



General

FA2026 • 2026-10-22

Game Hacking I

Tyler Mercado

Announcements

- We are playing **BuckeyeCTF 2025** hosted by OSU!
 - Tomorrow (**11/7**) at **7:00pm** (room TBD, likely Siebel 2406)
 - Unlike most CTFs, Buckeye offers prizes to the top 3 undergraduate teams
 - No graduate students are allowed to play
 - Try hard and win that prize!



ctf.sigpwny.com

sigpwny{sk111_155u3_p47ch3d}

Trampoline Hook

Our hook function

asm code
top of function
jmp
continue
execution

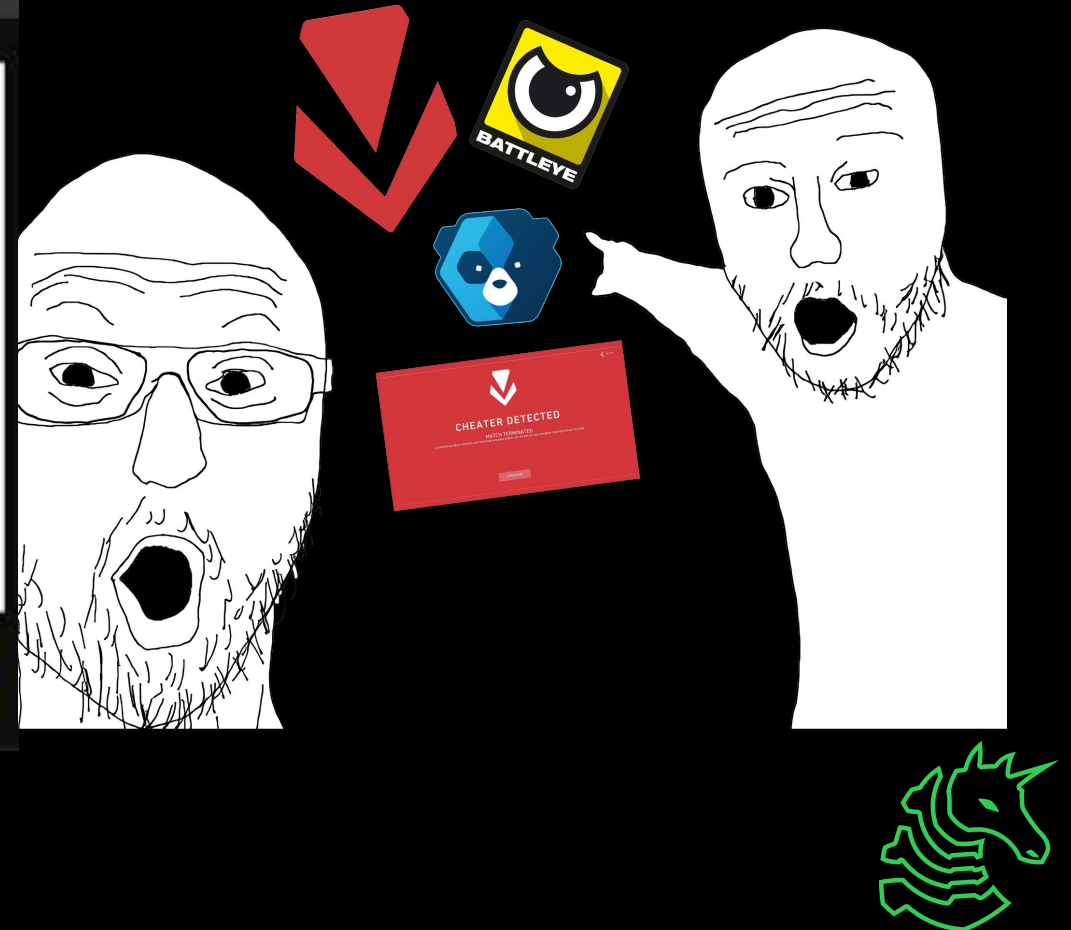
our code
cout << "Hooked"
call Gateway

Gateway

execute stolen bytes
jmp to original function
right after our jmp



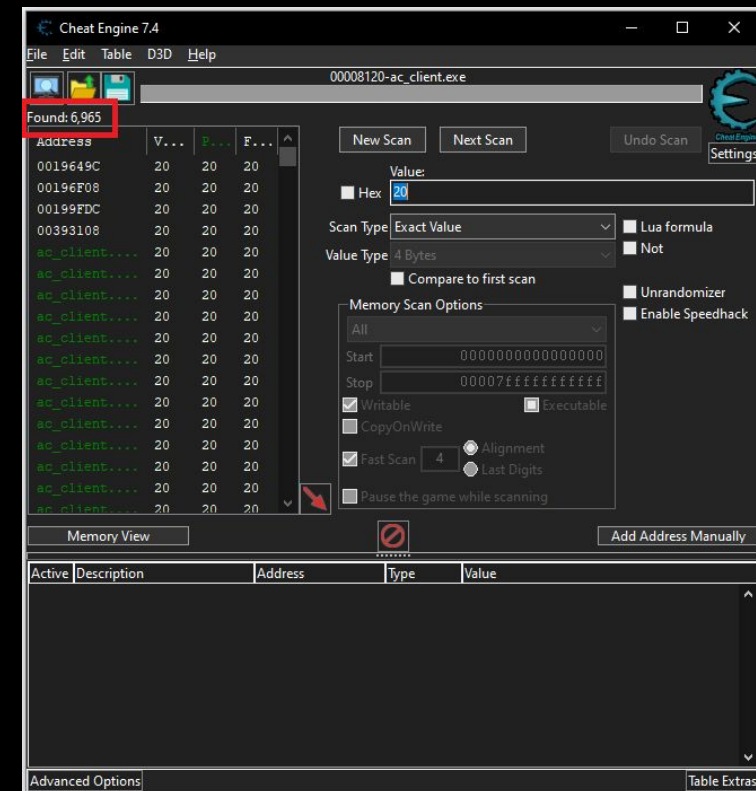
TRAMPOLINE HOOK



What is Game Hacking?

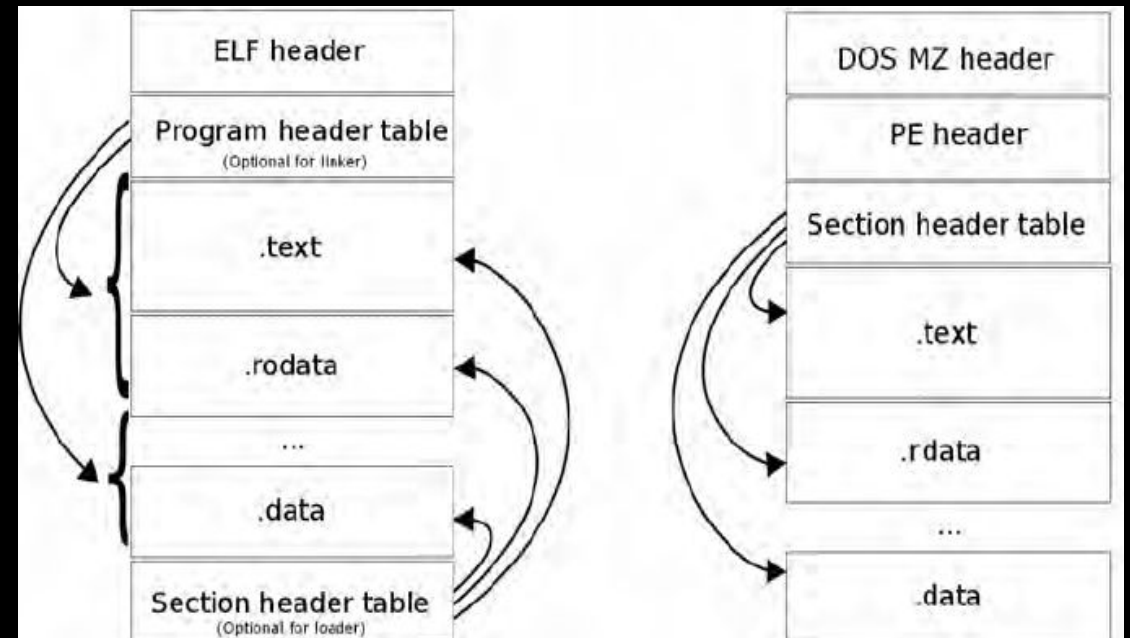
Game hacking is the practice of reverse engineering and modifying video game software to manipulate game behavior.

