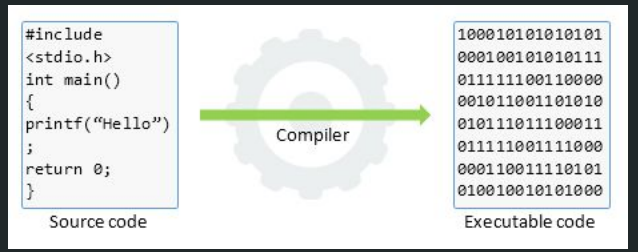
Intro to Reverse Engineering

with a not so good reverse-engineer

What is Reverse Engineering?

- A typical program is compiled into a bytecode / assembly language from source code



- Reverse engineering is the opposite of this process.
- If we ever wanted to get an idea of **exactly** what the binary does, we cannot be sure without peeking inside
- It's done to find security exploits, audit software, analyze malware, perform IP theft, create open-source drivers, crack software and create game hacks

Choose your weapon

- Our target applications are going to be running on Windows on the x86 architecture
- How are we going to reverse engineer them?
- There are a LOT of tools, all with pros and cons:
    - IDA Pro + Hex Rays decompiler
    - Ghidra
    - Ollydbg
    - Binary Ninja
    - Radare2 + Cutter UI
    - Hopper

- IDA Pro has always been the most mature disassembler and was king for 15+ years.
- However, it costs ~$7000, $1879 for the disassembler & $5258 for the Hex-Rays decompiler
- The freeware edition is very powerful but lacks Hex-Rays, you may need to *source* it somewhere else
- The alternative is Ghidra, released by the NSA for... free. And it includes a decompiler
- But hey, we're going to use IDA cause it makes us feel rich

Disassembler vs Decompiler

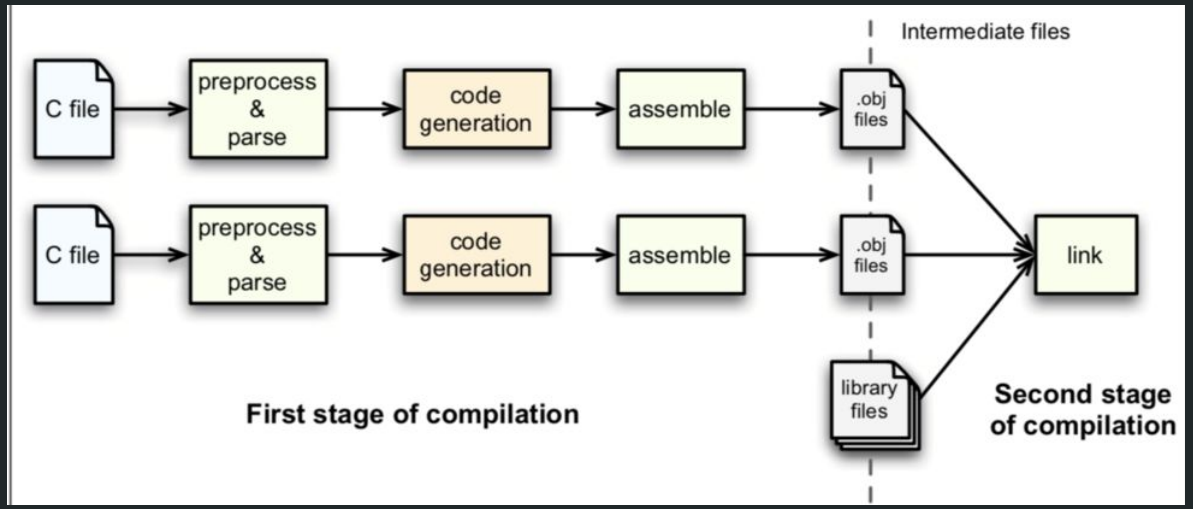
- A disassembler simply reads the machine code and translate it back into its relevant instructions.
- This is trivial in comparison to a decompiler
- A decompiler **turns disassembled code back into (somewhat pseudo) source code**

```
    sub_7FFA38B204F0 proc near

    var_28= qword ptr -28h
    var_20= qword ptr -20h
    var_18= qword ptr -18h
    var_10= dword ptr -10h
    arg_18= dword ptr  20h

000 sub     rsp, 48h
048 and     [rsp+48h+arg_18], 0
048 mov     r9b, r8b
048 movzx   r8d, dx
048 mov     rdx, rcx
048 lea     rcx, [rsp+48h+arg_18]
048 call    sub_7FFA38B28C24
048 mov     eax, [rsp+48h+arg_18]
048 lea     r9, aMilanreadrawda ; "MilanReadRawData"
048 mov     rcx, cs:pDebugStruct
048 lea     r8, aEGitGenevaRemo ; "e:\\git\\geneva_remote\\
048 mov     [rsp+48h+var_10], eax
048 mov     edx, 8
048 lea     rax, aExitX     ; "Exit:%x"
048 mov     [rsp+48h+var_18], rax
048 and     [rsp+48h+var_20], 0
048 mov     dword ptr [rsp+48h+var_28], 463h
048 call    DebugPrint      ; check if this is nullptr
                            ;
048 mov     eax, [rsp+48h+arg_18]
048 add     rsp, 48h
000 retn
    sub_7FFA38B204F0 endp
```

```
1  __int64 __fastcall sub_7FFA38B204F0(__int64 a1, unsigned __int16 a2, char a3, __int64 a4
2  {
3    __int64 v5; // [rsp+20h] [rbp-28h]
4    unsigned int v6; // [rsp+38h] [rbp-10h]
5    unsigned int v7; // [rsp+68h] [rbp+20h]
6
7    v7 = 0;
8    LOBYTE(a4) = a3;
9    sub_7FFA38B28C24(&v7, a1, a2, a4);
10   v6 = v7;
11   LODWORD(v5) = 1123;
12   DebugPrint(
13     (__int64)pDebugStruct,
14     8u,
15     L"e:\\git\\geneva_remote\\winfpcode\\driver\\common\\chipoperation\\milanfseries\\mc
16     L"MilanReadRawData",
17     v5,
18     0,
19     L"Exit:%x",
20     v6);
21   return v7;
22 }
```

