindowA("ID", 0i64) )
  break;
if ( v1 )
  break;
v2 = FindWindowA(0i64, "OllyDbg - [CPU]") != 0i64;
if ( FindWindowA(0i64, "Immunity Debugger - [CPU]") || v2 )
  break;
```

This following code looks for a window with a the following names. This check can just easily be bypassed by changing the name of the window, given that a lot of these tools that are checked here are open source so it can be changed by anyone. This is just a very inefficient, pointless check.



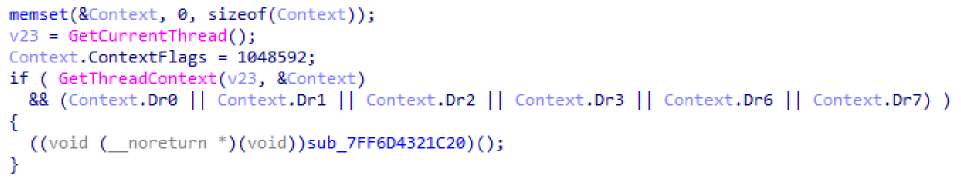
```
v27[0] = (__int64)L"cheatengine-x86_64.exe";
v27[4] = (__int64)L"radare2.exe";
v27[1] = (__int64)L"ollydbg.exe";
v27[5] = (__int64)L"x64dbg.exe";
v27[2] = (__int64)L"ida.exe";
v27[6] = (__int64)L"EchoMirage.exe";
v27[3] = (__int64)L"ida64.exe";
```

The variable v27 stores the names of processes that the anti cheat uses, such as cheat engine etc. This is important for the next part.



```
v19 = (const wchar_t **)v27;
do
{
  if ( !wcsicmp(*v19, pe.szExeFile) )
    sub_7FF6D4321C20();
  ++v19;
}
while ( v19 != (const wchar_t **)&v28 );
```

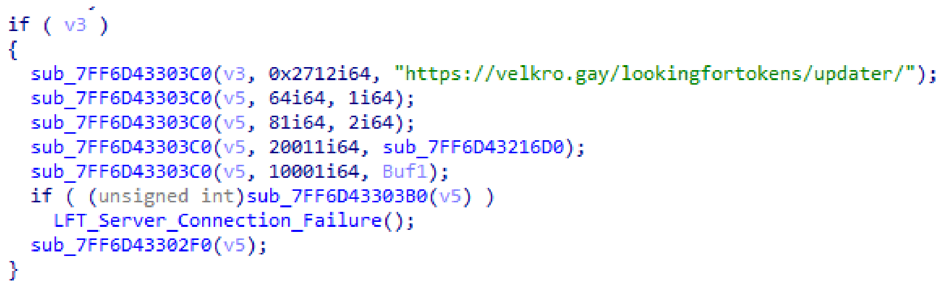
This part will enumerate every running process, and compare the process name to the variable v19, which now holds all the contents of v27 (so it has all the names of the processes). The processes itself can just be renamed so I also don't see a point of this.



```
memset(&Context, 0, sizeof(Context));
v23 = GetCurrentThread();
Context.ContextFlags = 1048592;
if ( GetThreadContext(v23, &Context)
  && (Context.Dr0 || Context.Dr1 || Context.Dr2 || Context.Dr3 || Context.Dr6 || Context.Dr7) )
{
  ((void (__noreturn *)(void))sub_7FF6D4321C20)();
}
```

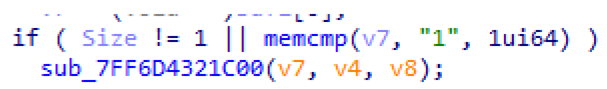
This section of code checks of debug related registers have been set for the current threads context, and if they are set, the function is called which prints a message box saying "SOFTWARE MEMORY ACCESSED BY THIRD-PARTY SOFTWARE" and then calls the abort function afterwards.

3) CURL Requests / AC version checking



