SIGPwny

General FA2025 • 2025-11-06

Game Hacking

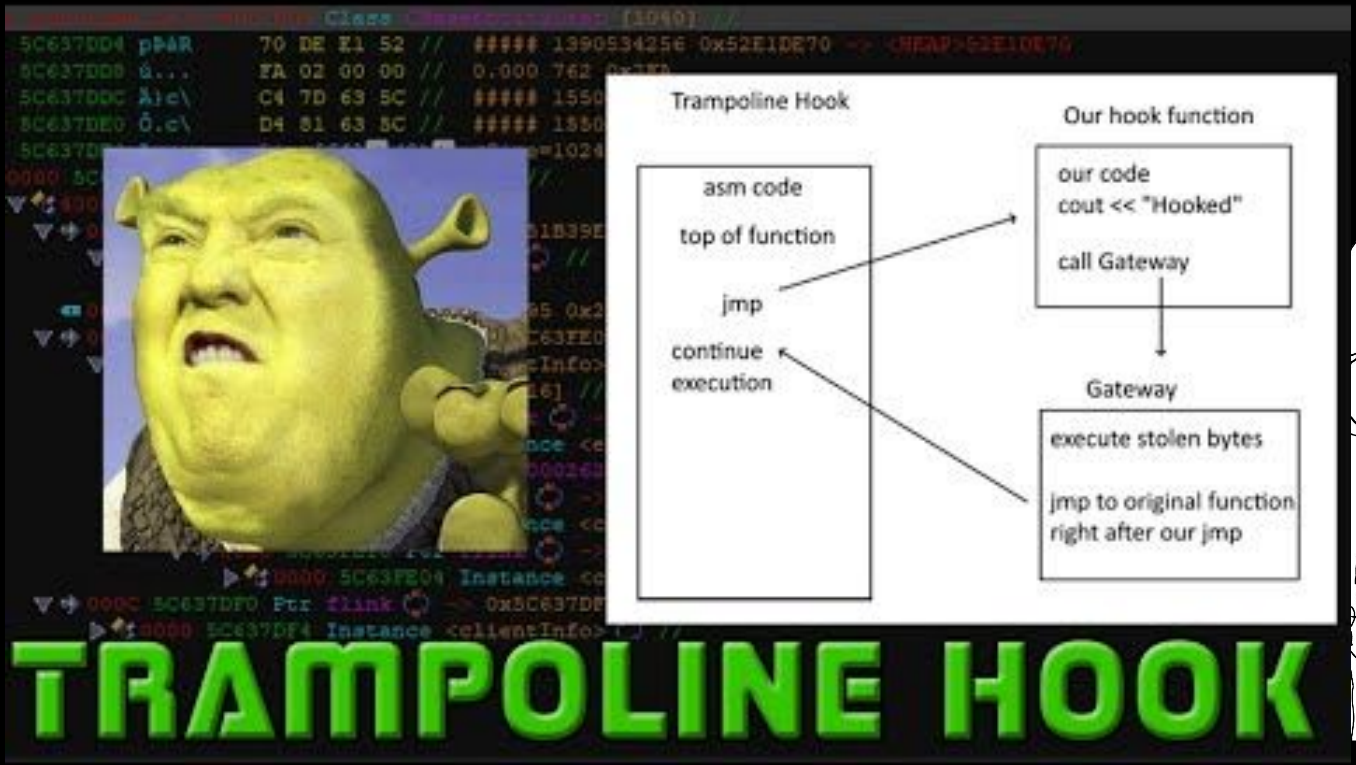
Tyler Mercado

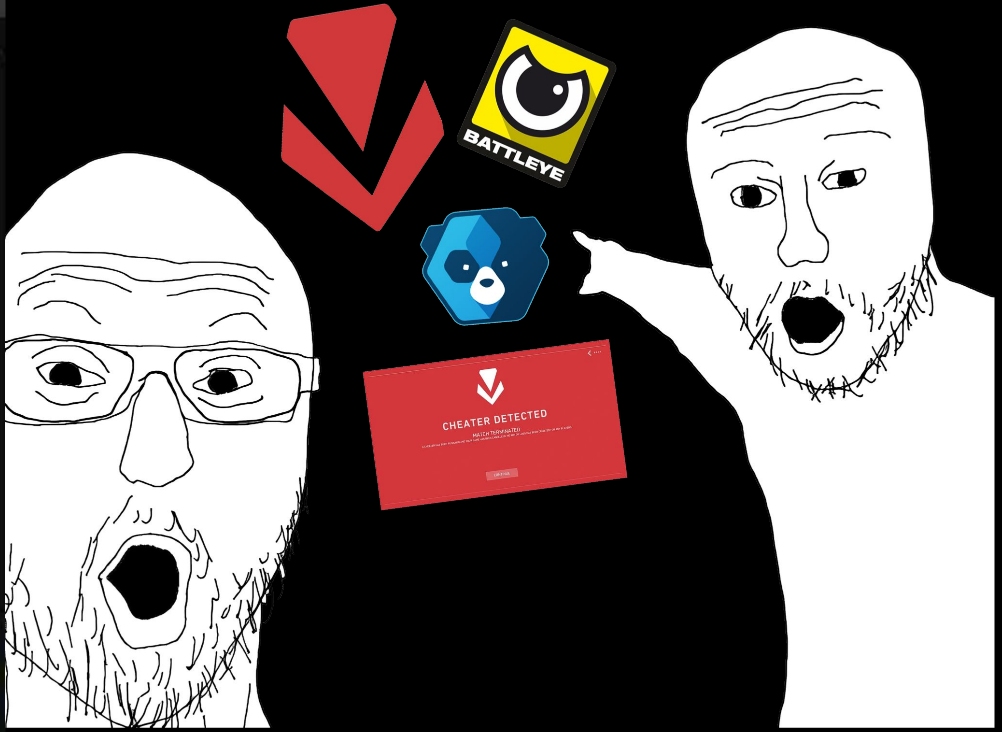
Announcements

- 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{ju57_h00k_7h3_ch3ck5}

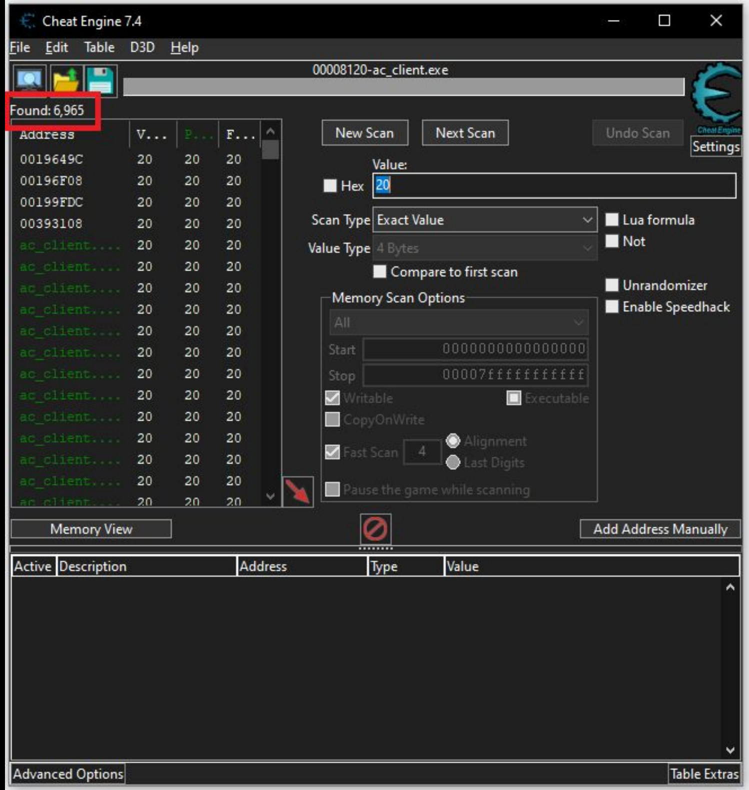




What is Game Hacking?

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



00008120-ac_client.exe

Found: 6,965

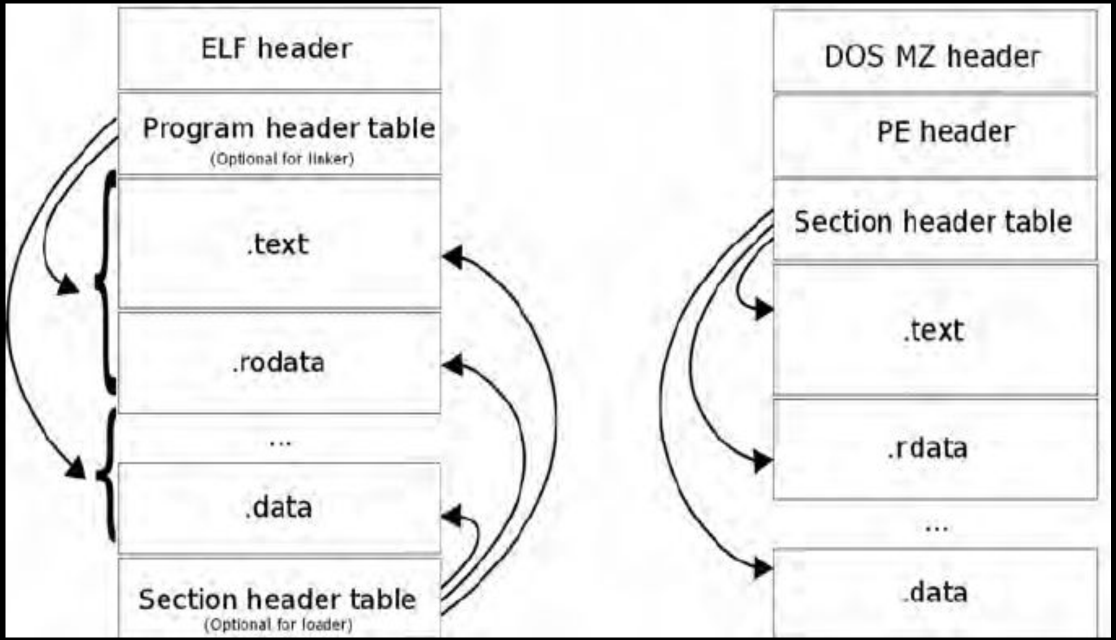
Address	V...	P...	F...
0019649C	20	20	20
00196F08	20	20	20
00199FDC	20	20	20
00393108	20	20	20
ac_client....	20	20	20
ac_client....	20	20	20
ac_client....	20	20	20
ac_client....	20	20	20
ac_client....	20	20	20
ac_client....	20	20	20
ac_client....	20	20	20
ac_client....	20	20	20
ac_client....	20	20	20
ac_client....	20	20	20
ac_client....	20	20	20
ac_client....	20	20	20
ac_client....	20	20	20