- Memory manipulation
- Code injection
- Reverse engineering
- Anti-cheat evasion



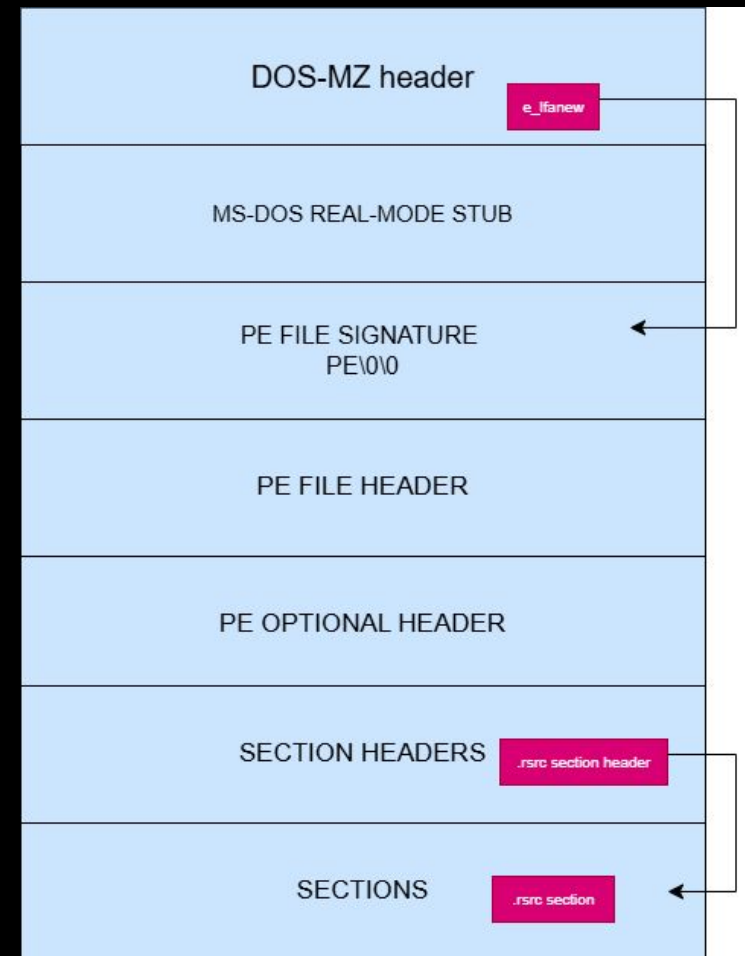
Windows PE vs. Linux ELF

- **.rdata** stores read-only constants (.rodata in ELF)
- **.text** stores executable code
- **.data** stores static and global variables



DOS MZ Header

- Legacy header from MS-DOS era
- Contains pointer to PE header location
- Includes MS-DOS stub program
- Required for backwards compatibility



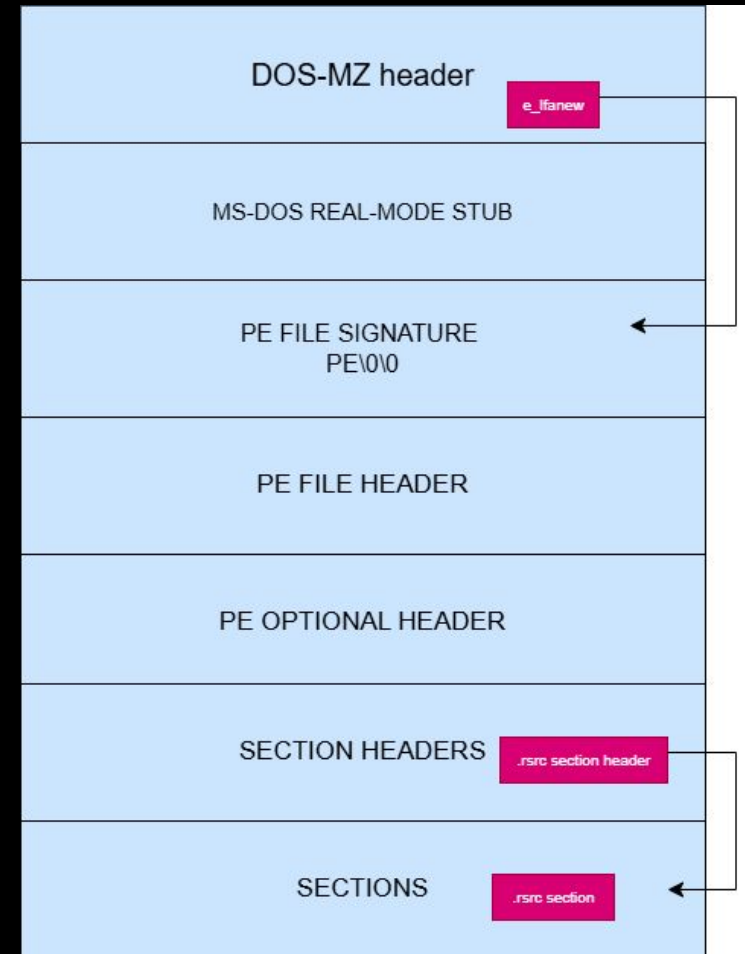
PE Header

- Defines architectures (x86, 64), entry point, and image base addresses
- Specifies number and locations of sections (.text, .data, .rdata)
- Includes import/export table for DLL functions

Why do we care?

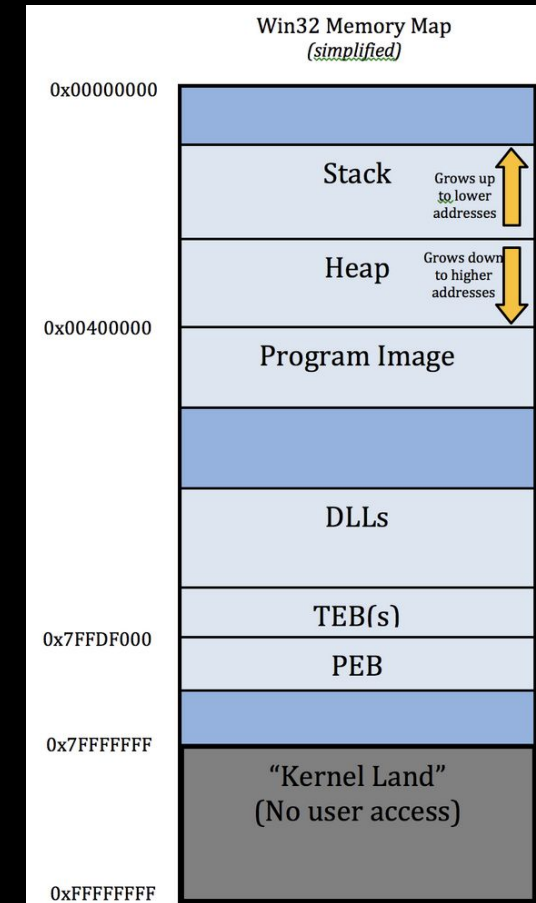
It reveals memory layout, and imported functions to hook.

Base address matters because of ASLR!



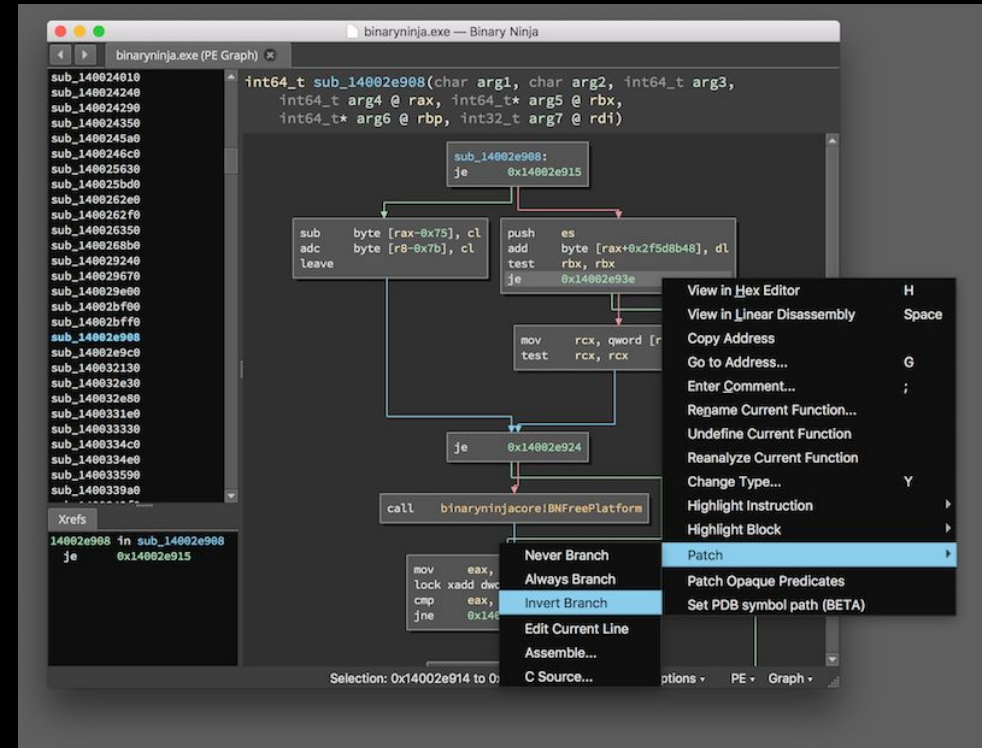
Virtual Memory in Windows

- Each process has isolated 2GB user space (32-bit) or 128TB (64-bit)
- Stack and Heap differ from Linux layout
- Program Image (loaded executable)
- DLLs are shared libraries mapped into process space
- PEB/TEB contain process/thread metadata
- Kernel Land is OS memory, inaccessible from user mode.