```
if ( v3 )
{
  sub_7FF6D43303C0(v3, 0x2712i64, "https://velkro.gay/lookingfortokens/updater/");
  sub_7FF6D43303C0(v5, 64i64, 1i64);
  sub_7FF6D43303C0(v5, 81i64, 2i64);
  sub_7FF6D43303C0(v5, 20011i64, sub_7FF6D43216D0);
  sub_7FF6D43303C0(v5, 10001i64, Buf1);
  if ( (unsigned int)sub_7FF6D43303B0(v5) )
    LFT_Server_Connection_Failure();
  sub_7FF6D43302F0(v5);
}
```

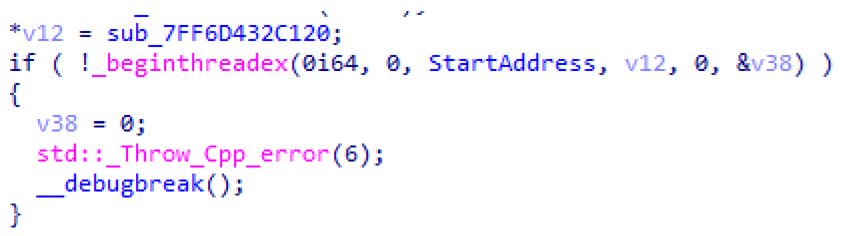
The condition of v3 is a function to retrieve the address of memory specific functions such as malloc, free, realloc etc. On the condition all those functions are retrieved, another function is called with the url https://velkro.gay/lookingfortokens/updater/. This website just shows a what seems to be a version number, which is later compared here :



```
if ( Size != 1 || memcmp(v7, "1", 1ui64) )
  sub_7FF6D4321C00(v7, v4, v8);
```

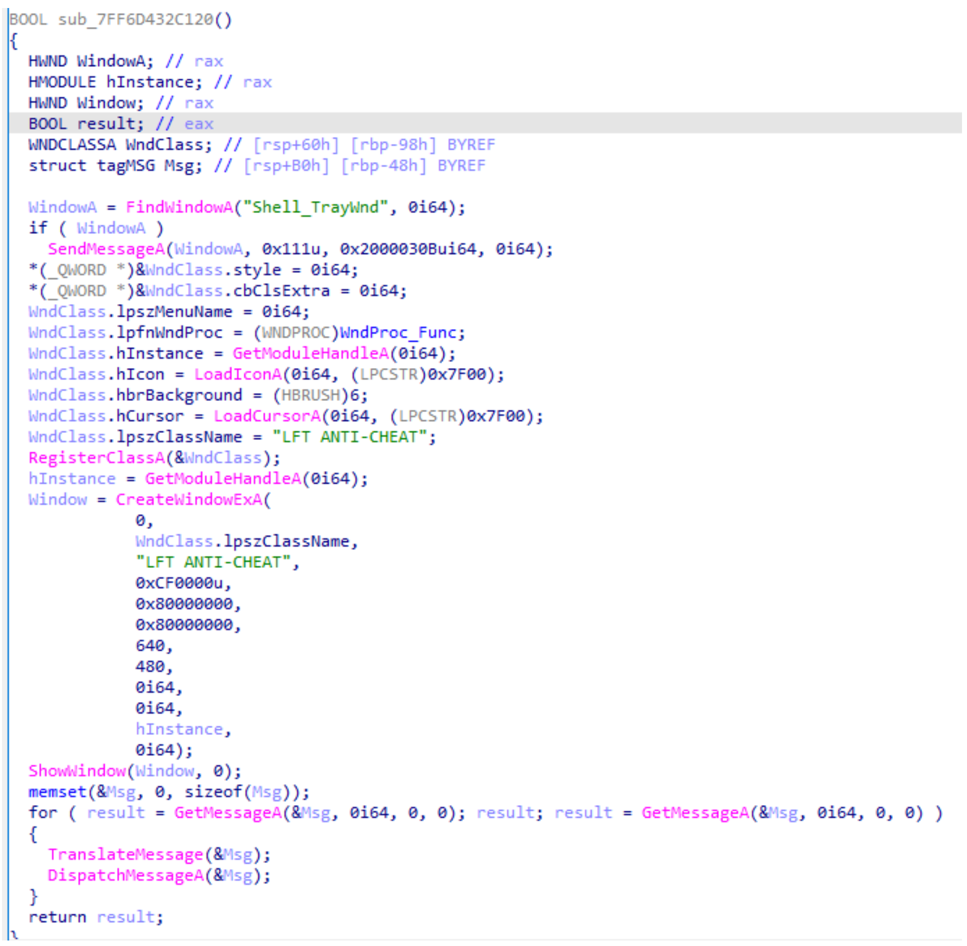
If it is not ”1”, you will get an error message box telling you to download the new one from the website.

4) LFT AC Window / WndProc Function