Value: 20
□ Hex
Scan Type: Exact Value
Value Type: 4 Bytes
□ Compare to first scan
□ Lua formula
□ Not
□ Unrandomizer
□ Enable Speedhack
All
Start: 0000000000000000
Stop: 00007fffffffffff
☑ Writable
□ Executable
□ CopyOnWrite
☑ Fast Scan 4
● Alignment
○ Last Digits
□ Pause the game while scanning

Active	Description	Address	Type	Value

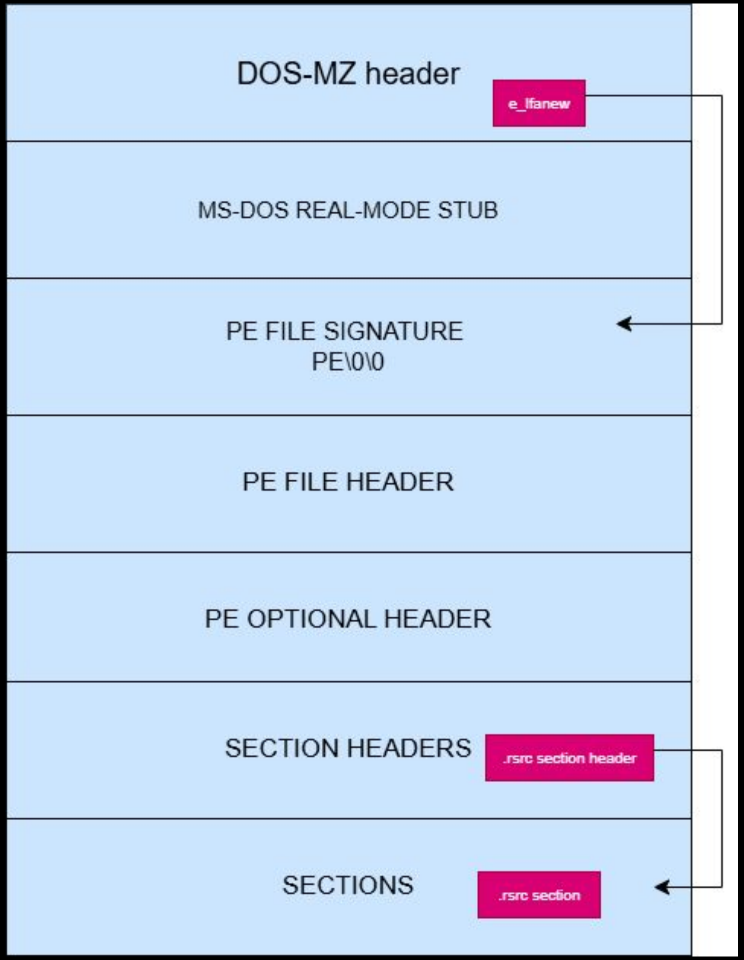
Windows PE vs. Linux ELF

- `.rodata` stores read-only constants
- `.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



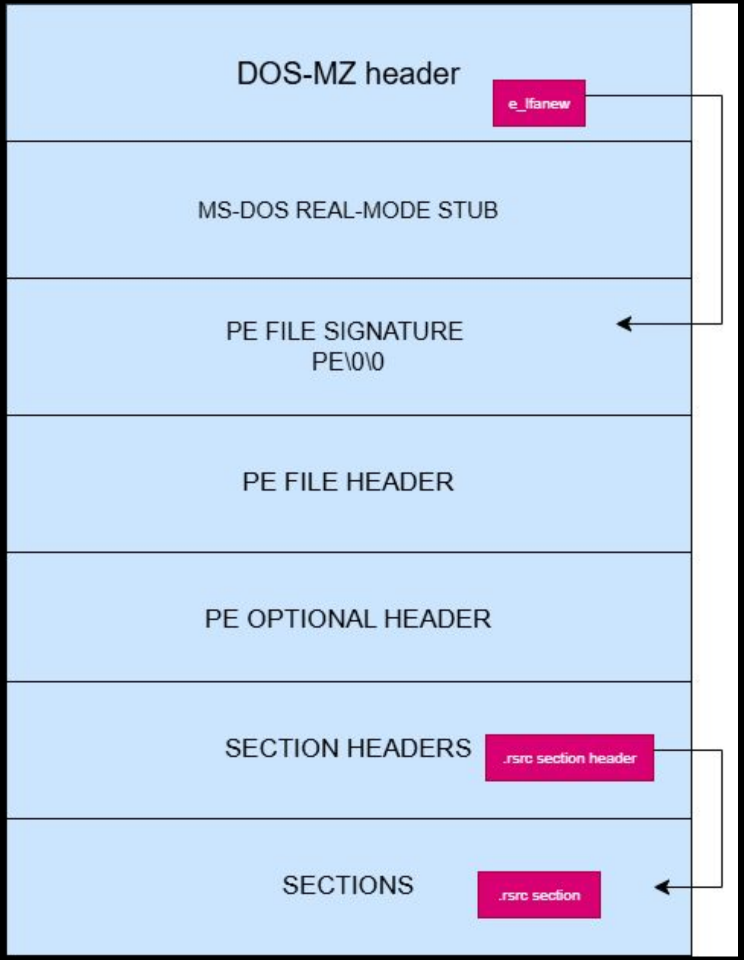
PE Header

- 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?

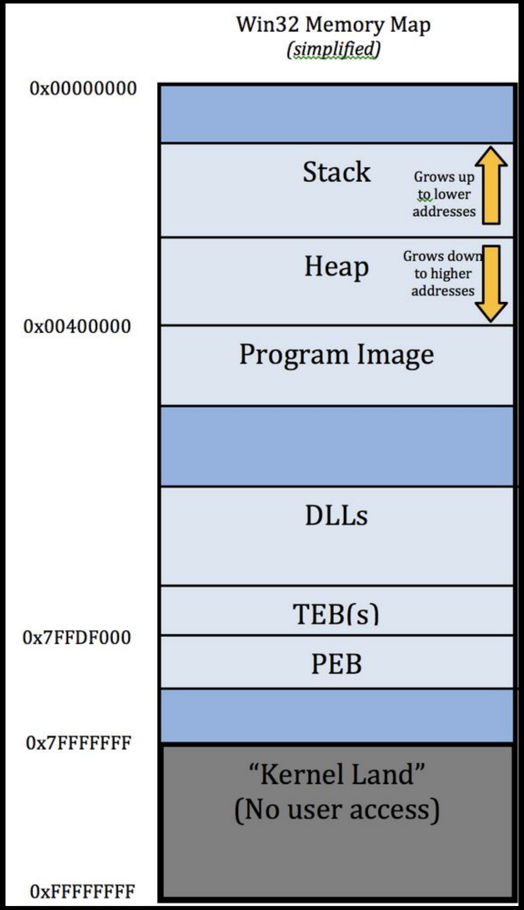
It reveals memory layout, and imported functions to hook.

Base address matters because of ASLR!



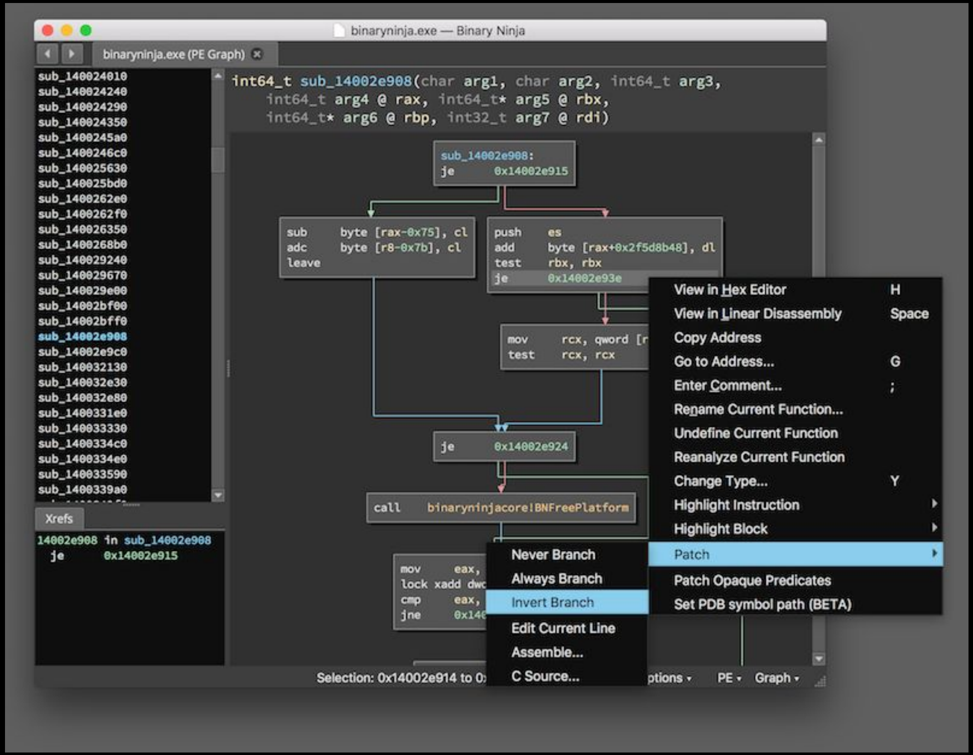
Virtual Memory in Windows

- Each process has isolated 4GB address space (32-bit) or 16EB (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, …)`



```
sub_140024010
sub_140024240
sub_140024290
sub_140024350
sub_1400245a0
sub_1400246c0
sub_140025630
sub_140025bd0
sub_1400262e0
sub_1400262f0
sub_140026350
sub_1400268b0
sub_140029240
sub_140029670
sub_140029e00
sub_14002bf00
sub_14002bff0
sub_14002e908
sub_14002e9c0
sub_140032130
sub_140032e30
sub_140032e80
sub_1400331e0
sub_140033330
sub_1400334c0
sub_1400334e0
sub_140033590
sub_1400339a0
```