Compilation 101

- We’re gonna need to know how a C and some basic C++ files end up as an executable, how are “dependencies” managed?



Compilation 101

- There exists two types of linkage, static linkage & dynamic linkage.
- Static linking a library means the library binary is copied into our application executable. When the application calls a library function, the address of the function to be called is simply hardcoded in
- Dynamic linking allows for a different library file to be loaded at runtime. This could mean we could update our library and as long as we kept the same **Application Binary Interface,** everything will still work.
- Typically, the OS libraries and C runtime (i.e the library that contains the implementations of C functions) are dynamically linked.
- This is where the term **Dynamic Link Library (.dll)** on Windows comes from. The Linux equivalent would be the **.so, shared object**
- These libraries are built to be relocatable, so they can be loaded at any memory address
- Should also be noted that we ourselves could dynamically load a library. More on this later

Portable Executables and Dynamic Linking

- How does the OS know what DLLs to load when a program boots?
- Portable Executables (.exe & .dll) have an import table
- An .exe which would import a function from a .dll would list the DLL name on it's Import table
- Whilst that DLL would list it's exported functions on it's export address table (EAT) of it's PE

- When we launch our program:
  - The OS reads our Import Table
  - Maps the requested library into the memory space
  - Adds a record to the Import Address Table of our application that points to the function
- When the application wants to call that function, it simply looks at it's IAT, follows the record to the loaded library and calls the function
- IDA, CFF Explorer Demo + Limits of Static Analysis

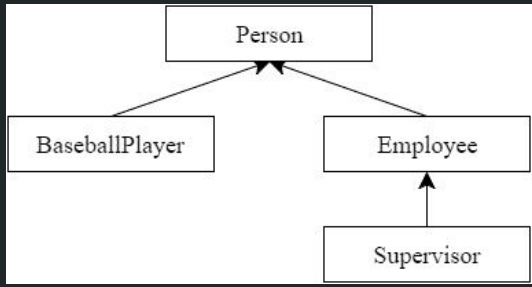
API Monitoring

- The nature of exports & imports means the symbol names must generally be human-readable.
- This mean it’s inevitable that an application will call a known function to us
- This makes it very easy to see the behaviour of the application, as we can see it use calls to the OS and public, known libraries
- These calls can easily be monitored with debugging breakpoints & other techniques
- API Monitor Demo + IDA Pro

