I'VE INJECTED A DLL - YOU WON'T BELIEVE WHAT HAPPENED NEXT!

BY @CAPTNBANANA
WHO R U MAN

- I do red teaming / pentesting
- Interested in reversing & exploit development
- And: Game hacking
- https://bananamafia.dev/
MOTIVATION
MOTIVATION: MONEY

- Win at Tournament: $$$
- Cheat: Easier $$$ ?
- Cheat subscription
  - 7 Days: 7€
  - 30 Days: 30€
  - 90 Days: 28€
PAID CHEATS

- "Humanized Bot"
- "We are undetected, we swear!"
- "If you attach a debugger we will ban you from the cheat service"
- "We don't log, trust us!"
CHEAT TYPES

- Wallhack
- Aimbot
- Game Specific:
  - No Flash
  - Anti Grip
  - See invisible players
  - Crosshair hack
HACK TYPES

- Internal
- External
- (Instrumented)
TOOLING

- Visual Studio: C++
- Debugger, e.g. x64dbg
- IDA/Ghidra/Radare2/Cutter/...
- Cheat Engine
ABOUT CHEAT ENGINE

• It's great
• Inspect and analyze process memory
• Disassembler
• Scripting engine
• Windows / Linux
  ▪ ceserver + GUI (wine)
HANDY CHEAT ENGINE FEATURES

- Scan for known values
- "What writes/reads" this address
- Freeze values
import time

text = "YOLOCON"

while True:
    print(text)
    time.sleep(1)
INTERNAL HACK ON WINDOWS

- For Jedi Academy and Counter Strike: GO
- Plan:
  - Build DLL loader
  - Build actual DLL
  - Inject DLL
  - Profit
HANDLE procHandle = OpenProcess(
    PROCESS_ALL_ACCESS,
    FALSE,
    PID);

LPVOID loadFunctionAddress = (LPVOID)GetProcAddress(
    GetModuleHandle("kernel32.dll"),
    "LoadLibraryA");

LPVOID allocatedMem = LPVOID(VirtualAllocEx(
    procHandle, 
    nullptr,
    MAX_PATH,
    MEM_RESERVE | MEM_COMMIT,
    offset));
BOOL APIENTRY DllMain (HMODULE hModule, DWORD ul_reason_for_call) {
    switch (ul_reason_for_call) {
    case DLL_PROCESS_ATTACH:
        MessageBox(0, "Cool, Works!", "1337 DLL", 0);
        break;
    }
    return TRUE;
}
GAME ENGINES

- From now: Game and engine specific
- What to hook and what to manipulate depends!
- Jedi Academy: idTech3 (Quake3) Engine
- CS:GO: Source engine
JEDI ACADEMY
WALLHACK, AIMBOT, ANTI GRIP
int CL_CgameSystemCalls(int *args) {
    switch(args[0]) {
        [...] 
        case CG_R_ADDREFENTITYTOSCENE:
            re.AddRefEntityToScene(VMA(1));
            return 0;
        [...] 
    }
}
CL_CgameSystemCalls()

- Implemented in cgamex86.dll
- Called by main executable (jamp.exe)
- Needs to be hooked
CALL INTO DLL: PROGRAM FLOW

- `jamp.exe` loads `cgamex86.dll`
- `jamp.exe` calls `GetProcAddress()` for desired function
- `GetProcAddress()` returns address of function inside of DLL
- `jamp.exe` executes function@Address
Hook `GetProcAddress()` and manipulate returned function address
- -> Execute own code instead
- -> Call original function in the end
CGAMEX86.DLL EXPORTS: DLLENTRY()

- Receives function pointer as parameter
- Hooked to manipulate existing code (e.g. for Wallhack)
- Events: Entity added, entity moves, game data received from server

```c
Q_EXPORT void dllEntry(intptr_t (QDECL *syscallptr)(intptr_t Q_syscall = syscallptr;
   TranslateSyscalls();
}
```
PLAN (2)

- Hook GetProcAddress()
- => hook dllEntry()
- Intercept calls with command CG_R_ADDREFENTITYTOSCENE
- Manipulate entity parameter
- Done!
HOOK SETUP: 

GetProcAddress() 

Redirect into own dllEntry()

Mhook_SetHook(
    (PVOID*)&originalGetProcAddress,
    hookGetProcAddress
);

[...]