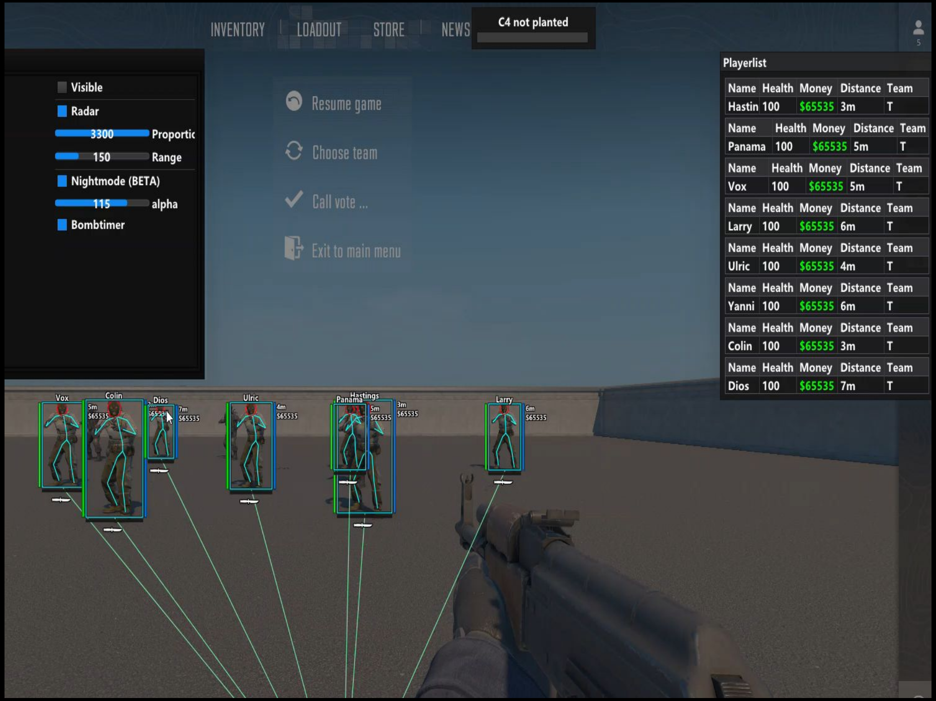
```
14002e908 in sub_14002e908
  je      0x14002e915
```

```
int64_t sub_14002e908(char arg1, char arg2, int64_t arg3,
    int64_t arg4 @ rax, int64_t* arg5 @ rbx,
    int64_t* arg6 @ rbp, int32_t arg7 @ rdi)

sub_14002e908:
je      0x14002e915

sub     byte [rax-0x75], cl
adc     byte [r8-0x7b], cl
leave

push    es
add     byte [rax+0x2f5d8b48], dl
test    rbx, rbx
je      0x14002e93e

mov     rcx, qword [r
test    rcx, rcx

je      0x14002e924

call    binaryninjacore!BNFreePlatform

mov     eax,
lock xadd dwo
cmp     eax,
jne     0x140
```

Selection: 0x14002e914 to 0x

External Cheats

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



□ Visible
■ Radar
3300 Proportic
150 Range
■ Nightmode (BETA)
115 alpha
■ Bombtimer

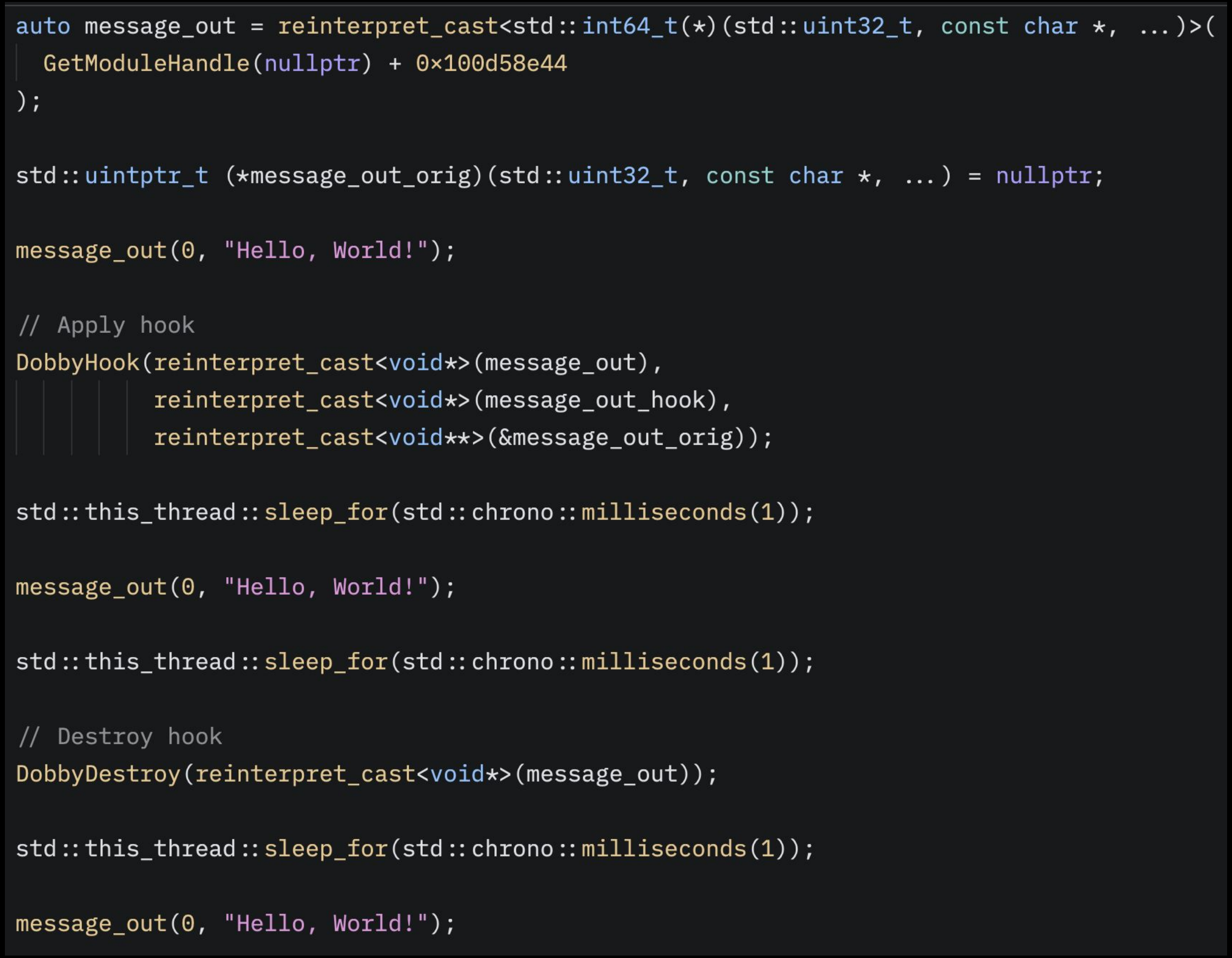
Playerlist

Name	Health	Money	Distance	Team
Hastin	100	$65535	3m	T
Name	Health	Money	Distance	Team
Panama	100	$65535	5m	T
Name	Health	Money	Distance	Team
Vox	100	$65535	5m	T
Name	Health	Money	Distance	Team
Larry	100	$65535	6m	T
Name	Health	Money	Distance	Team
Ulric	100	$65535	4m	T
Name	Health	Money	Distance	Team
Yanni	100	$65535	6m	T
Name	Health	Money	Distance	Team
Colin	100	$65535	3m	T
Name	Health	Money	Distance	Team
Dios	100	$65535	7m	T