Patching

- Directly modifying executable code/data in memory or on disk
- Memory patching changes bytes at runtime
- Disk patches are permanent modifications
- Common targets
 - NOP (0x90)
 - JE -> JNE, JMP
 - Constants (damage, health, ...)



External Cheats

- **ReadProcessMemory**
 - copies data from address range
- **WriteProcessMemory**
 - writes buffer data to address range
- **VirtualProtectEx**
 - sets memory permissions
- Generally easier to make with tools like Cheat Engine
- Harder to detect
- Less control



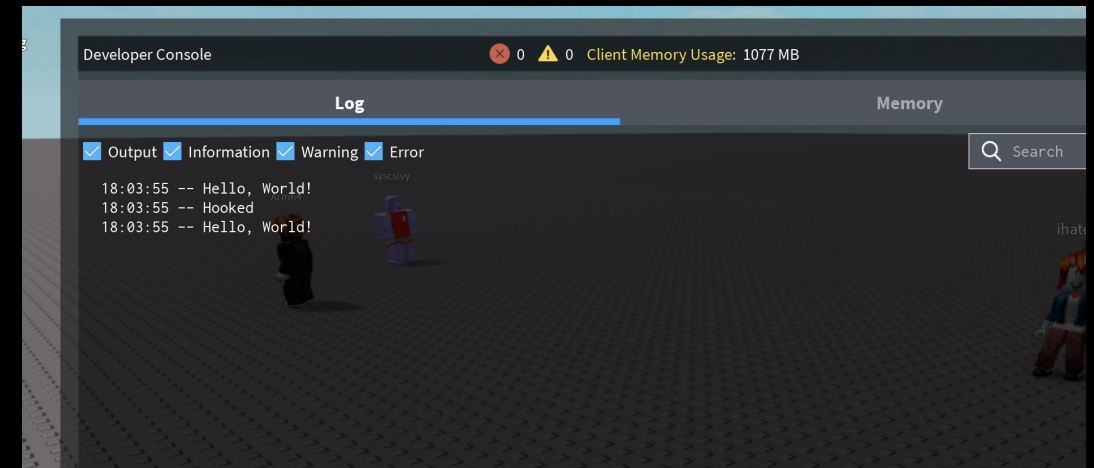
Internal Cheats

- Runs inside game processes
- Can call functions (spawn items, teleport, etc.)
 - `reinterpret_cast<void(*) (const char*)>(0xDEADBEEF)("Hello, World!");`
- Can hook functions
- Can access memory through pointer dereferencing
 - `*reinterpret_cast<std::uint8_t*>(0xDEADBEEF);`
- Requires more reversing
 - We need to find function addresses
- Easier to detect
 - Vulnerable to signature scans



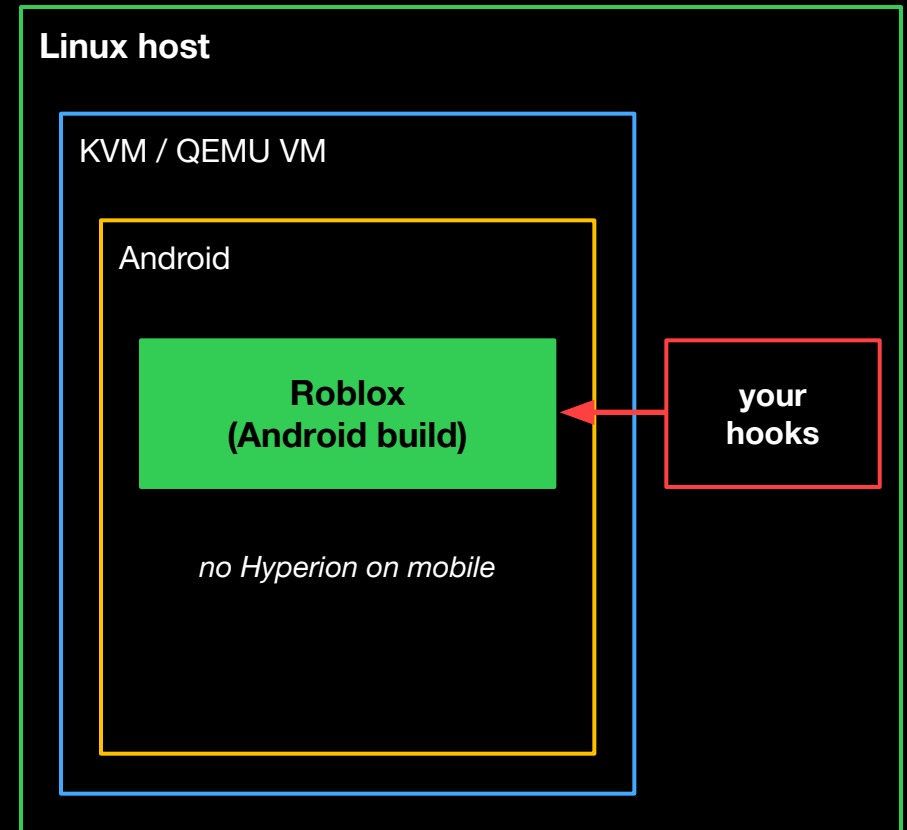
Example with Roblox

```
auto message_out = reinterpret_cast<std::int64_t*>(std::uint32_t, const char *, ...)>(  
    GetModuleHandle(nullptr) + 0x100d58e44  
);  
  
std::uintptr_t (*message_out_orig)(std::uint32_t, const char *, ...) = nullptr;  
  
message_out(0, "Hello, World!");  
  
// Apply hook  
DobbyHook(reinterpret_cast<void*>(message_out),  
    reinterpret_cast<void*>(message_out_hook),  
    reinterpret_cast<void**>(&message_out_orig));  
  
std::this_thread::sleep_for(std::chrono::milliseconds(1));  
  
message_out(0, "Hello, World!");  
  
std::this_thread::sleep_for(std::chrono::milliseconds(1));  
  
// Destroy hook  
DobbyDestroy(reinterpret_cast<void*>(message_out));  
  
std::this_thread::sleep_for(std::chrono::milliseconds(1));  
  
message_out(0, "Hello, World!");
```



Emulating Roblox Mobile

- Windows client: Hyperion (Byfron)
 - Packed, obfuscated, anti-tamper
- Modern technique: emulate Android
 - Mobile clients don't ship Hyperion
 - Run it in a Linux VM, hook from outside



Reversing Structs

- Analyze memory access patterns
 - Track how pointers are dereferenced, e.g. `*(data + 0xc)`, `*(data + 8)`
- Identify offsets and determine types
 - Each offset reveals a struct member location
- We can access/modify game data directly via typed structs instead of raw pointers

```
int64_t var_1d8;  
var_1d8 = (*(data_14009c378 + 0xc));  
sub_140040b60("300/100/50/Miss: %d / %d / %d / %d",  
    (*(data_14009c378 + 8)), (*(data_14009c378 + 4)), *data_14009c378,  
    var_1d8);
```

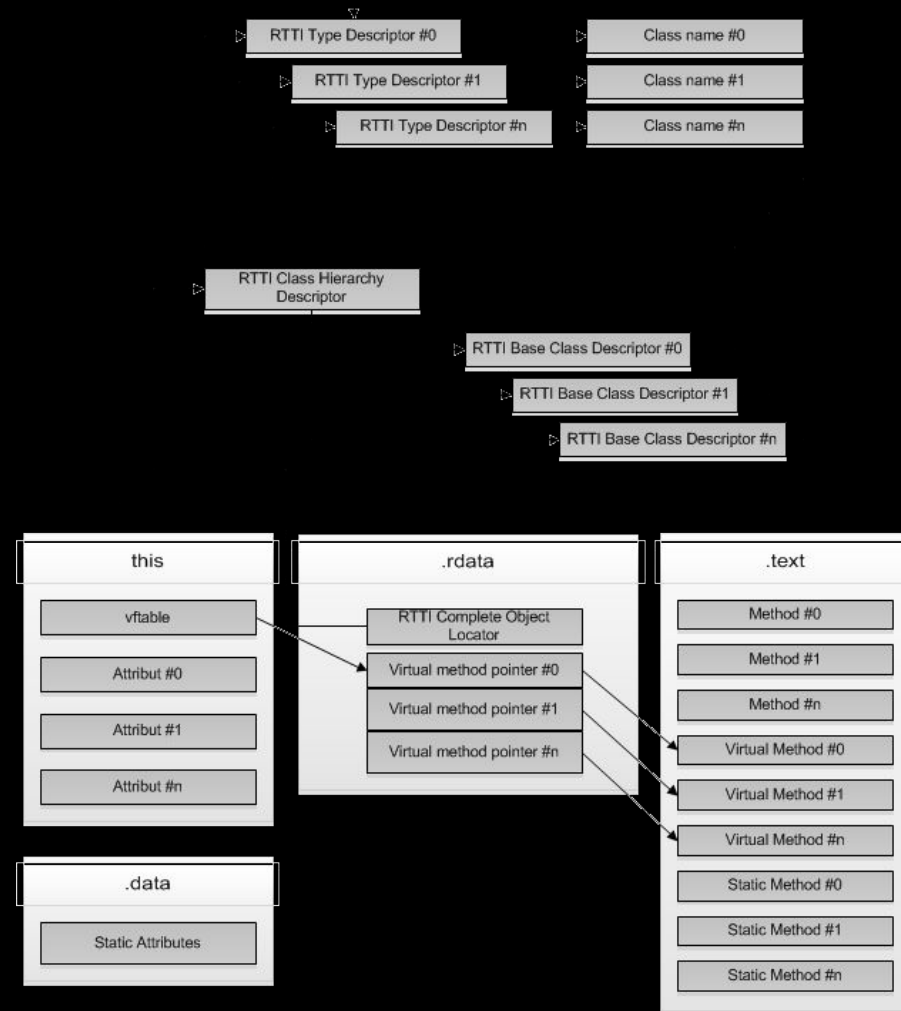
```
int64_t var_1d8;  
var_1d8 = data_14009c378.num_misses;  
sub_140040b60("300/100/50/Miss: %d / %d / %d / %d",  
    data_14009c378.num_300s, data_14009c378.num_100s,  
    data_14009c378.num_50s, var_1d8);
```

```
struct HitStruct __packed  
{  
00     int32_t num_50s;  
04     int32_t num_100s;  
08     int32_t num_300s;  
0c     int32_t num_misses;  
10 };
```