```
*v12 = sub_7FF6D432C120;
if ( !_beginthreadex(0i64, 0, StartAddress, v12, 0, &v38) )
{
  v38 = 0;
  std::_Throw_Cpp_error(6);
  __debugbreak();
}
```

This following code starts a thread with the following code :

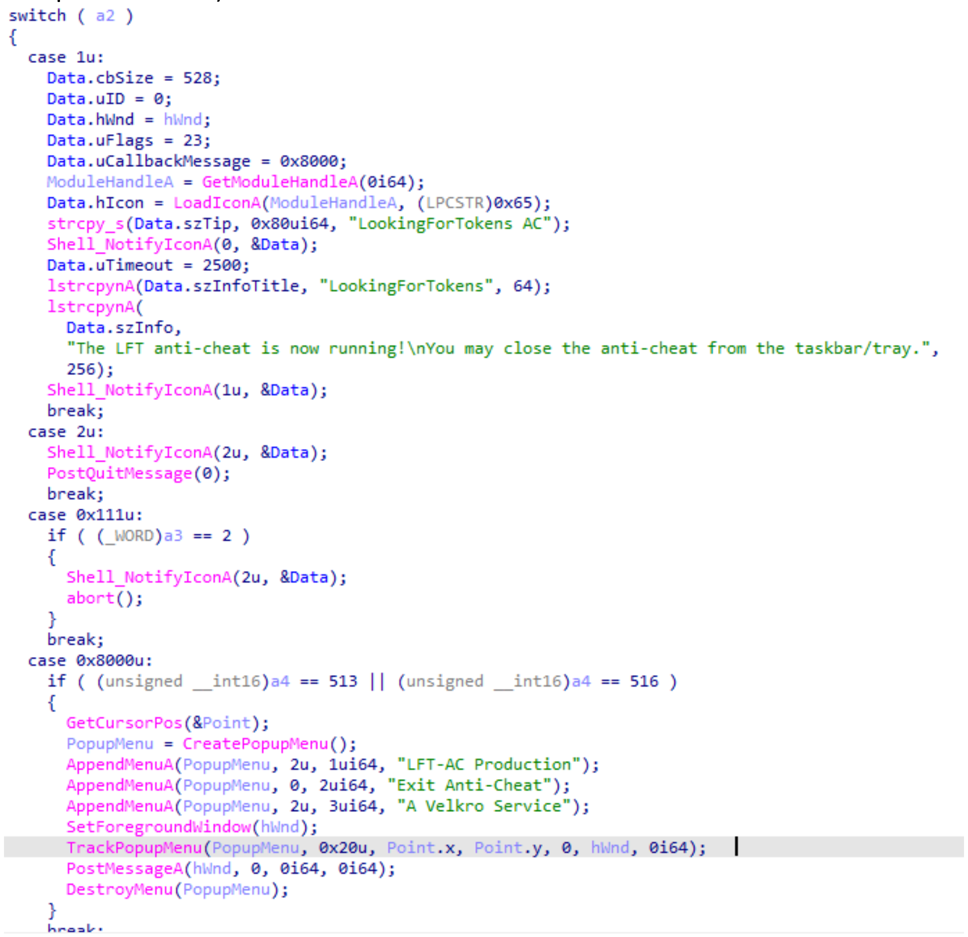


```
BOOL sub_7FF6D432C120()
{
  HWND WindowA; // rax
  HMODULE hInstance; // rax
  HWND Window; // rax
  BOOL result; // eax
  WNDCLASSA WndClass; // [rsp+60h] [rbp-98h] BYREF
  struct tagMSG Msg; // [rsp+B0h] [rbp-48h] BYREF