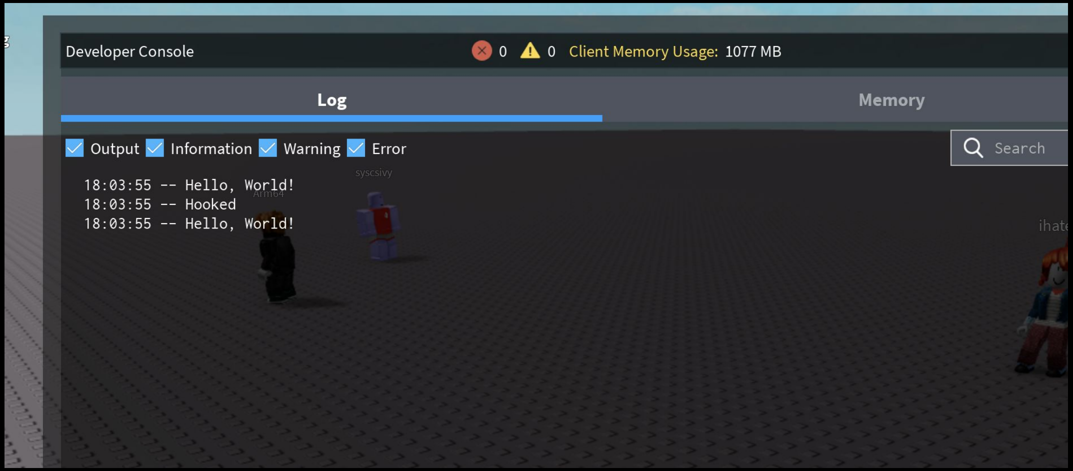
Internal Cheats

- 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!");
```

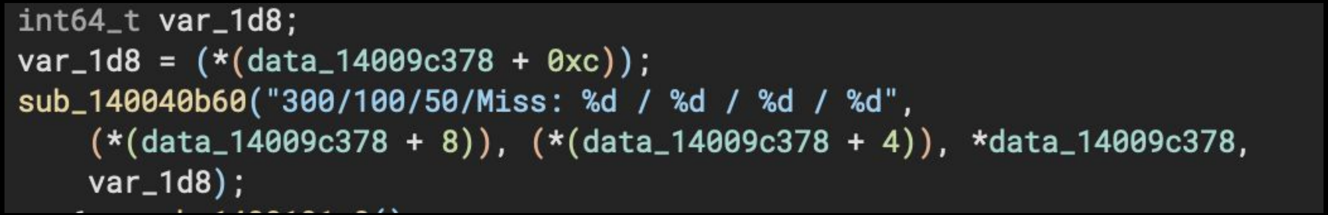


Developer Console
0
0
Client Memory Usage: 1077 MB
Log
Memory
☑ Output ☑ Information ☑ Warning ☑ Error
Search
syscsivy
ihate

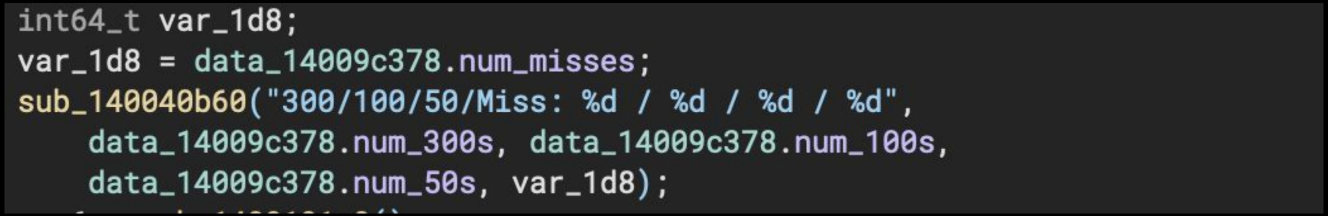
```
18:03:55 -- Hello, World!
18:03:55 -- Hooked
18:03:55 -- Hello, World!
```

Reversing Structs

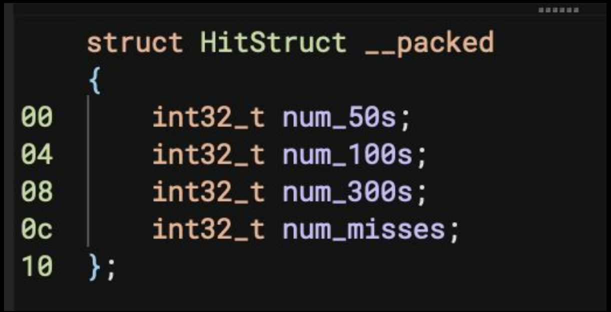
- Analyze memory access patterns
  - Track how pointers are dereferences e.g. `*(data + 0xc)`, `*(data + 8)`
- Identify offsets and determine types
  - Each offset reveals a struct member location
- We can access/modify game data directly via typed structs instead of raw pointers



```
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);
```



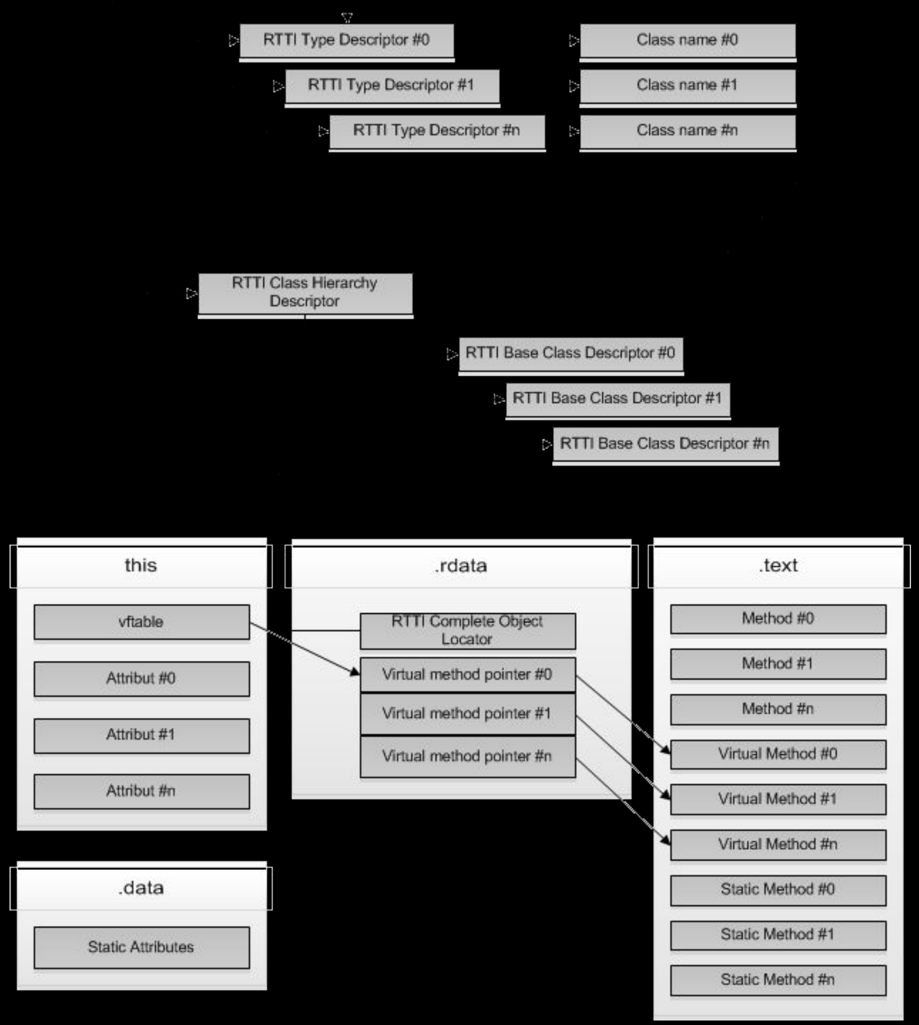
```
int64_t var_1d8;
var_1d8 = data_14009c378.num_misses;
sub_140040b60("300/100/50/Miss: %d / %d / %d / %d",
    data_14009c378.num_300s, data_14009c378.num_100s,
    data_14009c378.num_50s, var_1d8);
```



```
    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

- Windows 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()



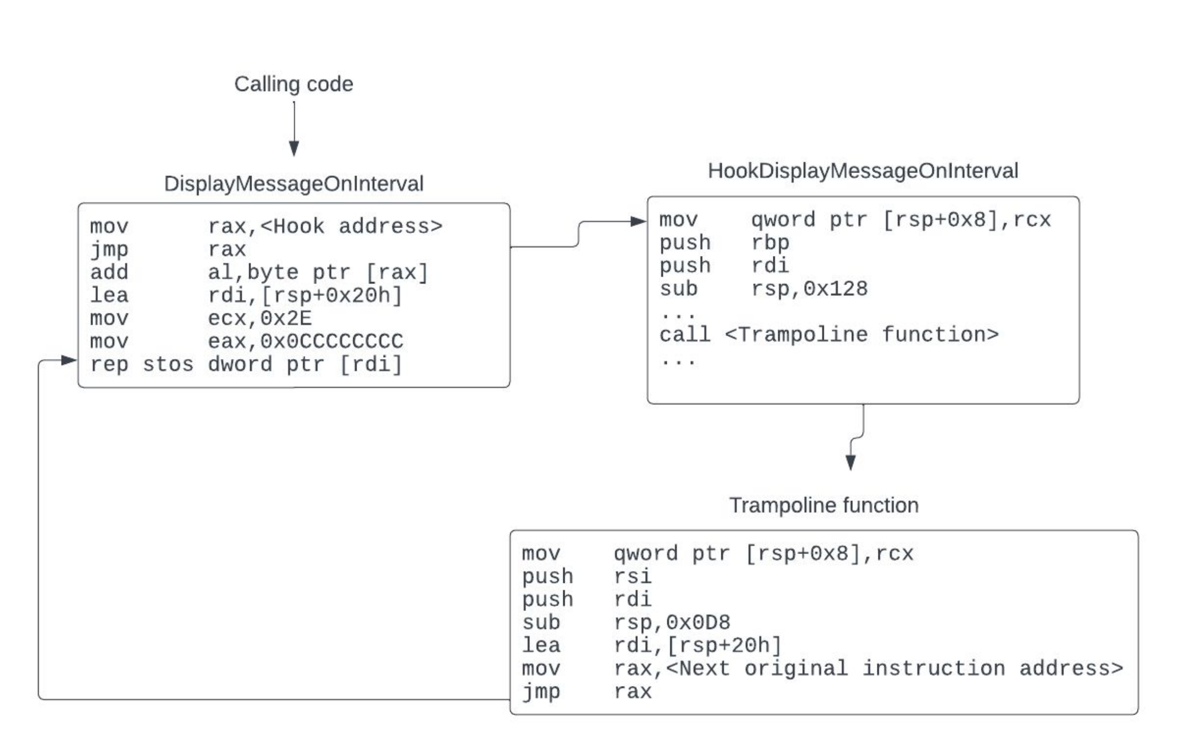
DLL Injection

- 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



Calling code

DisplayMessageOnInterval

```
mov         rax,<Hook address>
jmp         rax
add         al,byte ptr [rax]
lea         rdi,[rsp+0x20h]
mov         ecx,0x2E
mov         eax,0x0CCCCCCCC
rep stos dword ptr [rdi]
```

HookDisplayMessageOnInterval

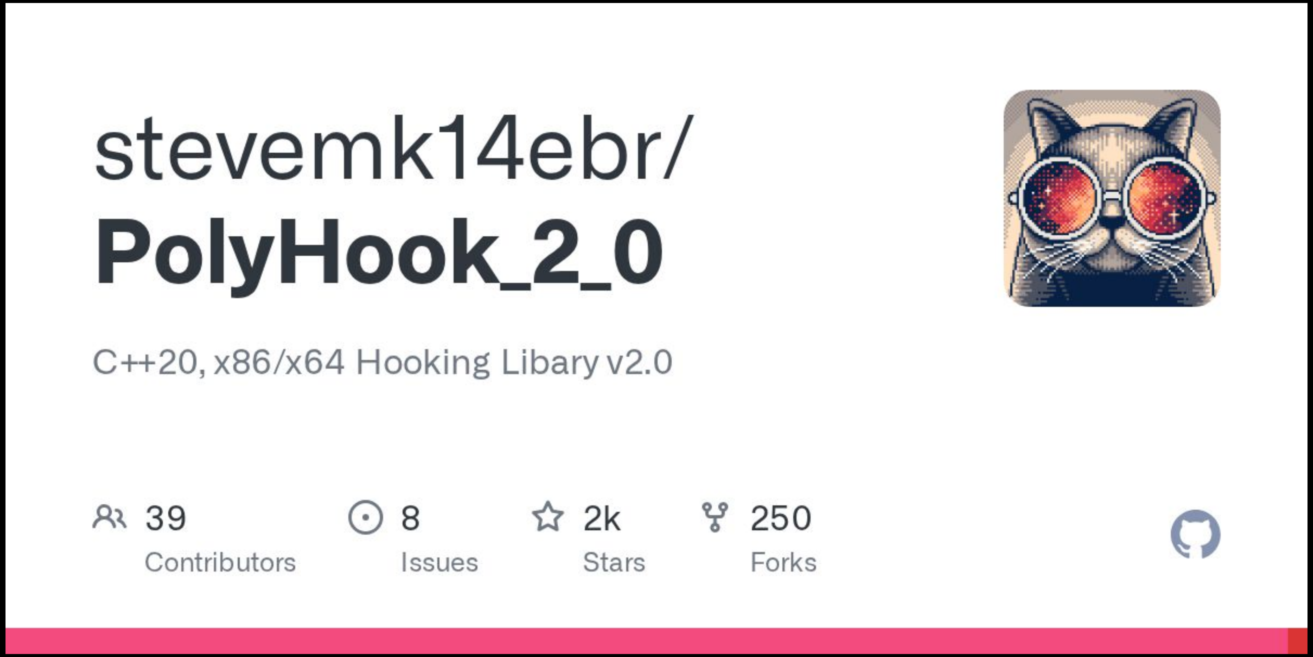
```
mov     qword ptr [rsp+0x8],rcx
push    rbp
push    rdi
sub     rsp,0x128
...
call <Trampoline function>
...
```