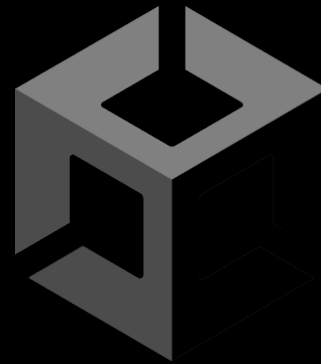
Reversing Virtual Tables

- MSVC adds RTTI metadata which disassemblers can parse
- RTTI reveals class names, inheritance hierarchy, and vtable structure
- Identify game objects (e.g. Player, Enemy, Weapon)
- Allows hooking virtual functions
- Find “Player” class -> locate vtable -> hook `TakeDamage()`



C#/Mono Game Cheats

- Managed code (IL bytecode, preserves type info)
- Decompilation with dotPeek recovers source-like C# code
- Direct IL patching or Mono.Cecil for assembly modification
- MonoInjector for runtime code execution
- Unity: Mono vs Il2Cpp, Unity Engine API access
- No signature scanning needed
 - use reflection to find methods



Advanced Reversing Techniques

- Unpacking binaries
- Devirtualization (lifting to LLVM IR)
- Binary Ninja lifts code to its own IL (BNIL)
- IDA Pro's Lumina pulls function names from a shared database
- BinDiff can import symbols from other binaries

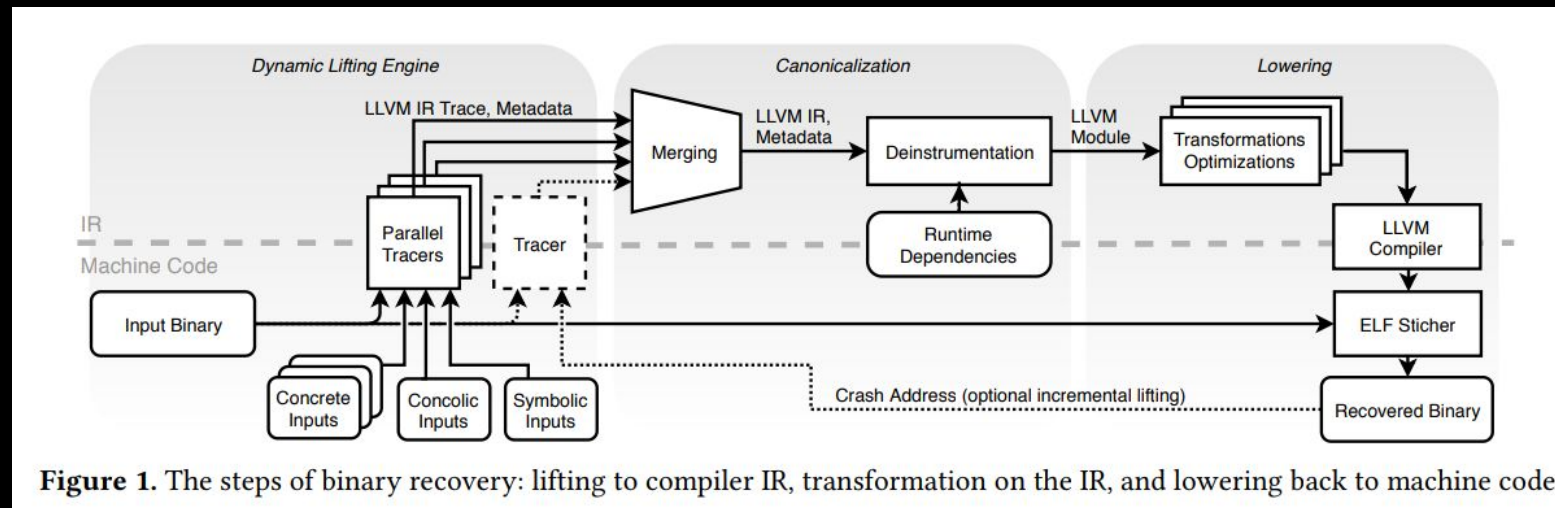


Figure 1. The steps of binary recovery: lifting to compiler IR, transformation on the IR, and lowering back to machine code.



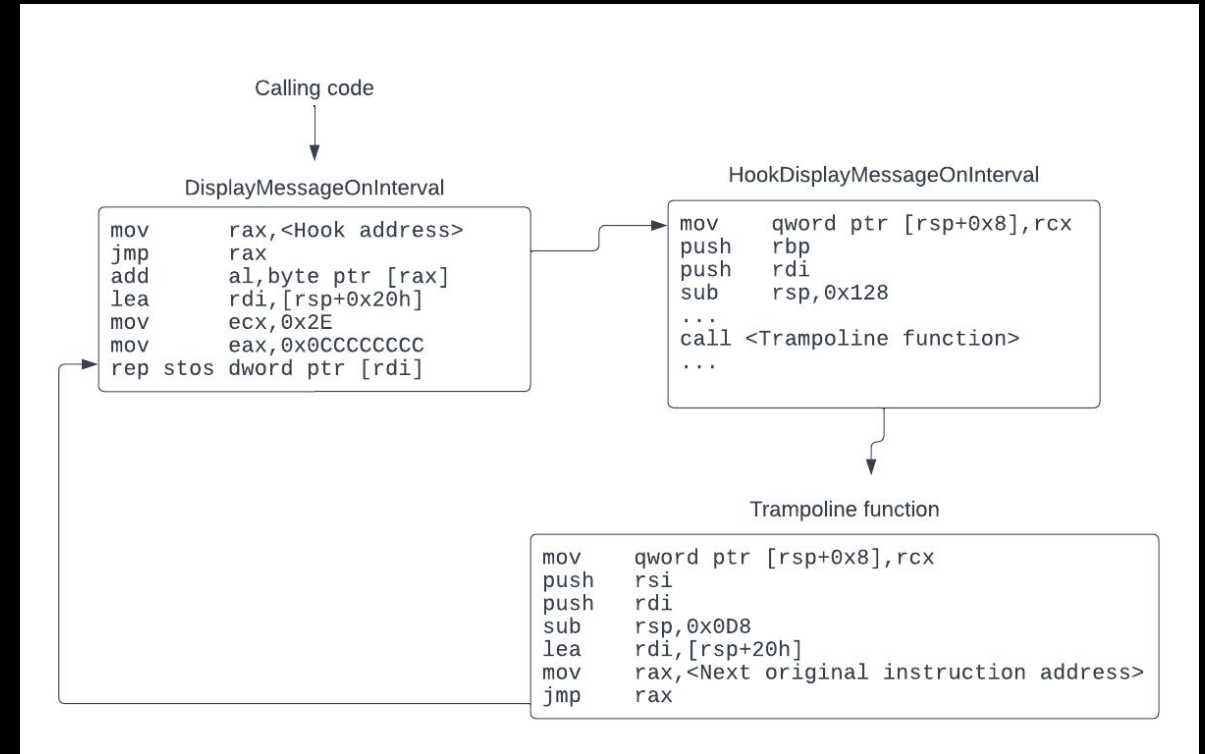
DLL Injection

- Traditional DLL Injection
 - Use **CreateRemoteThread** and **LoadLibrary**
 - Windows handles loading and calling **DllMain** automatically
- Manual Mapping
 - Manually allocate memory, copy DLL sections, fix imports/relocations
 - More stealthy
 - No entry in PEB
- **GetModuleHandleA(nullptr);**
 - Returns the base address of the current process's main module (with ASLR offset)
- **BOOL WINAPI DllMain(HINSTANCE module, DWORD reason, LPVOID);**
 - DLL entry point; we will run our code in here



Hooking Functions

- Traditionally requires writing inline assembly
1. Patch first few bytes of function to jmp to our code
 2. Process function arguments
 3. If we want, call the instructions we replaced and jmp back to function at the next instruction