  WindowA = FindWindowA("Shell_TrayWnd", 0i64);
  if ( WindowA )
    SendMessageA(WindowA, 0x111u, 0x2000030Bui64, 0i64);
  *(_QWORD *)&WndClass.style = 0i64;
  *(_QWORD *)&WndClass.cbClsExtra = 0i64;
  WndClass.lpszMenuName = 0i64;
  WndClass.lpfnWndProc = (WNDPROC)WndProc_Func;
  WndClass.hInstance = GetModuleHandleA(0i64);
  WndClass.hIcon = LoadIconA(0i64, (LPCSTR)0x7F00);
  WndClass.hbrBackground = (HBRUSH)6;
  WndClass.hCursor = LoadCursorA(0i64, (LPCSTR)0x7F00);
  WndClass.lpszClassName = "LFT ANTI-CHEAT";
  RegisterClassA(&WndClass);
  hInstance = GetModuleHandleA(0i64);
  Window = CreateWindowExA(
             0,
             WndClass.lpszClassName,
             "LFT ANTI-CHEAT",
             0xCF0000u,
             0x80000000,
             0x80000000,
             640,
             480,
             0i64,
             0i64,
             hInstance,
             0i64);
  ShowWindow(Window, 0);
  memset(&Msg, 0, sizeof(Msg));
  for ( result = GetMessageA(&Msg, 0i64, 0, 0); result; result = GetMessageA(&Msg, 0i64, 0, 0) )
  {
    TranslateMessage(&Msg);
    DispatchMessageA(&Msg);
  }
  return result;
}
```

This is code creates a windows with the name “LFT ANTI-CHEAT", and does some other utility such as loading the widnow icon etc. This windows is then created by calling the CreateWidnowExA function.

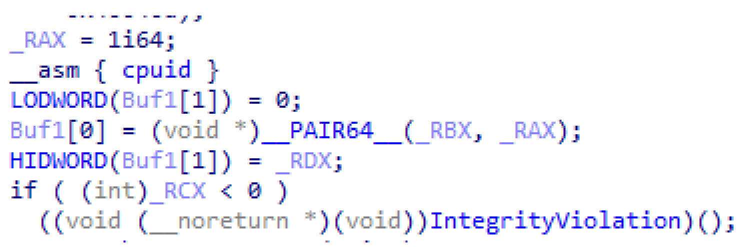
Now moving onto the WndProc function, we can see a switch case with the variable a2 (standard for a wndproc function) :