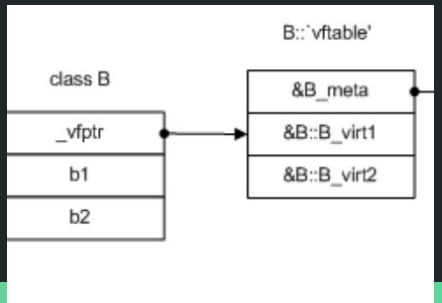
Classes

- Classes and structs members will be stored linearly in memory in the order they were declared
- There may be padding bytes in between items to align them to power of 2 boundaries
- We will investigate a way of modeling such items later
- When one class inherits another, the memory profile is simply their members concatenated together

Function run-time type information (RTTI) & vtables



- A byproduct of function inheritance in C++, specifically in MSVC
- RTTI is used to implement dynamic_cast and it's downcasting ability
- An instance of a class holds a pointer to a vtable as it's first member
- A vtable is a list of function pointers for each overridable method on that class
- When a class wants to override a method on a superclass, it uses a new vtable (and new pointer) with the relevant function pointer in the table changed to the overridden method
- https://godbolt.org/z/QFFjX6
- https://bit.ly/2NPNEdJ
- VTables store the name of the class types, giving us effective hints at what the class does

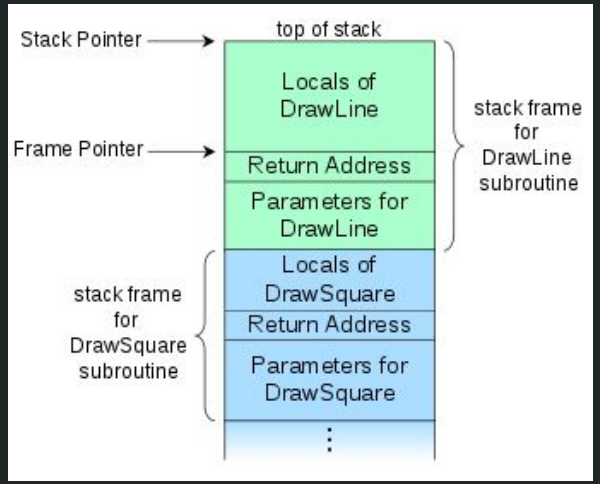


Strings

- Remember that IDA isn’t perfect and it may not find every string
- Make sure you set the string type in Compiler options correctly, Win32 often favors wchar_t 16bit characters more than char 8bits.
- Don’t underestimate the power of the xref feature

The Hard Work

- Now comes the most difficult part of reverse engineering: figuring out what the hell is going on
- The bad part: you're going to need to learn assembly
- The even worse part: you're going need to learn all the calling conventions
- https://en.wikipedia.org/wiki/X86_calling_conventions#List_of_x86_calling_conventions
- You're going to have to model structs as you examine function calls
- You're going to have to alias everything
- Demo?



Breaking It

- Now that we know how something works, how do we break it?
- Let’s do something fun: game hacks
- Generally game hacks can be categorized as two types: internal & external
- An external hack is one that lives outside the games process and modifies memory of the game using OS calls, namely ReadProcessMemory & WriteProcessMemory
- Some people believe this to be more “secure” against anti-cheats, but in general it tends to be pretty slow
- Internal game hacks inject code into the games process and runs it. An existing thread is either hijacked or we start a new one
- These perform just as fast as the game does, and you can even use the game’s internal APIs if you wish

External Hack Demo: ReClass & SigScans

- ReClass is a tool that allows you to model structs inside a process, you can then export them as C/C++ header files. We could then import them to IDA or in our hack code
- Sig Scanning is a method where we can scan the game process for a set of bytes.
- A signature can be made up of any number of bytes with “mask” bytes whose values are ignored during the search.
- You can generally make these signatures by hand
- CSGO Demo w/ CDebugOverlay, IVEntityList & s_pTextOverlays