Hooking Functions

- Modern libraries allow for JIT inline hooking.
- No inline assembly required
- Can be vulnerable to signature scans

stevemk14ebr/
PolyHook_2_0



C++20, x86/x64 Hooking Library v2.0

 39
Contributors

 8
Issues

 2k
Stars

 250
Forks





Example with OBS: Creating Overlays

```
#include "obs_hook.hpp"

namespace takoyaki::obs_hook {
bool copy_texture_hook::install() {
    return m_copy_texture_detour->hook();
}

bool copy_texture_hook::uninstall() {
    m_device = nullptr;
    m_device_context = nullptr;
    return m_copy_texture_detour->unhook();
}

bool copy_texture_hook::installed() {
    return m_copy_texture_detour->isHooked();
}

void copy_texture_hook::copy_texture_callback(void* gs_duplicator, ID3D11Texture2D* texture) {
    std::call_once(m_device_initialization_flag, [&]() {
        texture->GetDevice(&m_device);
        m_device->GetImmediateContext(&m_device_context);
    });

    D3D11_TEXTURE2D_DESC texture_description;
    texture->GetDesc(&texture_description);

    texture_description.CPUAccessFlags = D3D11_CPU_ACCESS_READ;
    texture_description.Usage = D3D11_USAGE_STAGING;
    texture_description.BindFlags = 0;
    texture_description.MiscFlags = 0;

    Microsoft::WRL::ComPtr<ID3D11Texture2D> staging_texture;
    if (m_device->CreateTexture2D(&texture_description, nullptr, &staging_texture) != S_OK) {
        PLH::FnCast(m_copy_texture_trampoline_address, reinterpret_cast<void*>(void*, ID3D11Texture2D*)>(m_copy_texture_address))(gs_duplicator, texture);
        return;
    }

    m_device_context->CopyResource(staging_texture.Get(), texture);

    D3D11_MAPPED_SUBRESOURCE mapped_subresource;
    if (m_device_context->Map(staging_texture.Get(), 0, D3D11_MAP_READ, 0, &mapped_subresource) != S_OK) {
        PLH::FnCast(m_copy_texture_trampoline_address, reinterpret_cast<void*>(void*, ID3D11Texture2D*)>(m_copy_texture_address))(gs_duplicator, texture);
        return;
    }

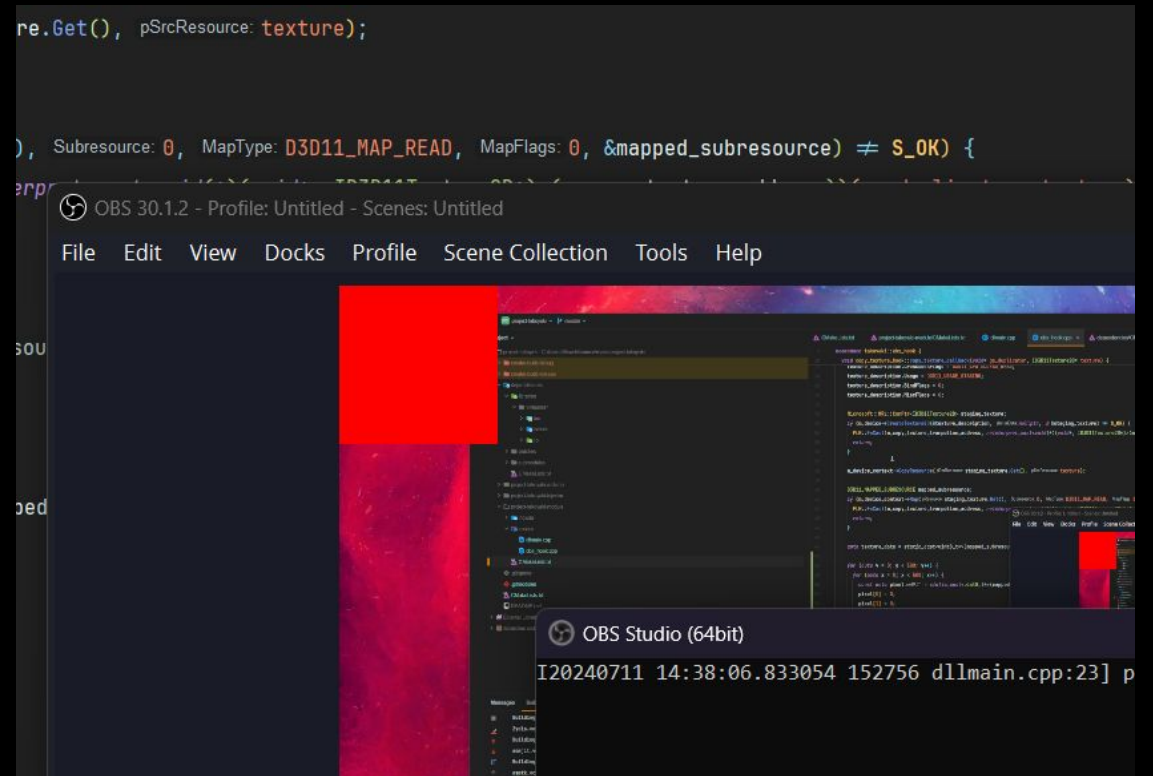
    auto texture_data = static_cast<uint8_t*>(mapped_subresource.pData);

    for (auto y = 0; y < 500; y++) {
        for (auto x = 0; x < 500; x++) {
            const auto pixel = static_cast<uint8_t*>(mapped_subresource.pData) + (y * mapped_subresource.RowPitch) + (x * 4);
            pixel[0] = 0;
            pixel[1] = 0;
            pixel[2] = 255;
            pixel[3] = 255;
        }
    }

    m_device_context->UpdateSubresource(texture, 0, nullptr, mapped_subresource.pData, mapped_subresource.RowPitch, 0);

    m_device_context->Unmap(staging_texture.Get(), 0);

    PLH::FnCast(m_copy_texture_trampoline_address, reinterpret_cast<void*>(void*, ID3D11Texture2D*)>(m_copy_texture_address))(gs_duplicator, texture);
}
}
```



Bypassing Return Address Checks

- Checks for valid function return addresses (when it is pushed to the stack)
- When we call a function that is protected by a return address checker, we need to either NOP/JMP the check branch or patch the function to return early

```
if ( retaddr - (_BYTE *)sub_401000 >= (unsigned int)sub_C02AE6 )
{
    result = sub_18C5000;
    if ( (unsigned int)(retaddr - (_BYTE *)sub_18C5000) >= 0x122D9A )
    {
        dword_16AAEE0 |= 0x2000000u;
        LODWORD(qword_16AAEB0) = qword_16AAEB0 | 0x1000000;
        result = (int (*)( ))HIDWORD(qword_16AAEB0);
        dword_16AE2AC = 0;
    }
}
```



Bypassing Integrity Checks

- Memory checksums
 - Anti-cheat calculates hash/CRC of code sections periodically
 - If hash mismatches (code patched), trigger detection
- We can bypass these checks by hooking them.
 - Use a debugger to look through threads and find the integrity checker
 - Calculate the expected/stored hash before code patches
 - Hook function to always receive valid hash
- What if there are integrity checkers that check each other?



Bypassing Signature Scans

- Scans memory for known cheat signatures
- Checks running processes file hashes against known cheats
- Trivially, hook the checks
- Alternatively, obfuscate and pack your code + shuffle structs and constants

```
std::vector<std::size_t> find_pattern(
    const std::uint8_t* data, std::size_t data_size, const std::vector<pattern_byte_t>& pattern
) {
    std::vector<std::size_t> matches;

    // Early exit for invalid inputs
    if (data == nullptr || pattern.empty() || data_size == 0 || data_size < pattern.size()) {
        return matches;
    }

    // Last position where pattern could possibly start
    const std::size_t end_pos = data_size - pattern.size();

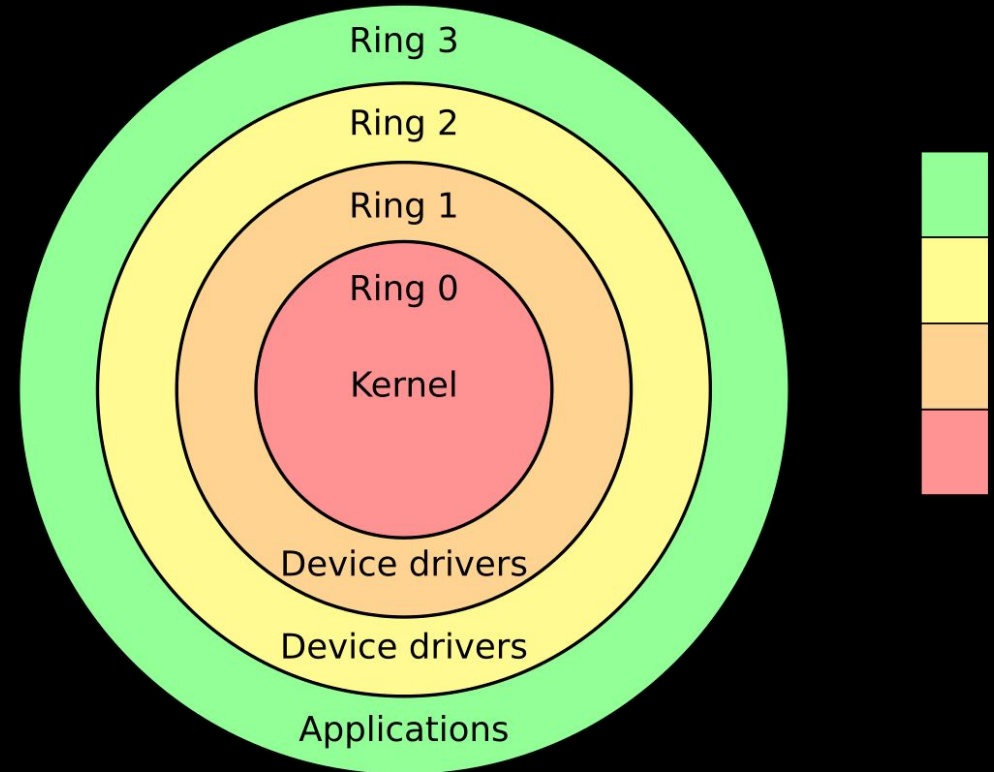
    // Scalar byte-by-byte matching with early exit on mismatch
    for (std::size_t i = 0; i <= end_pos; ++i) {
        bool match = true;
        // Check each byte in pattern
        for (std::size_t j = 0; j < pattern.size(); ++j) {
            if (!match_pattern_byte(data[i + j], pattern[j])) {
                match = false;
                break; // Early exit on first mismatch
            }
        }
        if (match) {
            matches.push_back(i);
        }
    }

    return matches;
}
```