Trampoline function

```
mov     qword ptr [rsp+0x8],rcx
push    rsi
push    rdi
sub     rsp,0x0D8
lea     rdi,[rsp+20h]
mov     rax,<Next original instruction address>
jmp     rax
```

Hooking Functions

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

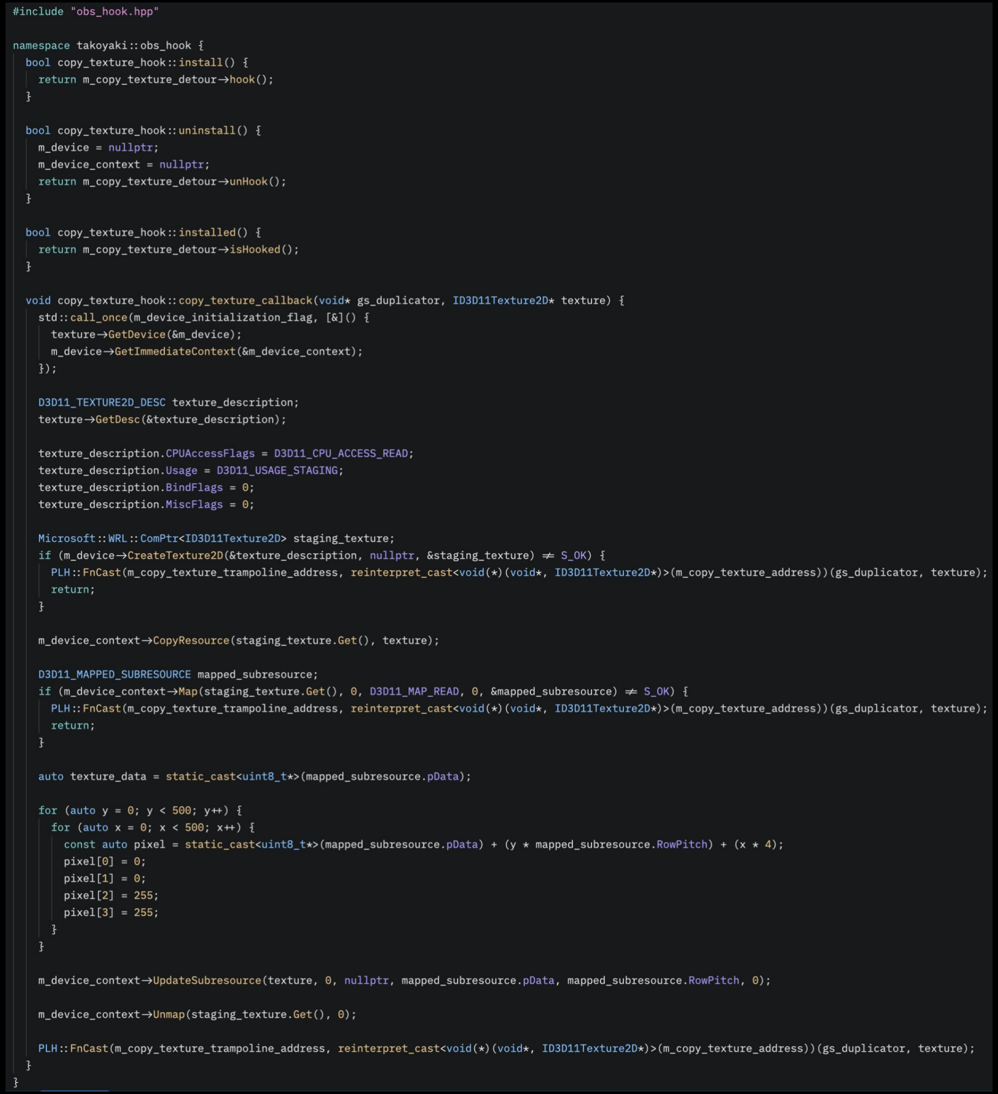


stevemk14ebr/
PolyHook_2_0

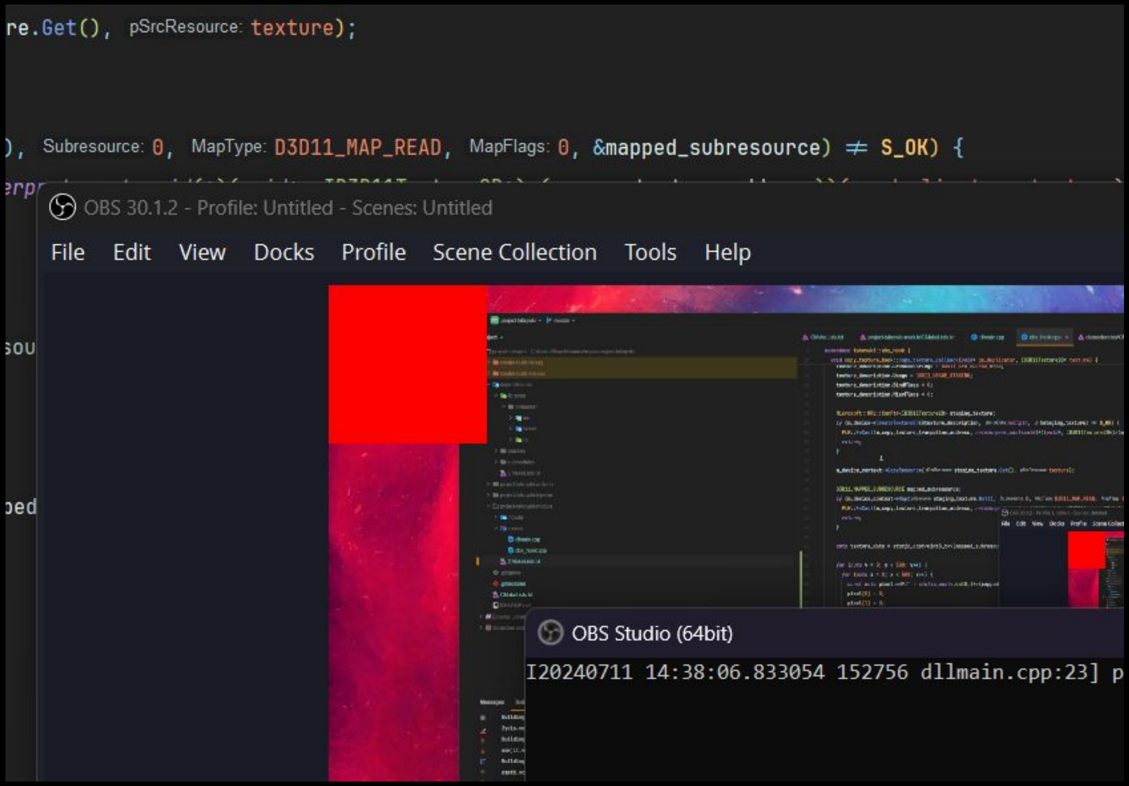
C++20, x86/x64 Hooking Libary v2.0

39 Contributors · 8 Issues · 2k Stars · 250 Forks

Example with OBS: Creating Overlays



```cpp
#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);
  }
}
```

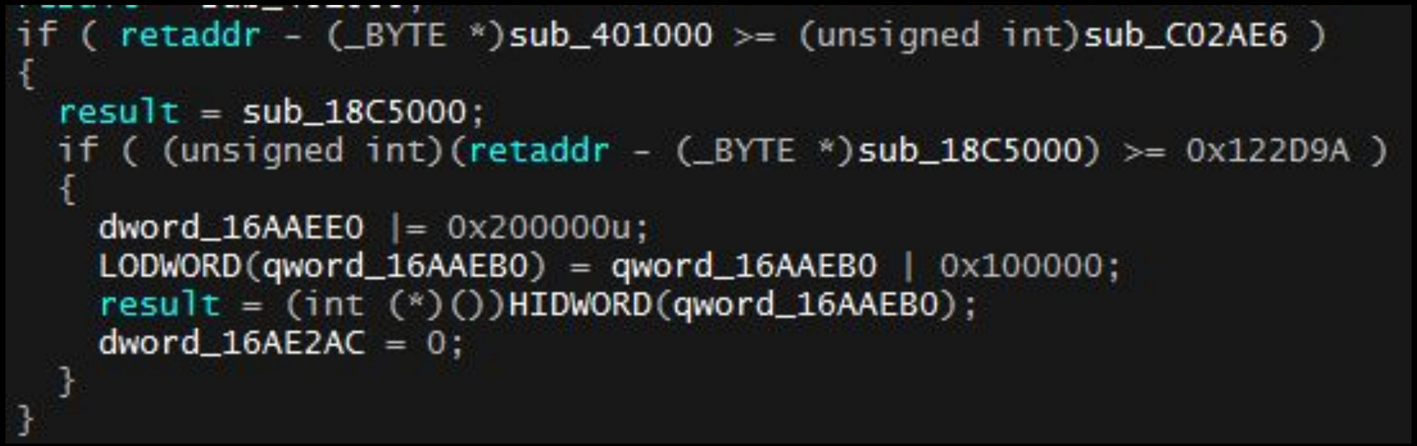


```
re.Get(), pSrcResource: texture);

), Subresource: 0, MapType: D3D11_MAP_READ, MapFlags: 0, &mapped_subresource) ≠ S_OK) {
```

```
I20240711 14:38:06.833054 152756 dllmain.cpp:23] p
```