```
switch ( a2 )
{
  case 1u:
    Data.cbSize = 528;
    Data.uID = 0;
    Data.hWnd = hWnd;
    Data.uFlags = 23;
    Data.uCallbackMessage = 0x8000;
    ModuleHandleA = GetModuleHandleA(0i64);
    Data.hIcon = LoadIconA(ModuleHandleA, (LPCSTR)0x65);
    strcpy_s(Data.szTip, 0x80ui64, "LookingForTokens AC");
    Shell_NotifyIconA(0, &Data);
    Data.uTimeout = 2500;
    lstrcpynA(Data.szInfoTitle, "LookingForTokens", 64);
    lstrcpynA(
      Data.szInfo,
      "The LFT anti-cheat is now running!\nYou may close the anti-cheat from the taskbar/tray.",
      256);
    Shell_NotifyIconA(1u, &Data);
    break;
  case 2u:
    Shell_NotifyIconA(2u, &Data);
    PostQuitMessage(0);
    break;
  case 0x111u:
    if ( (_WORD)a3 == 2 )
    {
      Shell_NotifyIconA(2u, &Data);
      abort();
    }
    break;
  case 0x8000u:
    if ( (unsigned __int16)a4 == 513 || (unsigned __int16)a4 == 516 )
    {
      GetCursorPos(&Point);
      PopupMenu = CreatePopupMenu();
      AppendMenuA(PopupMenu, 2u, 1ui64, "LFT-AC Production");
      AppendMenuA(PopupMenu, 0, 2ui64, "Exit Anti-Cheat");
      AppendMenuA(PopupMenu, 2u, 3ui64, "A Velkro Service");
      SetForegroundWindow(hWnd);
      TrackPopupMenu(PopupMenu, 0x20u, Point.x, Point.y, 0, hWnd, 0i64);
      PostMessageA(hWnd, 0, 0i64, 0i64);
      DestroyMenu(PopupMenu);
    }
    break;
```

On the case of 1u (WM_CREATE), it will initialize the data of the icon, setup a system tray icon provided other variables, then notifies the shell about the new system tray icon. This is what you can see when you open up the system tray and see the LFT logo.
The case of 2u (WM_DESTROY) and 0x111u (WM_COMMAND), handle basic window things that there is no need to get in to.
If the message is a custom message (0x8000u), and the event is a mouse click, it will store information about the cursor, create and append a pop up menu, post a message to the menu and then destroy it. This is also nothing unique or unusual.

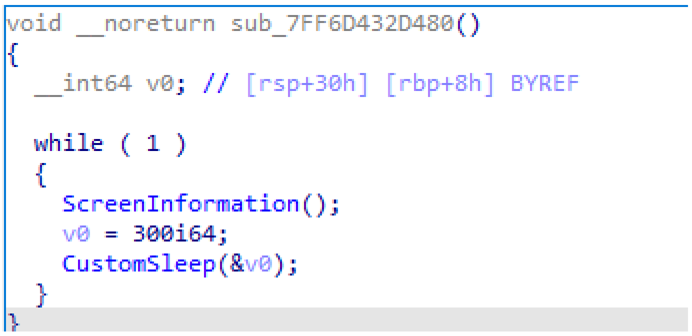
5) Assembly [VM check]



```
_RAX = 1i64;
__asm { cpuid }
LODWORD(Buf1[1]) = 0;
Buf1[0] = (void *)__PAIR64__(_RBX, _RAX);
HIDWORD(Buf1[1]) = _RDX;
if ( (int)_RCX < 0 )
  ((void (__noreturn *)(void))IntegrityViolation)();
```

This following code performs some inline assembly. __asm { cpuid } fills several registers with information about the CPU, such as the vendor, cache information etc. They then compares the int value of RCX and if it is < 0, it will throw an integrity violation. This is a simple virtual machine / hypervisor check, which is nothing that special at all.

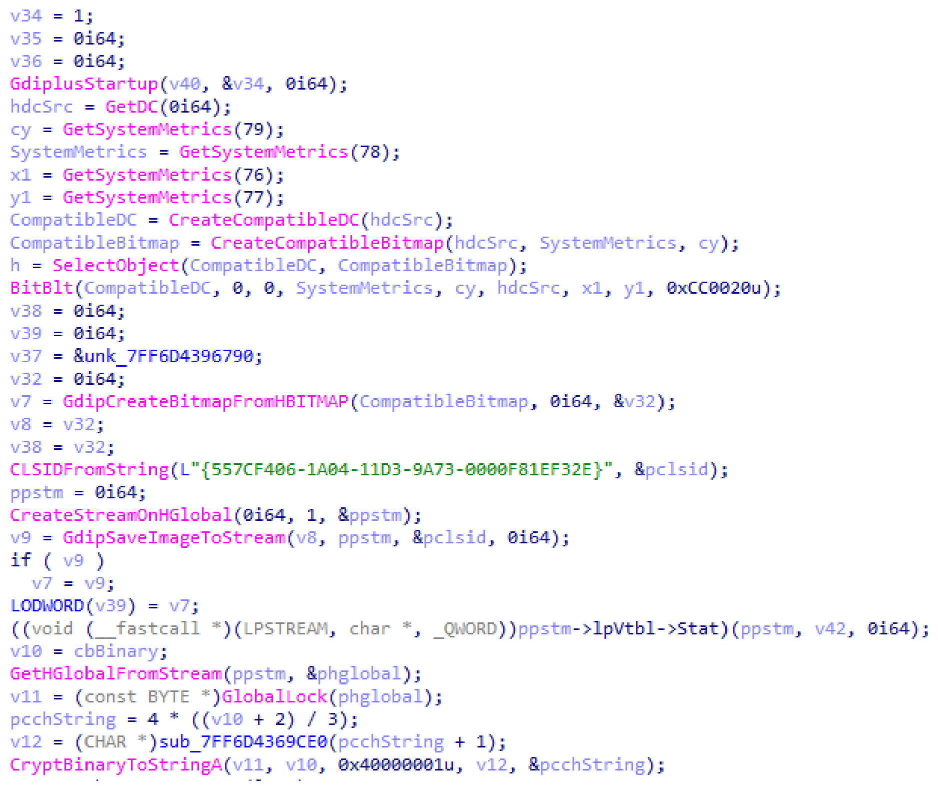
6) Screenshot Screen