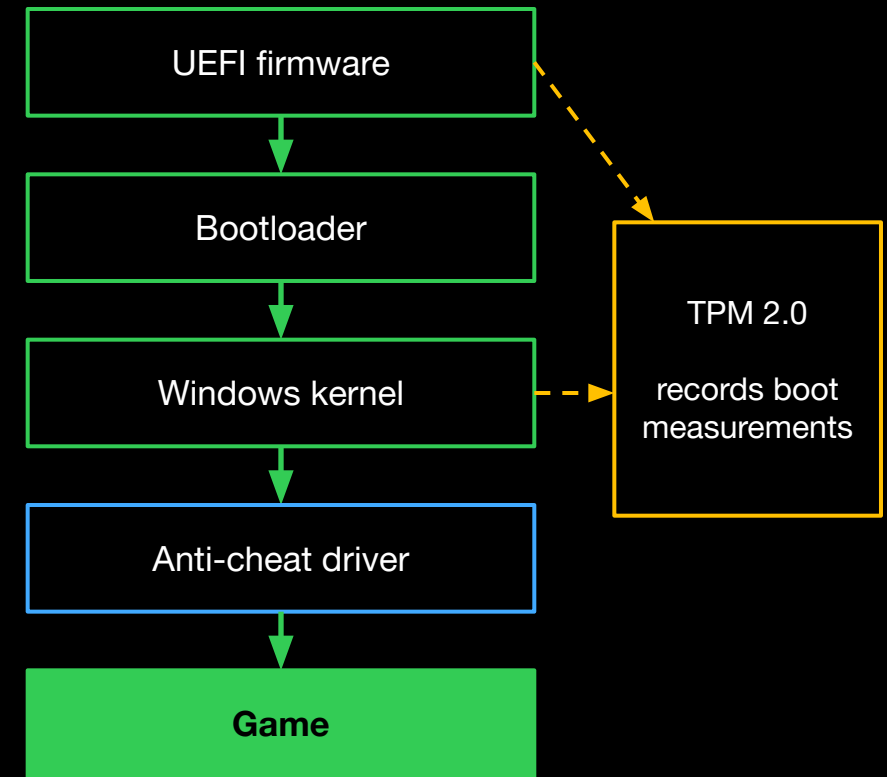
Modern Anti-cheats

- Ring 0 (Kernel Mode)
 - Full system access (can scan all memory, processes, drivers)
 - Monitor hardware events, syscalls and kernel callbacks
- Protection techniques
 - Heavy code obfuscation
 - Encrypt critical memory regions
 - Signature scans
- Invasive
 - Boot-level drivers
 - Continuous system activity monitoring



Secure Boot and TPM

- Shooters won't launch without them
 - Valorant (Win 11): TPM 2.0 + Secure Boot
 - Battlefield 6: Secure Boot (Javelin)
 - Black Ops 7 / Warzone: TPM + Secure Boot
- Secure Boot -> signed boot code only
- TPM -> hard-to-spoof hardware ID
 - Each TPM has a unique endorsement key
 - Measured boot -> server can verify boot

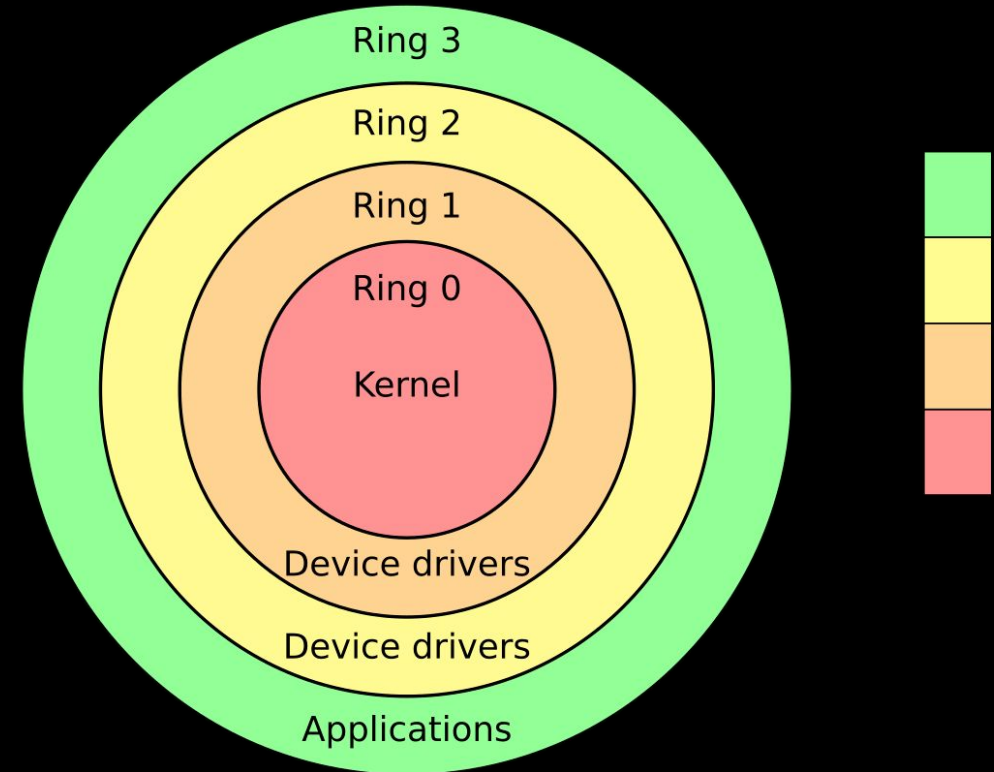


Secure Boot: firmware -> kernel
Windows enforces driver signing



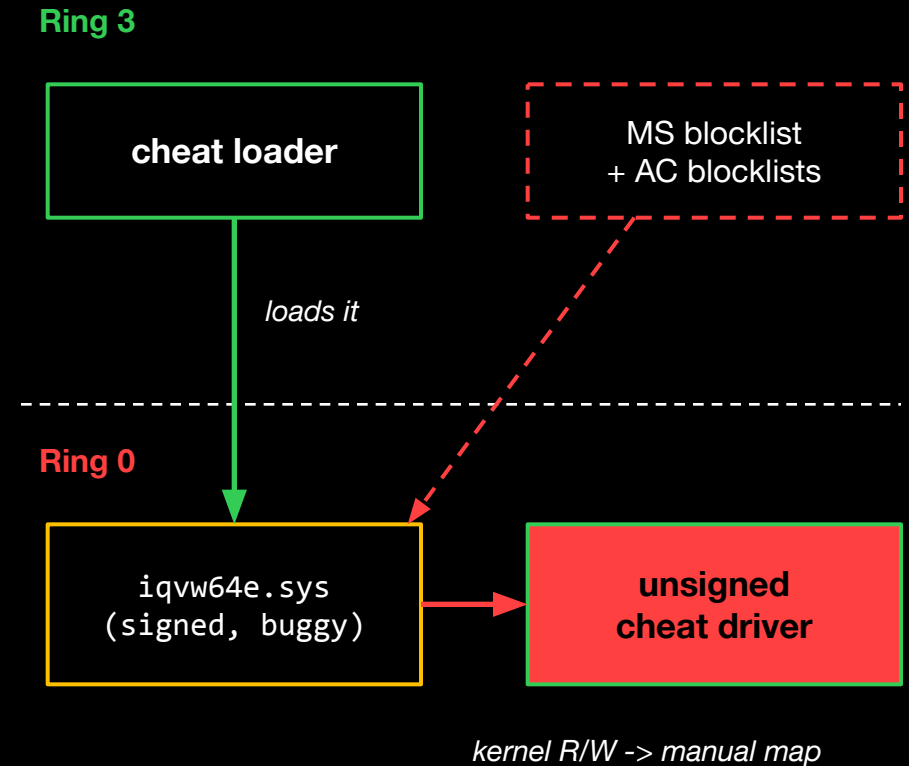
Drivers

- Kernel drivers (Ring 0)
- Cheat drivers
 - Read/write process memory from kernel
 - Hide processes, modules, and registry keys (rootkit techniques)
 - Disable anti-cheat callbacks and kernel protections
- Must be Microsoft-signed (EV cert + Hardware Dev Center)
- Vulnerable to behavioral analysis and signature scans
- Must start before the anti-cheat driver