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



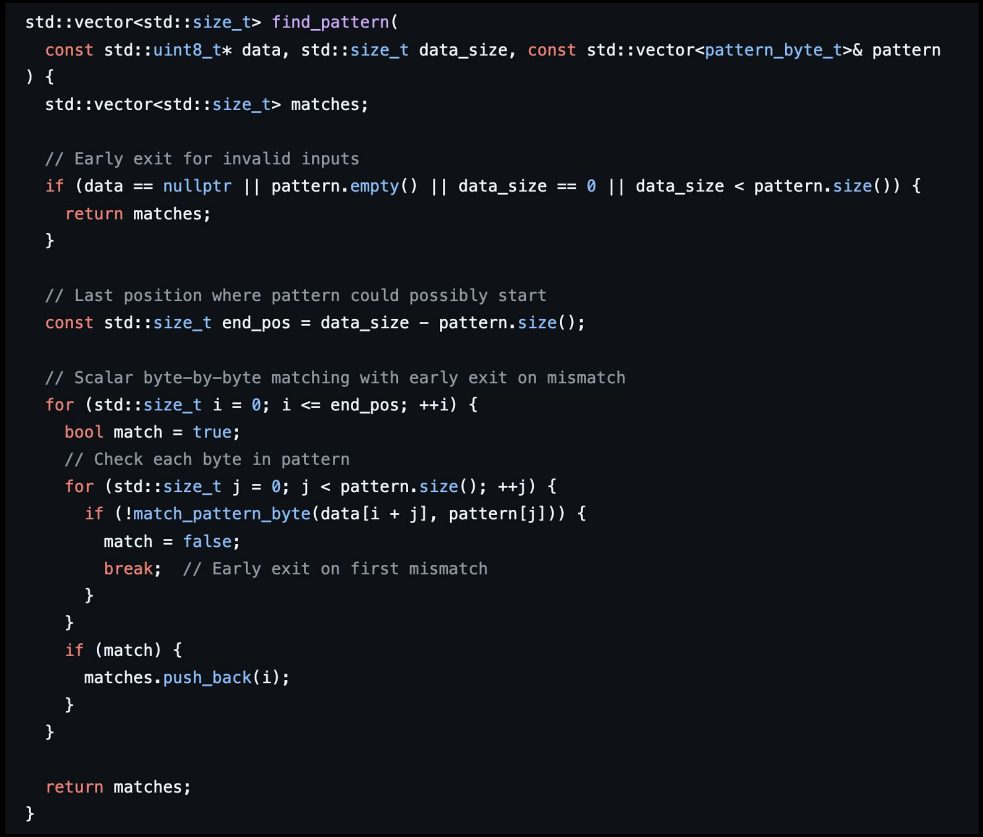
```
if ( retaddr - (_BYTE *)sub_401000 >= (unsigned int)sub_C02AE6 )
{
  result = sub_18C5000;
  if ( (unsigned int)(retaddr - (_BYTE *)sub_18C5000) >= 0x122D9A )
  {
    dword_16AAEE0 |= 0x200000u;
    LODWORD(qword_16AAEB0) = qword_16AAEB0 | 0x100000;
    result = (int (*)())HIDWORD(qword_16AAEB0);
    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

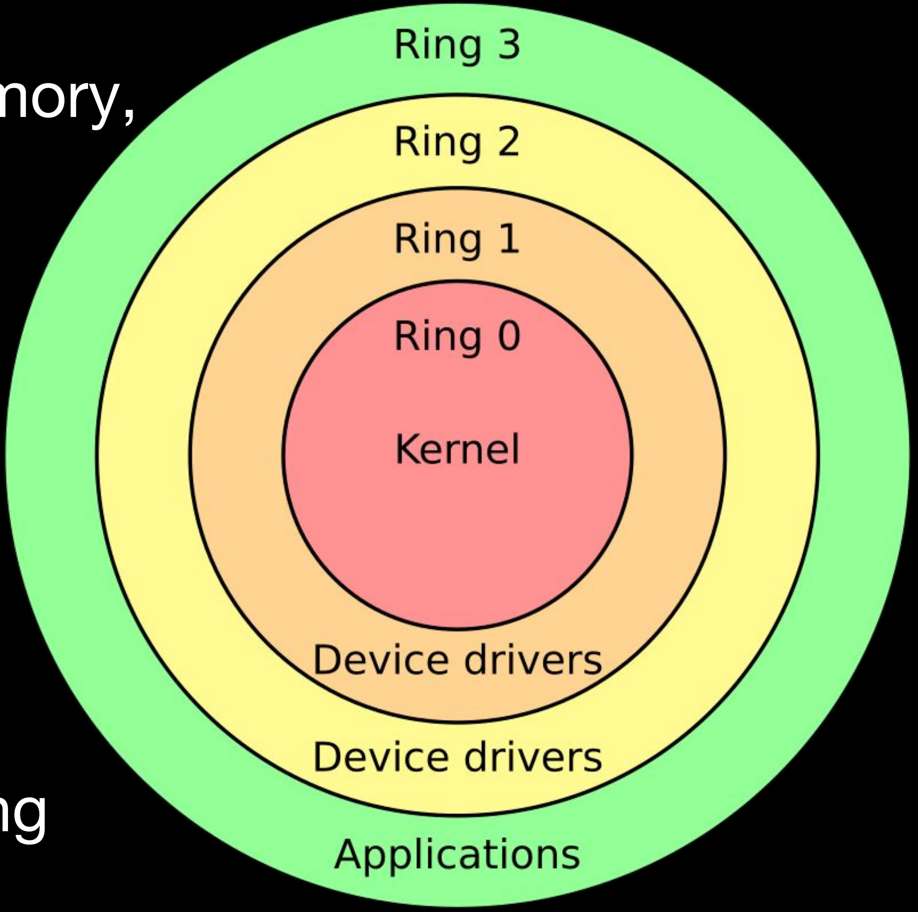
- Scans memory for known cheat signatures
- Checks running processes file hashes against known cheats
- Trivially, hook the checks
- Alternatively, obfuscate and pack your code + shuffle structs and constants