FARPROC WINAPI hookGetProcAddress(HMODULE hModule, LPCSTR lpProcName)
{
    [...]
    if (isSubstr(lpProcName, "dllEntry")) {
        return (PROC)hookDLEntry;
    }
}

return (FARPROC)originalGetProcAddress(hModule, lpProcName)
int syscall_hook(int cmd, ...) {
    [...] 
    case CG_R_ADDREFENTITYTOSCENE: {
        // get the passed parameter (an entity)
        refEntity_t *ref = (refEntity_t *)arg[0];

        // HAX!!1!
        ref->renderfx |= RF_DEPTHHACK;

        break;
    }

    [...] 
    // call the original
AIMBOT

- Lock view at enemy
- Designated Aim key
- Requirements:
  - Engine Structures: Done via cgame hooks
  - Get enemy entity via crosshair
  - Calculate correct angle: World to Screen
  - Set angle programmatically
int crosshairClientNum = client_game->crosshairClientNum;
auto ent = entFromClientNum(crosshairClientNum);

centity_t* entFromClientNum(int clientNum) {
    [...]  
    for (int i = 0; i < MAX_GENTITIES; i++) {
        centity_t* cur = pEntities[i];
        if (!cur) { continue; }
        if (cur->playerState->clientNum == clientNum) {
            res = cur;
            break;
        }
    }
}
Matrix Vector Product and Matrix Matrix Product
Are specific formulas used for multiplying matrices & vectors.

\[
\begin{align*}
\text{screen.x} &= \left( \frac{\text{windowWidth}}{2} \right) \times NDC.x + (NDC.x + \frac{\text{windowWidth}}{2}) \\
\text{screen.y} &= -\left( \frac{\text{windowHeight}}{2} \right) \times NDC.y + (NDC.y + \frac{\text{windowHeight}}{2})
\end{align*}
\]

3D World Coords

- Multiply these together to get ModelViewProjection Matrix

- Perspective Division

- Divide by w

- Viewport Transform

- 2D SCREEN COORDS

- DING DING

- 3dCoords X ModelViewProjectionMatrix = ClipCoords.

- ClipCoords / w = NDC

Picture and Code Source: GuidedHacking
bool w2s(float fovx, float fovy, float windowWidth, float windowHeight)
{
    v3_t transform;
    float xc = 0, yc = 0;
    float px = 0, py = 0;
    float z = 0;

    px = tan(fovx * M_PI / 360.0);
    py = tan(fovy * M_PI / 360.0);

    transform = this->sub(origin);  //this = destination

    xc = windowWidth / 2.0;
    yc = windowHeight / 2.0;
void moveMouse(v3_t SCREEN)
{
    [...] 

    INPUT Input = {0};
    Input.type = INPUT_MOUSE;
    Input.mi.dwFlags = MOUSEEVENTF_MOVE;
    Input.mi.dx = SCREEN.x - client_game->refdef.width / 2;
    Input.mi.dy = SCREEN.y - client_game->refdef.height / 2;

    SendInput(1, &Input, sizeof(INPUT));
}
NO GRIP
THE PLAN

- Get enemy entity
- Automatically aim at enemy (Use Aimbot)
- Execute force throw -> Cancel grip
// If we are currently being gripped
if (ps && (ps->fd.forceGripBeingGripped || ps->fd.forceGripCri
auto ent = entFromClientNum(ps->persistant[PERS_ATTACKER]))
  // focus the current target -> use aimbot
  focusEnt(ent);
  syscall_hook(CG_SENDCONSOLECOMMAND, "force_throw;"));
}
HOW IT WORKS

// modify local player state
ps.fd.forcePowersActive |= (1 << FP_SEE);

- Works even though force is disabled on server
ALLOWING BLOCKED SETTINGS

```c
syscall_hook(CG_CVAR_SET, "sv_cheats", "1");
```