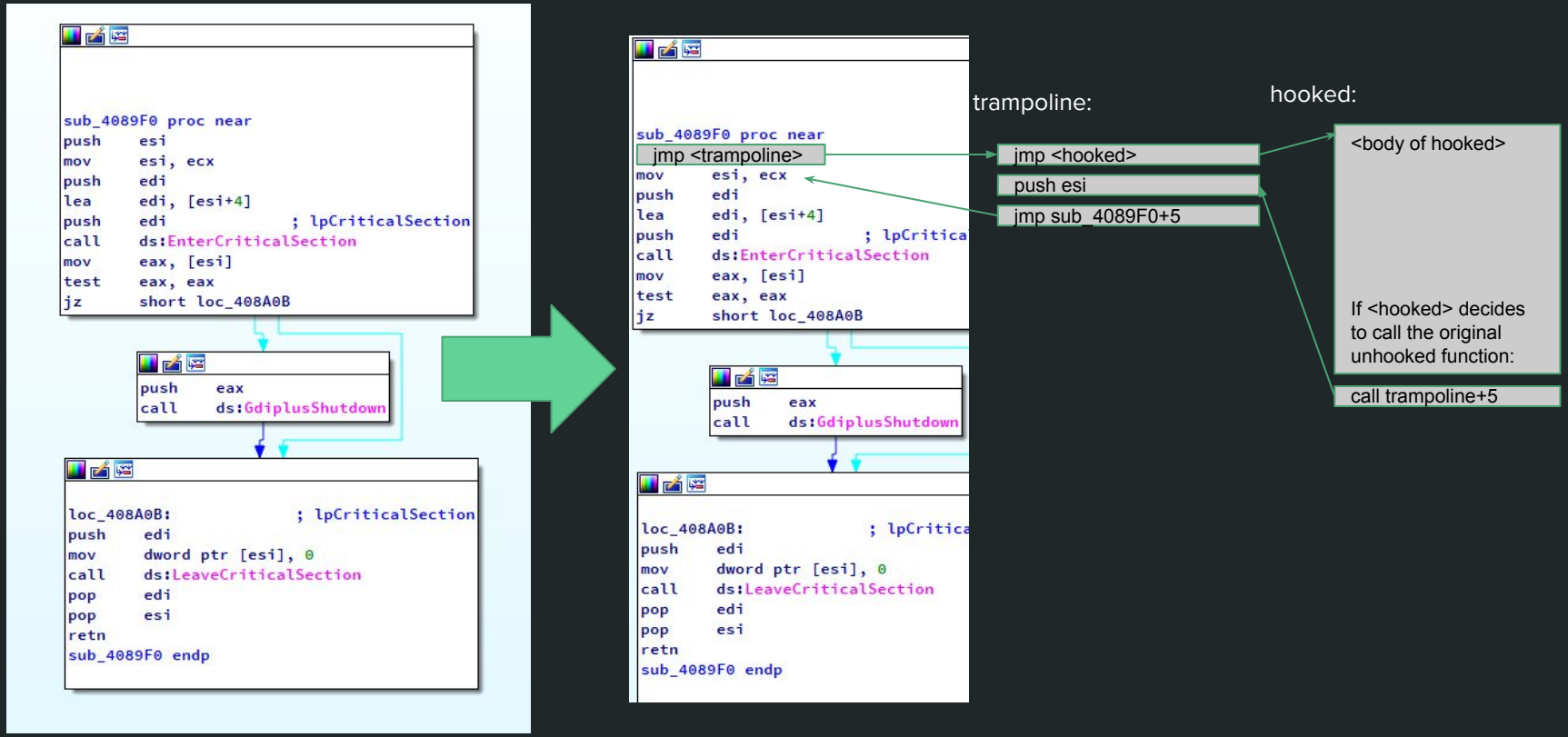
Internal Hacks: DLL Injection & Hooks

- We can build our game hack as a DLL and force the game to load it
- There are a few methods to do this:
    - Call CreateRemoteThread & use it to run LoadLibrary(“uwu.dll”)
    - Pause all threads, use NtSetContextThread
    - Manual mapping

- Once we’re in, what can we do?
    - Our main goal would be to “hook” functions
    - Function hooking is similar to writing an event handler
    - Whenever the hooked function gets called, our function gets called
    - We can modify any parameters that get passed, completely block the call, extend functionality, etc...