```
void __noreturn sub_7FF6D432D480()
{
  __int64 v0; // [rsp+30h] [rbp+8h] BYREF

  while ( 1 )
  {
    ScreenInformation();
    v0 = 300i64;
    CustomSleep(&v0);
  }
}
```

This function is also ran in a thread, so runs continuously in a while true loop. As we can seen, the function ScreenInformation() is being constantly called in this loop. The following snapshot is a small section of that function :

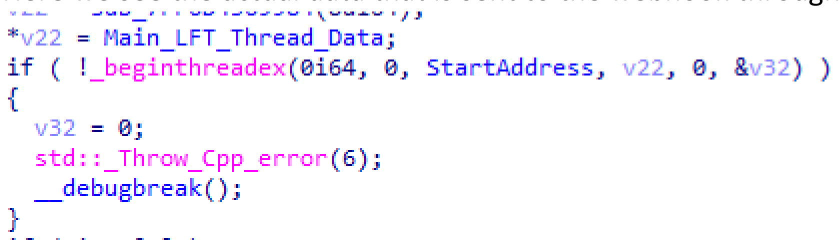


```
v34 = 1;
v35 = 0i64;
v36 = 0i64;
GdiplusStartup(v40, &v34, 0i64);
hdcSrc = GetDC(0i64);
cy = GetSystemMetrics(79);
SystemMetrics = GetSystemMetrics(78);
x1 = GetSystemMetrics(76);
y1 = GetSystemMetrics(77);
CompatibleDC = CreateCompatibleDC(hdcSrc);
CompatibleBitmap = CreateCompatibleBitmap(hdcSrc, SystemMetrics, cy);
h = SelectObject(CompatibleDC, CompatibleBitmap);
BitBlt(CompatibleDC, 0, 0, SystemMetrics, cy, hdcSrc, x1, y1, 0xCC0020u);
v38 = 0i64;
v39 = 0i64;
v37 = &unk_7FF6D4396790;
v32 = 0i64;
v7 = GdipCreateBitmapFromHBITMAP(CompatibleBitmap, 0i64, &v32);
v8 = v32;
v38 = v32;
CLSIDFromString(L"{557CF406-1A04-11D3-9A73-0000F81EF32E}", &pclsid);
ppstm = 0i64;
CreateStreamOnHGlobal(0i64, 1, &ppstm);
v9 = GdipSaveImageToStream(v8, ppstm, &pclsid, 0i64);
if ( v9 )
  v7 = v9;
LODWORD(v39) = v7;
((void (__fastcall *)(LPSTREAM, char *, _QWORD))ppstm->lpVtbl->Stat)(ppstm, v42, 0i64);
v10 = cbBinary;
GetHGlobalFromStream(ppstm, &phglobal);
v11 = (const BYTE *)GlobalLock(phglobal);
pcchString = 4 * ((v10 + 2) / 3);
v12 = (CHAR *)sub_7FF6D4369CE0(pcchString + 1);
CryptBinaryToStringA(v11, v10, 0x40000001u, v12, &pcchString);
```

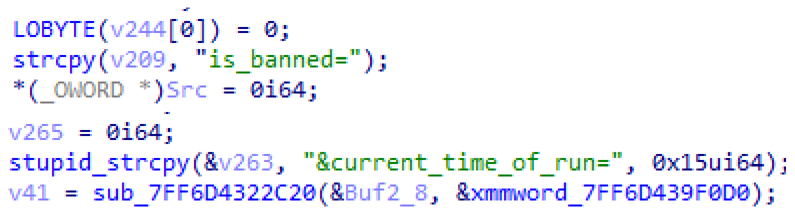
This function takes a screenshot of the users screen, and it is the screenshot that it then sent to their discord webhook afterwards.

7) LFT User Data Thread

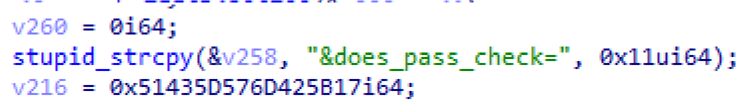
Here we see the actual data that is sent to the webhook through network requests



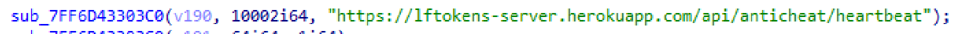
```
*v22 = Main_LFT_Thread_Data;
if ( !_beginthreadex(0i64, 0, StartAddress, v22, 0, &v32) )
{
  v32 = 0;
  std::_Throw_Cpp_error(6);
  __debugbreak();
}
```



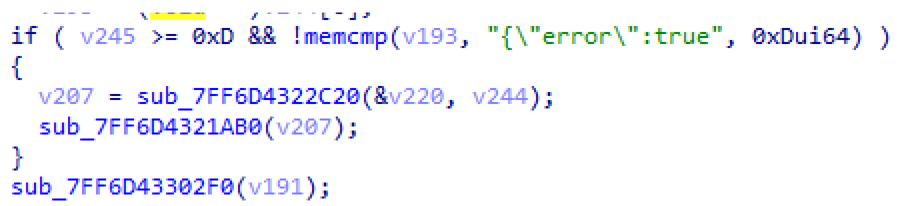
```
LOBYTE(v244[0]) = 0;
strcpy(v209, "is_banned=");
*(_OWORD *)Src = 0i64;
v265 = 0i64;
stupid_strcpy(&v263, "&current_time_of_run=", 0x15ui64);
v41 = sub_7FF6D4322C20(&Buf2_8, &xmmword_7FF6D439F0D0);
```