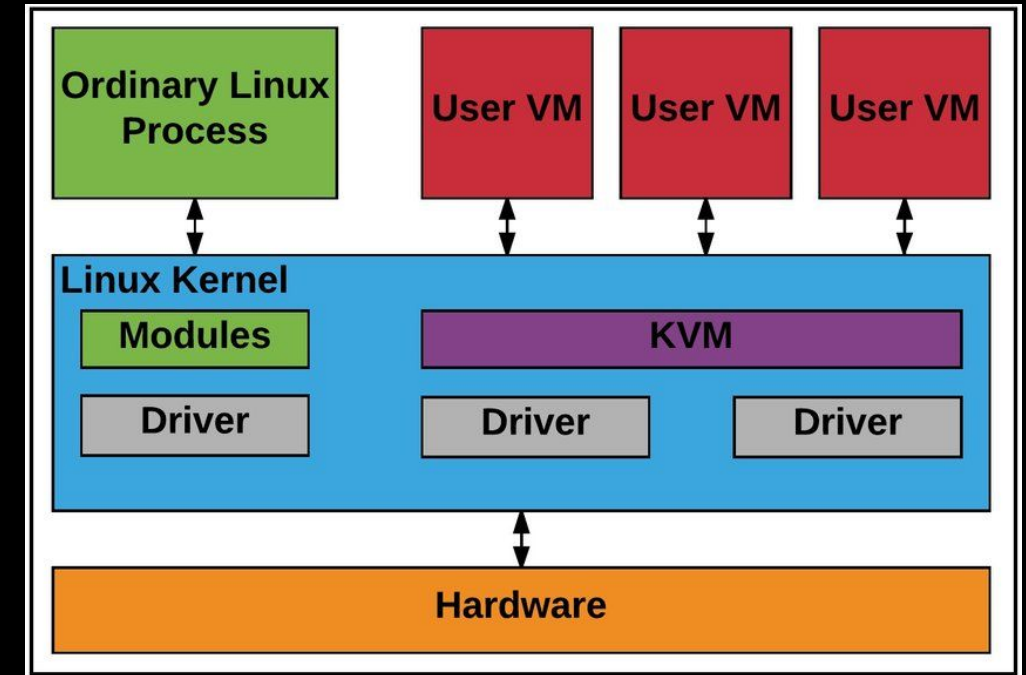
Bring Your Own Vulnerable Driver

- No signing cert? Borrow someone else's
 - Legit signed driver with a kernel R/W bug
 - Classic: `Capcom.sys` ran code in Ring 0
 - `kdmapper` abuses Intel's `iqvw64e.sys`
- Defenses
 - MS driver blocklist (default, Win 11 22H2+)
 - AC blocklists + leftover hunting (`PiDDBCacheTable`, `MmUnloadedDrivers`)



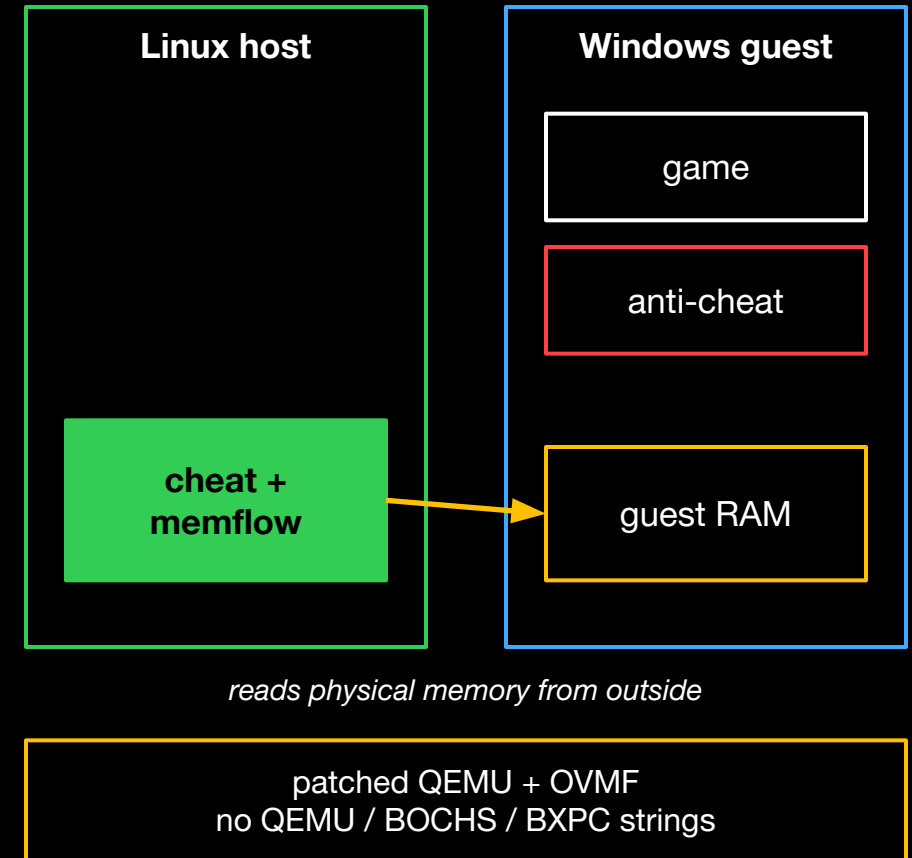
Kernel-based Virtual Machines

- Ring -1 (Hypervisor level)
- Hypervisor intercepts hardware instructions
- Can read/modify guest memory invisibly from outside the VM
- Run cheat on host and game in VM
- Anti-cheats check for VM artifacts, (e.g. CPUID flags, timing consistency, hypervisor presence)



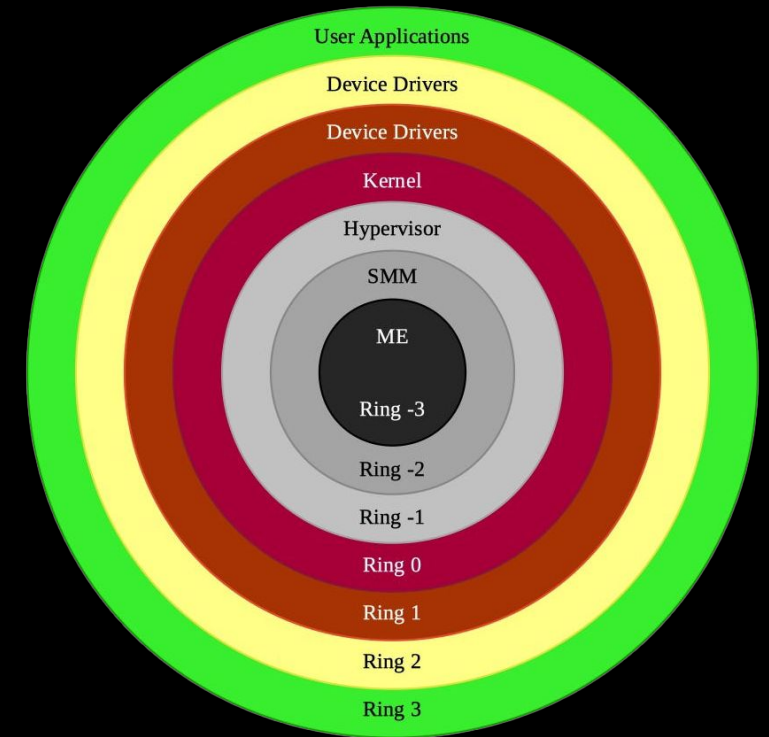
Hiding Virtual Machines

- Anti-cheats look for VM fingerprints
 - CPUID hypervisor bit, KVM signature
 - QEMU / BOCHS / BXPC in SMBIOS + ACPI
 - OVMF strings, virtio IDs, RDTSC timing
- Patch QEMU + OVMF to scrub them
 - Spoof SMBIOS with your real board
 - VFIO GPU passthrough -> looks bare-metal
- Game in KVM, cheat on the host
 - VMI (e.g. **memflow**) -> zero code in guest
 - Dev: snapshots, **gdbstub**, no host BSODs