```
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;
}
```

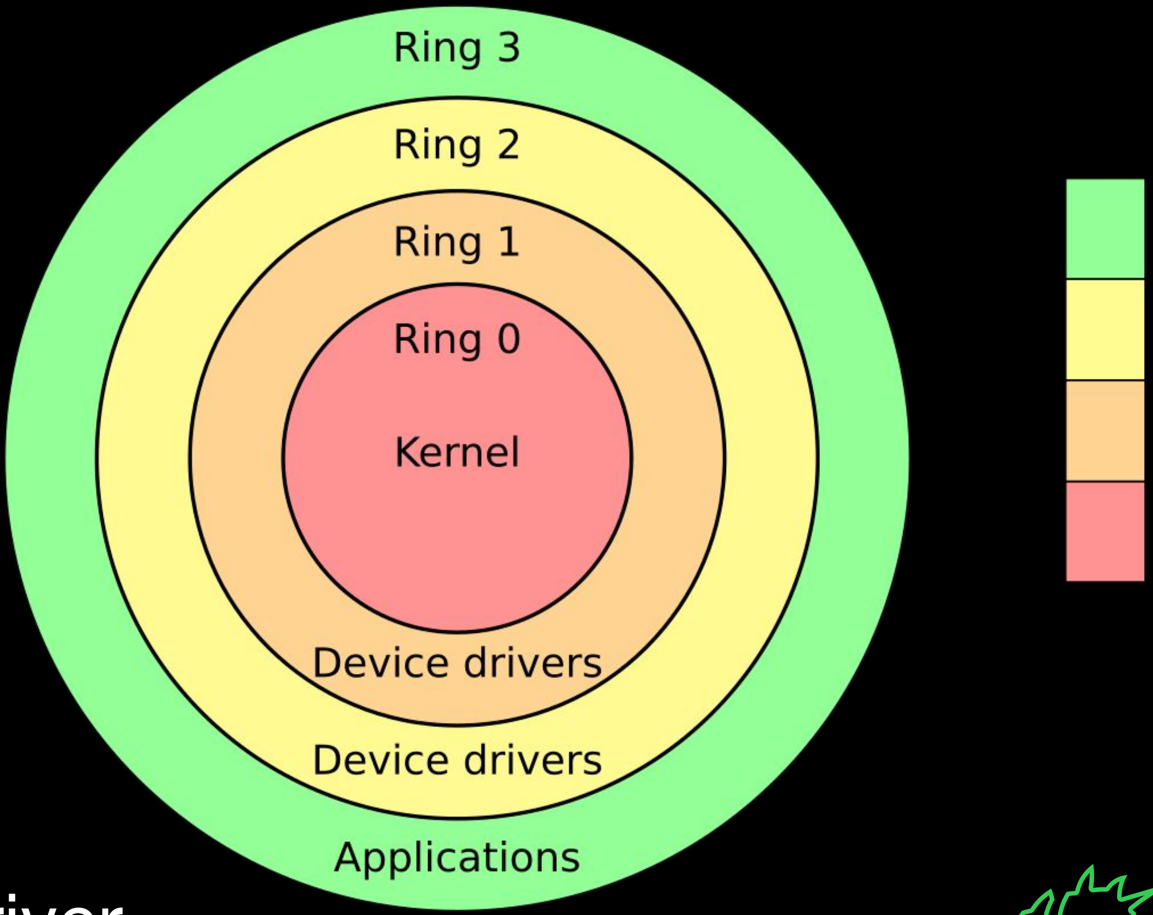
Modern Anti-cheats

- 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



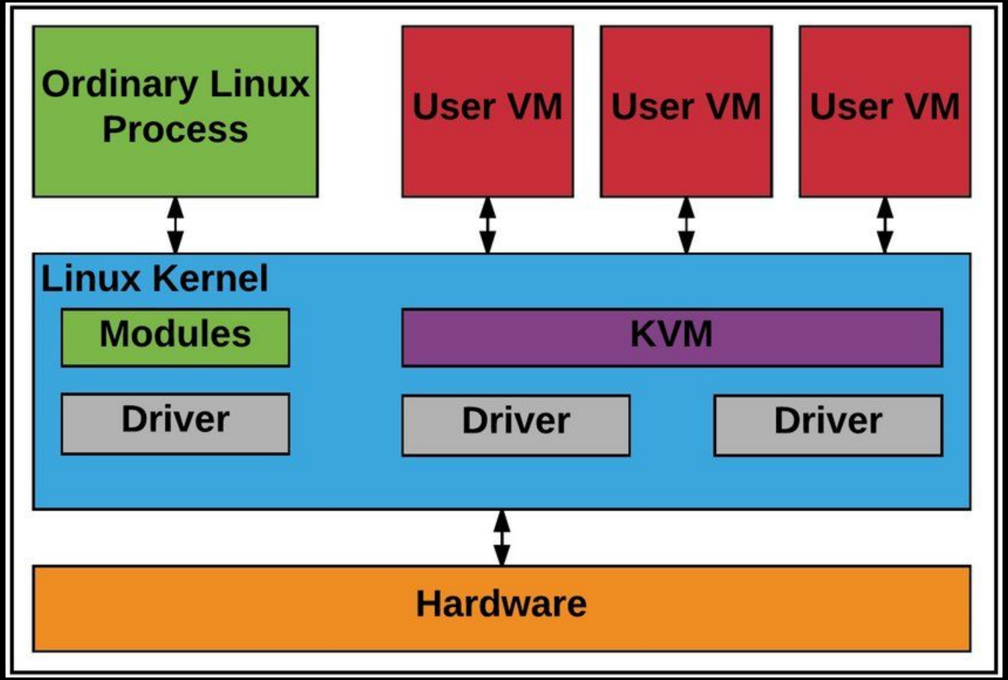
Drivers

- Kernel drivers (Ring 0)
- Cheat drivers
  - Read/write process memory from kernel
  - Hide processes, modules, and registry keys (roootkit techniques)
  - Disable anti-cheat callbacks and kernel protections
- Requires valid code signing certificate
- Vulnerable to behavioral analysis and signature scans
- Must start before the anti-cheat driver



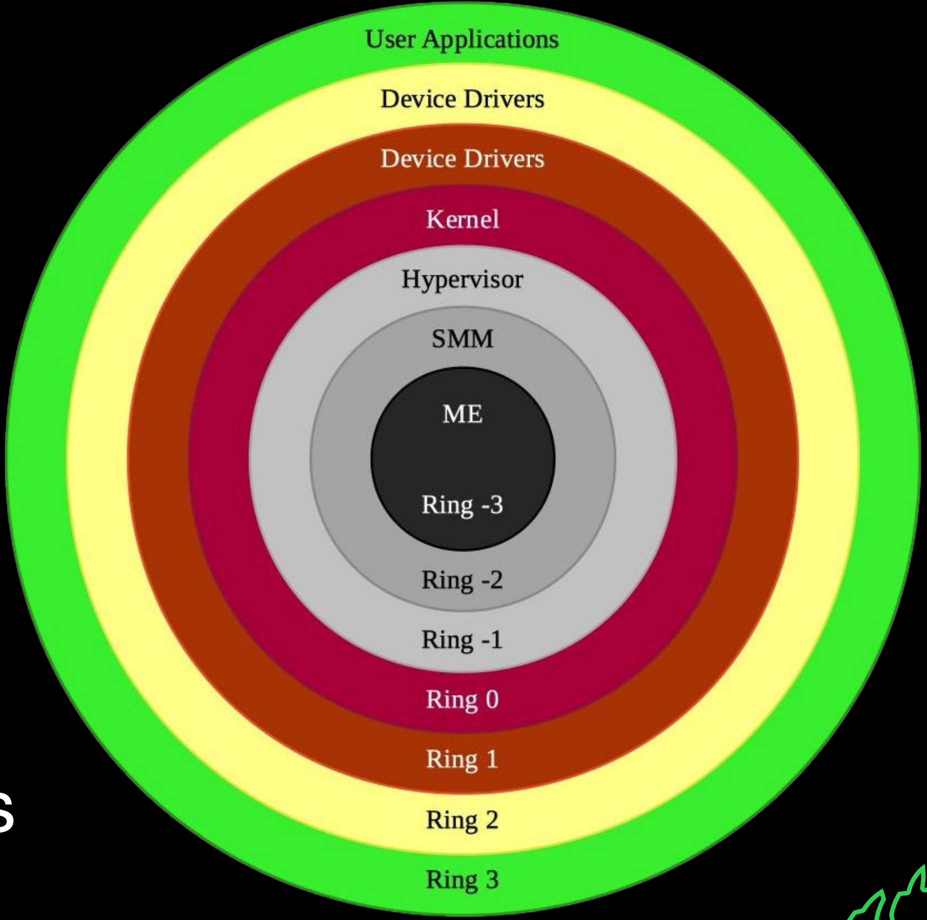
Kernel-based Virtual Machines

- 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)



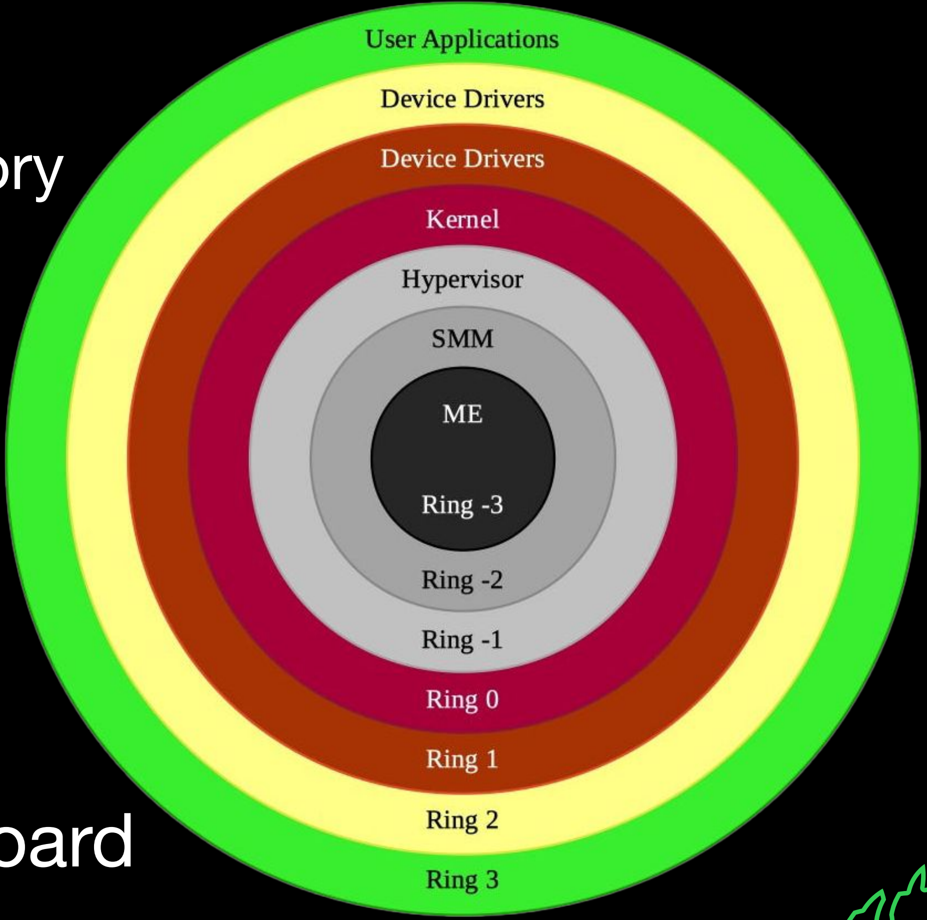
System Management Mode

- 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 regardless of VBS/HVCI settings



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
- You need a BIOS programmer to flash custom UEFI firmware to your motherboard

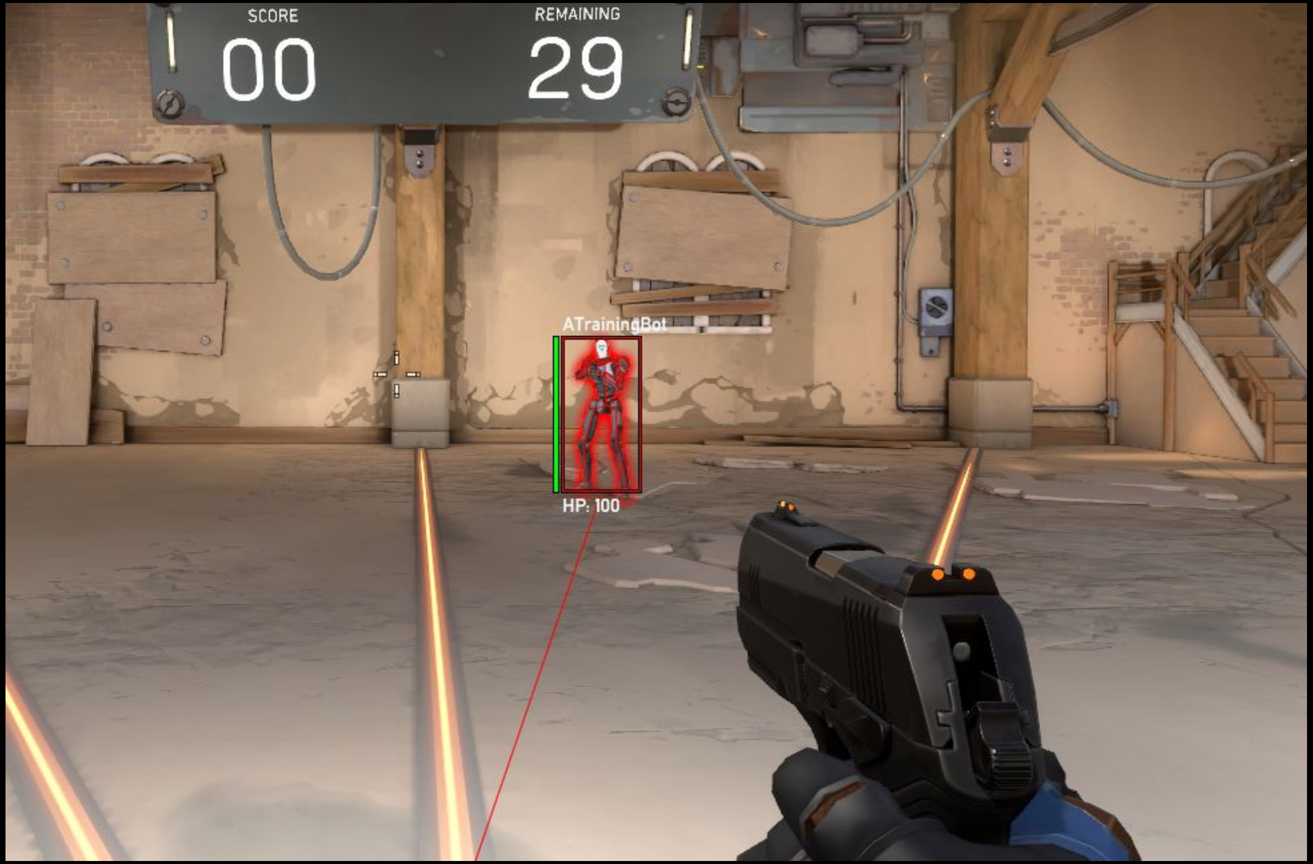


Direct Memory Access

- 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

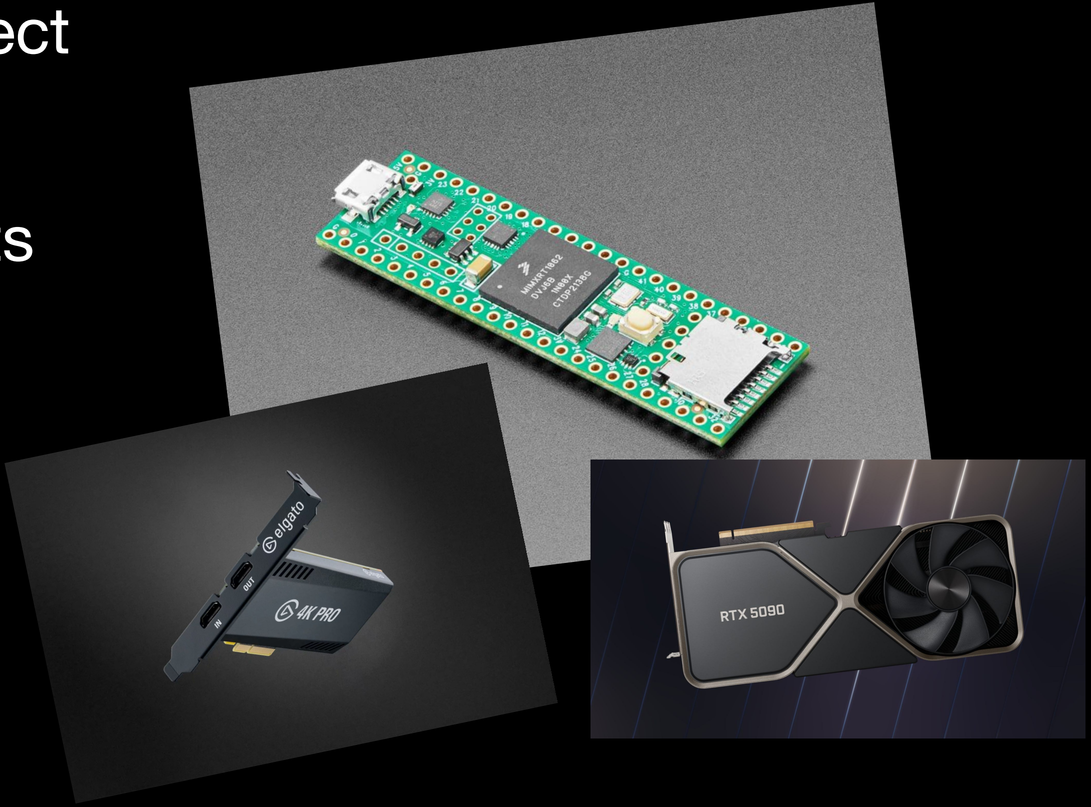
Example with Valorant

- 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



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 can perform lifts from LLVM IR to BNIL
- IDA Pro’s Lumina can import public symbols
- BinDiff can import symbols from other binaries

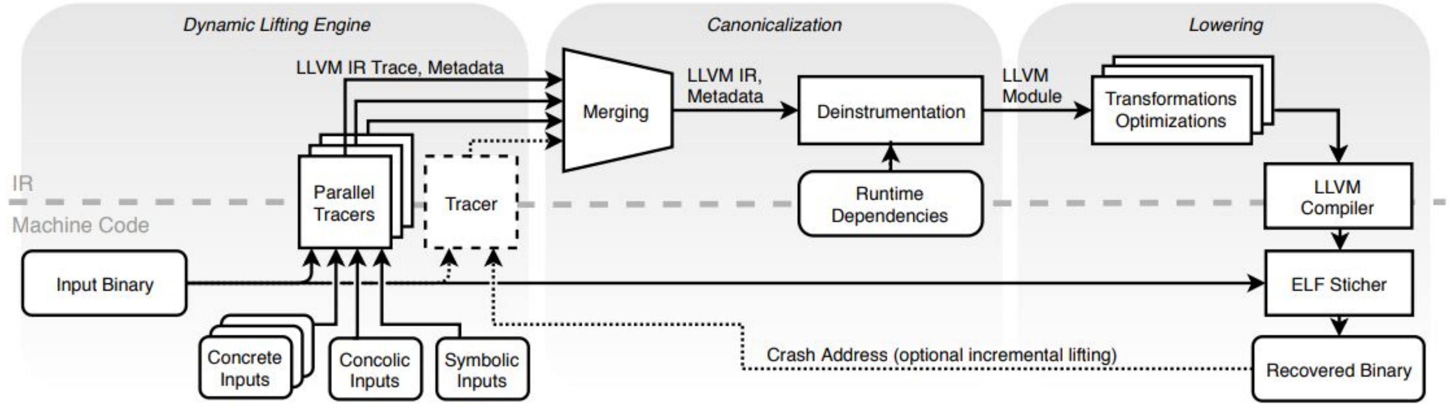


Figure 1. The steps of binary recovery: lifting to compiler IR, transformation on the IR, and lowering back to machine code.

Resources

- Cheat Engine (GitHub)
- PolyHook2 (GitHub)
- GuidedHacking (Website)
- dotPeek (Website)

Setup (CMake + Ninja + MSVC)

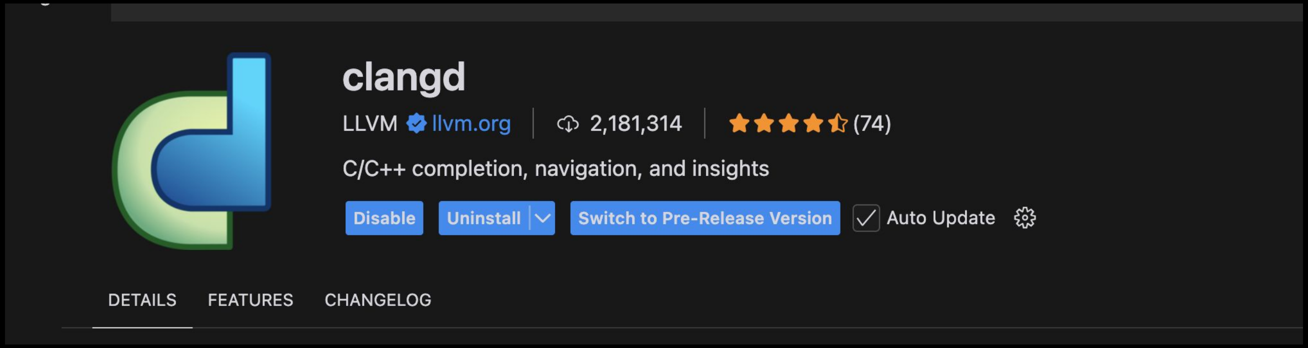
- Download and install CMake.