```
v260 = 0i64;
stupid_strcpy(&v258, "&does_pass_check=", 0x11ui64);
v216 = 0x51435D576D425B17i64;
```



```
sub_7FF6D43303C0(v190, 10002i64, "https://lftokens-server.herokuapp.com/api/anticheat/heartbeat");
```



```
if ( v245 >= 0xD && !memcmp(v193, "{\"error\":true", 0xDui64) )
{
  v207 = sub_7FF6D4322C20(&v220, v244);
  sub_7FF6D4321AB0(v207);
}
sub_7FF6D43302F0(v191);
```

This function in general has bit of obfuscation / xor going through it, however these parts are left untouched. These functions just send the data to the actual server, and can be altered / changed to make the server think otherwise.

8) Extra Stuff

Project Dir : D:\\Project Velkro\\LFTACUM\\x64\\Release\\6449ik1hk28285i47409j13j.pdb
Install Dir : C:\\ProgramData\\LFTAC\\

CURL / FTP Network Requests :
https://lftokens-server.herokuapp.com/api/anticheat/code
https://lftokens-server.herokuapp.com/api/anticheat/heartbeat
https://velkro.gay/lookingfortokens/updater/
https://velkro.lol/lookingfortokens/logs/images/

To access the data of these links it requires some network reverse engineering

Extra thanks to dormant, zephin, Smelo, upr1sing

Emulating / Disabling Anti Cheat

- Prevent certain threads / functions running by setting them as NOP.
- Injecting a dll into the process (can exploit RWX sections), and hooking certain functions to see the return values / output. This can be used to hook the, for example, network requests exchanged to the actual server.
- Using a better dumper, for the purpose of this I used scylla, and I just used the base settings (unmodified). If you can dump and unpack themida with it, you can see a lot more undeneath.
- Doing live analysis, for example using x64dbg. This was all done using static analysis, and during live analysis you can see a better view on how everything runs / works.