- Makes the game think that cheat settings are OK
- Enable e.g. `r_fullbright`
- `->` No shadows on the map
CS:GO
CROSSHAIR HACK
DIRECT3D EndScene()

- Queues scene for output (~ a frame)
- Executed after scene creation is completed
- Ideal for hooking
HOOKING EndScene()

- Create DLL with endSceneHook() function
- Important: Accept same parameters as original
- Inject DLL that hooks the function
- Use same loader as before
HOOK FUNCTION: PROTOTYPE

```c
void APIENTRY endSceneHook(LPDIRECT3DDEVICE9 p_pDevice);
```

- It's parameterless
- But: Implicit `this` parameter
REQUIREMENTS

- Get address of EndScene()
- Function that performs the actual hooking
DIRECT3D PSEUDO DEVICE TECHNIQUE

- Game restart, memory mapping, library versions, etc.:
  - No idea where endScene() actually is
  - Need reliable way to find it
  - Possible but not universal
- Here comes the Pseudo Device Technique
- Credits: GuidedHacking
DIRECT3D PSEUDO DEVICE TECHNIQUE

- Create own D3D device
- It contains a vTable with an entry to EndScene()
- Copy this function address
- Throw the device away
- Hook the function at the retrieved address
typedef struct IDirect3DDDevice9ExVtbl
{
    [... 41 Elements ...]
    HRESULT (WINAPI *EndScene)(IDirect3DDDevice9Ex *This);
    [...]
}
bool d3dHelper::getD3D9Device() {
    IDirect3D9* d3dSys = Direct3DCreate9(D3D_SDK_VERSION);
    IDirect3DDevice9* dummyDev = NULL;

    // Options to create dummy device
    D3DPRESENT_PARAMETERS d3dpp = {};
    d3dpp.Windowed = false;
    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
    d3dpp.hDeviceWindow = hwnd;

    HRESULT dummyDeviceCreated = d3dSys->CreateDevice(D3DADAPTOR_DEFAULT, D3DDEVTYPE_HAL, nullptr, 0, &d3dpp, &dummyDev);

    // Copy memory to our own data structure
THAT WAS EASY

cchar* ogEndSceneAddress = d3dHelper.d3d9DeviceTable[42];
WHAT'S MISSING?

- We now know what to hook
- We know how `endSceneHook()` has to be implemented
- How to actually hook?
TRAMPOLINE HOOKS
TRAMPOLINE HOOKS: OVERVIEW
TRAMPOLINE HOOKS: CODE

// adapted from https://guidedhacking.com/threads/simple-x86-c
const char* REL_JMP = "\xE9";
const unsigned int SIZE_OF_REL_JMP = 5;

void* WINAPI hookFn(char* hookedFn, char* hookFn, int copyByte)

    // Backup original
    ReadProcessMemory(GetCurrentProcess(), hookedFn, backupByte);

    // Trampoline setup
    char* trampoline = (char*)VirtualAlloc(0, copyBytesSize + PAGE_EXECUTE_READWRITE);
    memcpy(trampoline, hookedFn, copyBytesSize);


TRAMPOLINE HOOKS: ASM VIEW

; Original EndScene()
0x5F8F46A0 6A 14 push 14 ; Prologue
0x5F8F46A2 B8 2E01915F mov eax,d3d9.5F91012E ; Prologue
0x5F8F46A7 E8 3E8B0100 call d3d9.__EH_prolog3_catch ; Actual call

; [more code]

; Hooked EndScene():
0x5F8F46A0 E9 XXXXXXXX jmp dll.EndSceneHook ; Jump to Own Code
0x5F8F46A5 91 ??? ; Trash, never executed
0x5F8F46A6 5F ??? ; Trash, never executed
0x5F8F46A7 E8 3E8B0100 call d3d9.__EH_prolog3_catch ; Actual call

; [more code]

; The Trampoline()
EndSceneHook() IMPLEMENTATION

// the returned trampoline
extern endSceneFunc trampEndScene;

void APIENTRY d3dHelper::endSceneHook(LPDIRECT3DDEVICE9 p_pDev
    [...]  
    // Do own stuff  
    drawRectangle(25, 25, 100, 100, D3DCOLOR_ARGB(255, 255, 25
    // Call original function using the trampoline  
    trampEndScene(d3dDevice);  
}
COMPILING