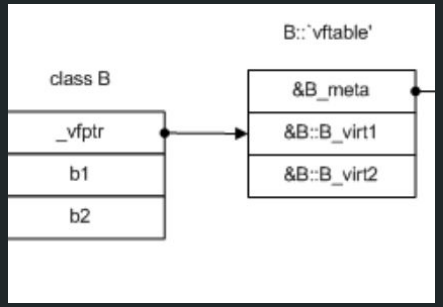
JMP Hooking

- JMP hooking involves modifying the contents of a function directly. This means that memory permissions of the page containing the function will need to be modified.
- We will need to replace the first 5 bytes of the function with a jmp to our “trampoline”
- We will need to allocate memory for the “trampoline” stub, the trampoline stub contains a jmp to our hooked function, and the instruction we removed above followed by a jmp back to the original function


sub_4089F0 proc near
push esi
mov esi, ecx
push edi
lea edi, [esi+4]
push edi ; lpCriticalSection
call ds:EnterCriticalSection
mov eax, [esi]
test eax, eax
jz short loc_408A0B
push eax
call ds:GdiplusShutdown
loc_408A0B: ; lpCriticalSection
push edi
mov dword ptr [esi], 0
call ds:LeaveCriticalSection
pop edi
pop esi
retn
sub_4089F0 endp
jmp <trampoline>
trampoline:
jmp <hooked>
push esi
jmp sub_4089F0+5
hooked:
<body of hooked>
If <hooked> decides to call the original unhooked function:
call trampoline+5

```
void Attached();
void APIENTRY h_glBegin(GLenum mode);


void Attached()
{
    hOpenGL = GetModuleHandle("opengl32.dll");

    if (hOpenGL == INVALID_HANDLE_VALUE) return;
    o_glBegin = (glBegin_t)DetourFunction((LPBYTE)GetProcAddress(hOpenGL, "glBegin"), (LPBYTE)&h_glBegin);
}

void APIENTRY h_glBegin(GLenum mode)
{
    float  curcol[4];
    glGetFloatv(GL_CURRENT_COLOR, curcol);
    glDisable(GL_DEPTH_TEST);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glColor4f(curcol[0], curcol[1], curcol[2], 0.5f);

    (*o_glBegin)(mode);
}

BOOL WINAPI DllMain(HINSTANCE hDllInstance, DWORD nReason, LPVOID Reserved)
{
    switch (nReason)
    {
    case DLL_PROCESS_ATTACH:
        DisableThreadLibraryCalls(hDllInstance);
        hThisDll = hDllInstance;
        hThisThread = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)Attached, 0, 0, NULL);
        return (hThisThread != INVALID_HANDLE_VALUE);
        break;
    }

    return 0;
}

struct Cheat_t
```

VTable/VMT Hooking



- Remember those vtables from earlier? The vfptr is the first member variable of a class
- We could create a copy of the vtable and modify a function pointer from its original address to the address of our hooked function
- Then all we'll need to do is change the vfptr to our copy
- We can now hook any method that has been declared as virtual
- This method is incredibly popular with game overlays / screen recorders
- One goal is to hook IDXGISwapChain::Present. Our hooked function will be called whenever the screen is drawn by DirectX. i.e from there we can draw what we wish on the game screen, even if fullscreen

VEH / PAGE_GUARD Hooking

- Vectored Exception Handling can be used as a hooking mechanism by triggering exceptions.
- We can set the PAGE_GUARD bit on a memory page, whenever the CPU tries to access any memory in that page it triggers the STATUS_GUARD_PAGE_VIOLATION exception, and removes the PAGE_GUARD bit
- Think of it like a bear trap
- We can register an VEH exception handler and catch that exception
- By checking the exception context, we can check if the instruction pointer is pointed at the address of a function we want to hook (i.e the CPU tried to read the first instruction of the function). If so, enable single stepping or not, set PAGE_GUARD enabled again.
- If it is the address, we can set the instruction pointer to our detoured function

IAT (Import Address Table) Hooking

- All that needs to be done is to change the pointer to the function on the IAT
- Save the original so you can call it if needed

Conclusion & trivia

- ReactOS (they’re going to be at FOSDEM!)
- Wine
- Phoenix Technologies / clean-room reverse engineering
- Movfuscator

Questions