  -
    ```
    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

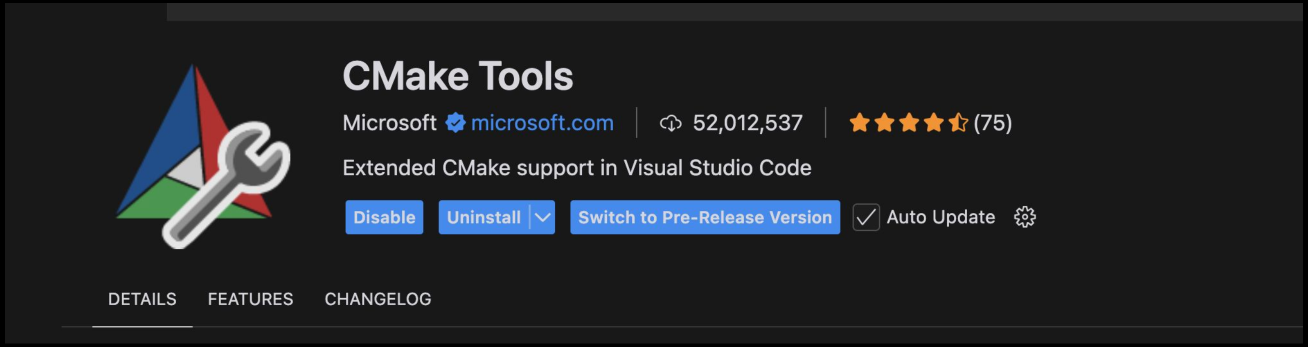


clangd

LLVM llvm.org | 2,181,314 | ★★★★☆ (74)

C/C++ completion, navigation, and insights

☑ Auto Update



CMake Tools

Microsoft microsoft.com | 52,012,537 | ★★★★☆ (75)

Extended CMake support in Visual Studio Code

☑ Auto Update

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

2025-11-09 • This Sunday

- Movie Social
- We have a movie in mind but it's still a secret 🙂

2025-11-13 • Next Thursday

- Rubber Ducky / Bad USB
- Turn physical access into RCE with this one simple trick

2025-11-16 • Next Sunday

- SIGPwny x SIGArch
- Our first SIG x SIG meeting of the semester!

ctf.sigpwny.com

sigpwny{ju57_h00k_7h3_ch3ck5}

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

SIGPwny