- Add DirectX SDK to linker libraries
- Compile DLL for x86
DEBUGGING WITH SYMBOLS

<table>
<thead>
<tr>
<th>Base</th>
<th>Module</th>
<th>Party</th>
<th>Address</th>
<th>Type</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>79640000</td>
<td>acgeneral.dll</td>
<td>System</td>
<td>5F8CB544</td>
<td>Symbol</td>
<td>??_C0_0BB@KBBLCIKD@EndScene?sf</td>
</tr>
<tr>
<td>77020000</td>
<td>advapi32.dll</td>
<td>System</td>
<td>5F8CB484</td>
<td>Symbol</td>
<td>??_C0_0BD@OIBIHEEB@BeginScene?sf</td>
</tr>
<tr>
<td>70CE0000</td>
<td>amsi.dll</td>
<td>System</td>
<td>5F8CF588</td>
<td>Symbol</td>
<td>??_C0_OCFOCJAIAKBD@Driver?sf</td>
</tr>
<tr>
<td>70B20000</td>
<td>antimalware_provider.dll</td>
<td>System</td>
<td>5F8CB4C8</td>
<td>Symbol</td>
<td>??_C0_OCJO6A6C38E69@EndScene?sf</td>
</tr>
<tr>
<td>68BB0000</td>
<td>apphelp.dll</td>
<td>System</td>
<td>5F8CC2C0</td>
<td>Symbol</td>
<td>??_C0_OC30HAMGEN@EndScene?sf</td>
</tr>
<tr>
<td>62DF0000</td>
<td>audioses.dll</td>
<td>System</td>
<td>5F8CC284</td>
<td>Symbol</td>
<td>??_C0_OCK@AOECIGM@Need?sto</td>
</tr>
<tr>
<td>62740000</td>
<td>avifil32.dll</td>
<td>System</td>
<td>5F8CC434</td>
<td>Symbol</td>
<td>??_C0_ODB@OBPNBBBE@BeginScene</td>
</tr>
<tr>
<td>62FC0000</td>
<td>avrt.dll</td>
<td>System</td>
<td>5F8CB4A0</td>
<td>Symbol</td>
<td>BeginScene@CD3DBase@UAGJXZ</td>
</tr>
<tr>
<td>770F0000</td>
<td>bcrypt.dll</td>
<td>System</td>
<td>5F8F45A0</td>
<td>Symbol</td>
<td>EndScene@CD3DBase@UAGJXZ</td>
</tr>
<tr>
<td>77850000</td>
<td>bcyptprimitives.dll</td>
<td>System</td>
<td>5F8F46A0</td>
<td>Symbol</td>
<td>??_C0_0BB@KBBLCIKD@EndScene</td>
</tr>
<tr>
<td>78770000</td>
<td>calro.dll</td>
<td>System</td>
<td>5F8F4684</td>
<td>Symbol</td>
<td>??_C0_0BB@KBBLCIKD@EndScene</td>
</tr>
<tr>
<td>777A0000</td>
<td>cfgmgr32.dll</td>
<td>System</td>
<td>5F99E0A3</td>
<td>Symbol</td>
<td>??_C0_0BB@KBBLCIKD@EndScene</td>
</tr>
<tr>
<td>76AC0000</td>
<td>clbcatg.dll</td>
<td>System</td>
<td>5F99E10F</td>
<td>Symbol</td>
<td>??_C0_0BB@KBBLCIKD@EndScene</td>
</tr>
<tr>
<td>34ED0000</td>
<td>clclient_panorama.dll</td>
<td>System</td>
<td>5F9A4220</td>
<td>Symbol</td>
<td>??_C0_0BB@KBBLCIKD@EndScene</td>
</tr>
<tr>
<td>75DF0000</td>
<td>combase.dll</td>
<td>System</td>
<td>5F9D8C0D</td>
<td>Symbol</td>
<td>??_C0_0BB@KBBLCIKD@EndScene</td>
</tr>
<tr>
<td>6FDA0000</td>
<td>comctl32.dll</td>
<td>System</td>
<td>5F8F93C0</td>
<td>Symbol</td>
<td>??_C0_0BB@KBBLCIKD@EndScene</td>
</tr>
<tr>
<td>66EE0000</td>
<td>coremessaging.dll</td>
<td>System</td>
<td>5F982EA5</td>
<td>Symbol</td>
<td>??_C0_0BB@KBBLCIKD@EndScene</td>
</tr>
<tr>
<td>66C80000</td>
<td>coreuicomponents.dll</td>
<td>System</td>
<td>5F8F45A0</td>
<td>Symbol</td>
<td>??_C0_0BB@KBBLCIKD@EndScene</td>
</tr>
<tr>
<td>67000000</td>
<td>crashhandler.dll</td>
<td>System</td>
<td>5F8F46A0</td>
<td>Symbol</td>
<td>??_C0_0BB@KBBLCIKD@EndScene</td>
</tr>
<tr>
<td>76CC0000</td>
<td>crypt32.dll</td>
<td>System</td>
<td>5F8F46A0</td>
<td>Symbol</td>
<td>??_C0_0BB@KBBLCIKD@EndScene</td>
</tr>
<tr>
<td>75560000</td>
<td>cryptbase.dll</td>
<td>System</td>
<td>5F8F46A0</td>
<td>Symbol</td>
<td>??_C0_0BB@KBBLCIKD@EndScene</td>
</tr>
<tr>
<td>62A20000</td>
<td>cryptnet.dll</td>
<td>System</td>
<td>5F8F46A0</td>
<td>Symbol</td>
<td>??_C0_0BB@KBBLCIKD@EndScene</td>
</tr>
<tr>
<td>77290000</td>
<td>cryptsp.dll</td>
<td>System</td>
<td>5F8F46A0</td>
<td>Symbol</td>
<td>??_C0_0BB@KBBLCIKD@EndScene</td>
</tr>
<tr>
<td>70F80000</td>
<td>cspoo.exe</td>
<td>User</td>
<td>5F8F4684</td>
<td>Symbol</td>
<td>??_C0_0BB@KBBLCIKD@EndScene</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Ordina</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>5F890000</td>
<td>Symbol</td>
<td>EndScene@CD3DBase::BeginScene</td>
<td></td>
</tr>
<tr>
<td>5F890000</td>
<td>Symbol</td>
<td>CD3DBase::EndScene</td>
<td></td>
</tr>
</tbody>
</table>
CS:GO
EXTERNAL CHEAT
HOW IT WORKS