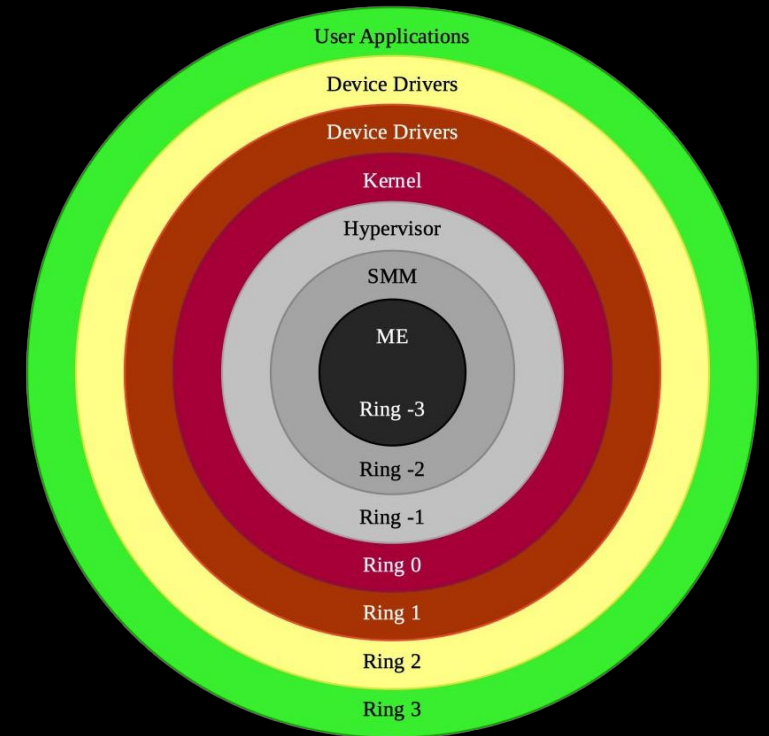
System Management Mode

- Ring -2 (SMM)
 - Most privileged x86 CPU mode, below hypervisor and kernel
- Special CPU mode triggered by System Management Interrupt (SMI)
- Has complete access to all physical memory and hardware
- Operates in SMRAM (System Management Ram)
- Can intercept USB traffic
- Full physical memory access, even with VBS/HVCI (unless firmware isolates SMM)



System Management Mode

- Execution flow
 - SMI triggered on USB event
 - CPU switches to SMM, saves game memory via physical addresses
 - Modifies USB mouse data buffer for aimbot
 - CPU exits SMM, restores OS
- Anti-cheats can't scan SMM memory
- Vulnerable to side-channel timing attacks
- Typically via an SPI flash patch or a UEFI exploit



Direct Memory Access

- PCIe card plugged into second computer
- Reads physical memory over PCIe bus without executing any code on target system
- Decrypt and process memory on second computer
- Vulnerable to PCIe device ID scans, VBS, side-channel attacks, and hardware heuristic detections
- Requires custom DMA firmware and FPGA firmware development skills to make undetected



Example with Valorant

- We read and decrypt player positions from Vanguard
- We do some trigonometry to calculate their position on our screen
- We render the ESP overlay on our monitor through OBS hooking



AI Cheats

- Computer Vision aimbots
 - Use capture card + AI model to detect enemies and aim
 - Typically use YOLO
 - Sends calculated mouse movements to target computer through microcontroller
- Microcontroller acts as an HID device
- Vulnerable to player statistic heuristic detections and device descriptor scanning



Resources

- Cheat Engine ([GitHub](#))
- PolyHook2 ([GitHub](#))
- GuidedHacking ([Website](#))
- dotPeek ([Website](#))



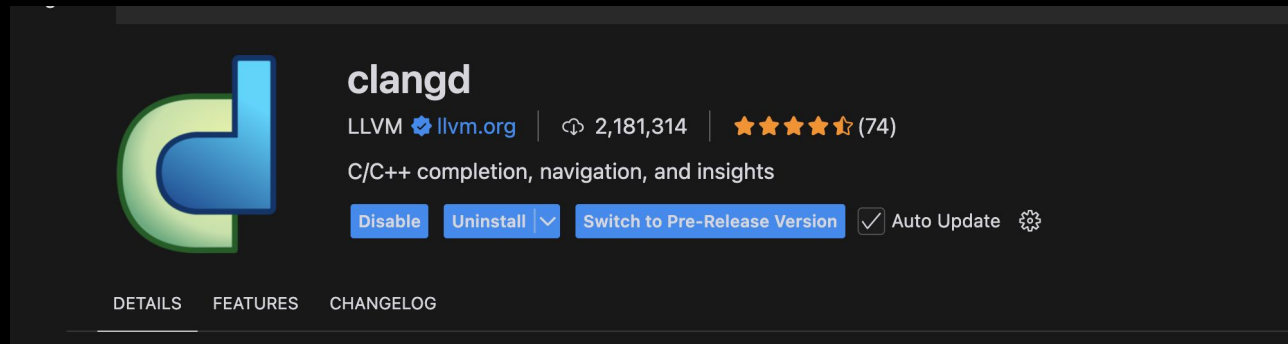
Setup (CMake + Ninja + MSVC)

- Download and install [CMake](#).
 - `winget install --id=Kitware.CMake -e`
- Download and install [Visual Studio 2022](#).
 - Make sure to enable **Desktop development with C++**.
- Download and install [Ninja](#).
 - `winget install --id=Ninja-build.Ninja -e`



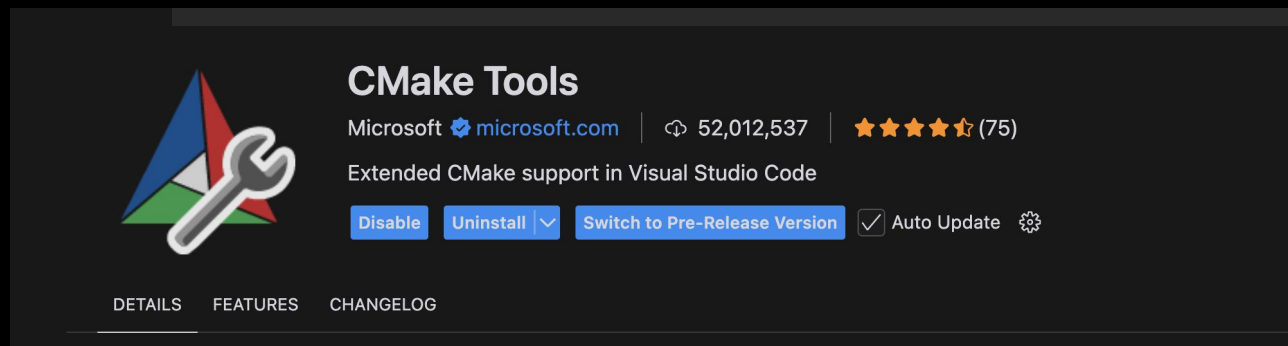
Visual Studio Code

- Install the `clangd` and `CMake Tools` extensions



The image shows the Visual Studio Code extension marketplace entry for 'clangd'. It features a large blue and green 'd' logo. The text includes the extension name 'clangd', the publisher 'LLVM' with a link to 'llvm.org', a download count of '2,181,314', and a rating of '4.5 stars (74)'. A description states 'C/C++ completion, navigation, and insights'. At the bottom, there are buttons for 'Disable', 'Uninstall', and 'Switch to Pre-Release Version', along with a checked 'Auto Update' checkbox and a settings gear icon. Tabs for 'DETAILS', 'FEATURES', and 'CHANGELOG' are visible at the bottom.

clangd
LLVM llvm.org | 2,181,314 | ★★★★★ (74)
C/C++ completion, navigation, and insights
Disable Uninstall Switch to Pre-Release Version ☒ Auto Update ⚙️
DETAILS FEATURES CHANGELOG



The image shows the Visual Studio Code extension marketplace entry for 'CMake Tools'. It features a logo consisting of a blue, red, and green triangle with a grey wrench. The text includes the extension name 'CMake Tools', the publisher 'Microsoft' with a link to 'microsoft.com', a download count of '52,012,537', and a rating of '4.5 stars (75)'. A description states 'Extended CMake support in Visual Studio Code'. At the bottom, there are buttons for 'Disable', 'Uninstall', and 'Switch to Pre-Release Version', along with a checked 'Auto Update' checkbox and a settings gear icon. Tabs for 'DETAILS', 'FEATURES', and 'CHANGELOG' are visible at the bottom.

CMake Tools
Microsoft microsoft.com | 52,012,537 | ★★★★★ (75)
Extended CMake support in Visual Studio Code
Disable Uninstall Switch to Pre-Release Version ☒ Auto Update ⚙️
DETAILS FEATURES CHANGELOG



Build and Inject the Template DLL

- Create a new repository using the [template](#).
- Open the repository in Visual Studio Code
- Set `cmake.generator` to `Ninja` in settings
- Select `Visual Studio Community 2022 Release - amd64`
- Build
- Run with `dll-injector.exe <pid> <dll_path>`
 - e.x. `dll-injector.exe 1337 template-dll.dll`



Next Meetings

2026-10-25 • This Sunday

- Download More RAM: Dismantling Windows Operating System Defences with Mischievous Memory
- Our first seminar of the semester!



ctf.sigpwny.com

sigpwny{sk111_155u3_p47ch3d}

Meeting content can be found at
sigpwny.com/meetings.