- Read and analyze game memory
- Manipulate memory accordingly
- Works without code injection
WHAT CAN BE IMPLEMENTED?

- Aimbot
- No Flash
- Auto-Bunnyhop
- Probably more
MEMORY ANALYSIS

- Attach CheatEngine
- Scan for known values
- Reverse structures
MEMORY ANALYSIS

- Use leaked game source code
- Refer to existing cheat source code
# Static Pointers

**client_panorama_client.so**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12</td>
<td>0x34</td>
<td></td>
</tr>
<tr>
<td>0x32</td>
<td>0x13</td>
<td></td>
</tr>
<tr>
<td>0x78</td>
<td>Static Pointer</td>
<td></td>
</tr>
</tbody>
</table>

**Game Memory**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x23</td>
<td></td>
<td>start of player_base</td>
</tr>
<tr>
<td>0x33</td>
<td>Start of something</td>
<td></td>
</tr>
<tr>
<td>0x44</td>
<td>health</td>
<td></td>
</tr>
<tr>
<td>[...]</td>
<td>location</td>
<td></td>
</tr>
</tbody>
</table>

**Offset:** 0x214AEFO

**Offset:** 0xC
hazedumper

🚀 Up to date offset and dumper-config for Counter-Strike: Global Offensive. For more informations visit the release page on UnKnoWnCheaTs.

Local Player

Since in the past the signature for the LocalPlayer was broken a few times and/or pointed to something wrong, I want to offer you an alternative. All required offsets are already in the repo, why 99% of people don't use them is questionable - or they are just too incompetent and complain that nothing works, but they can't find a single signature themselves. However, pretty much every hack uses the entity list ( `dwEntityList` ) and also ClientState ( `dwClientState` ). All you need is a third offset which is located in ClientState, called `dwClientState_GetLocalPlayer`.

```cpp
const auto client_state = read_memory<std::uint32_t>( engine_image->base + hazedumper::signatures::dwClientState );
if( client_state ) {
    const auto local_player = get_client_entity( read_memory<std::int32_t>( client_state + hazedumper::signatures::dwClientState_GetLocalPlayer );
}
if( local_player ) {
    printf( "[+] Found local player: 0x%X, health: %d\n",
        local_player,
```
read_memory<
std::int32_t
>(
local_player
+
hazedumper::netvars::m_iHealth
);
UPDATES

- Game update -> Cheat update
- Different offsets, addresses, struct members
IMPLEMENTING CHEATS: NO FLASH

- Write 0 to member of LocalPlayer
- It's that easy
I NEED MORE INFORMATION!

Check out my talk from BSides Munich 2020!
GAME HACKING WITH FRIDAJA
HOW IT WORKS

- Inject JS code into black box process
- Hook, trace
Simpsons: Hit & Run API

This code is in a pre-pre-pre-alpha experimental state.

This is a library to automate the abandonware game Simpsons: Hit & Run with JavaScript. It uses frida to access internal state, and exposes JavaScript classes that can be used to query and control the game.

The intention is to allow easy scriptable access to state, in a similar way to how pysc2 enabled deepmind to learn how to play Starcraft II. Eventually I'd like to be able to automate finding glitches, crashes, strategies, routes and so on for speedrunning.

This is a solo hobby project, I'm a long way off from that point.

Want to see it in action? Here is an early demo video using the debugging console.
function kick(count = 1) {
    while (count-- > 0) {
        input.SimulateKeyPress(Mappable.id.Character.Attack);
        Thread.sleep(0.5);
    }
}

// Move in current direction.
step();

// Did we find the object?
if (myPos.distanceTo(object) <= 1) {
CHEAT DETECTION
VAC DETECTION

• "VAC is a Joke"
• Uses signatures (among other things)
• Detects specific kinds of hooks
• Solution: Hook mid function
• Don't use public code
• Manual Mapping, Polymorphism, Unhooking
• But: Kernel mode anti cheats exist
MANUAL MAPPING?

- A fancier DLL injector
- Bypass:
  - `LoadLibrary()` Hooks
  - `CreateToolhelp32Snapshot()`
  - `EnumModules()`
  - `NtQueryVirtualMemory()`
- DLL doesn't show up in loaded modules
- Also not in Process Environment Block (PEB)
MANUAL MAPPING

- Re-Implement LoadLibrary()
- -> Kernel doesn't know a DLL is loaded :(
MANUAL MAPPING: HOW IT WORKS

- Load raw DLL into own memory
- Map DLL sections in game process
- Inject and run loader shellcode in game process
  - Relocate
  - Fix imports
  - TLS callbacks
  - Call DllMain()
- Cleanup: Free memory
MANUAL MAPPING: RELOCATIONS

- Required if allocated space != ImageBase of DLL
- DLL includes relocation information
- For global variables, addresses for CALL instructions
- Relocate: Adjust addresses based on new base address
MANUAL MAPPING: IMPORTS

- Injected DLL may require additional functions of other DLLs
- Fixing imports:
  - Loading these DLLs
  - Setting pointers to imported functions in DLL header
MANUAL MAPPING: TLS CALLBACKS

- TLS = Thread Local Storage
- Executed before DLL entry point
- TLS Table in DLL header
- Executing TLS Callbacks -> Initialize per-thread data
@CaptxnBanana
REFERENCES

- My Blog Posts
- Source Code: CS:GO EndScene Hook
- d3d9.h Source Code
- Simpsons Hit and Run Frida API
- GuidedHacking: World to Screen Functions
- GuidedHacking: Manual Mapping
- Rohitab: Manual Mapping
- TLS Section
- Slides for my Talk @ BSides Munich