



General

SP2026 • 2026-04-05

Game Hacking II

Tyler Mercado

Announcements

- DiceCTF 2025 Quals on 3/28, the Friday after spring break!
 - Starts at 4:00 pm, come in person!



ctf.sigpwny.com

sigpwny{v4ngu4rd_c4n7_c47ch_m3}



Last time on Game Hacking

Game Hacking I covered how cheats work:

- Memory patching & manipulation
- DLL injection (traditional + manual mapping)
- Function hooking (inline + JIT)
- External vs. Internal cheats
- Reversing structs & vtables

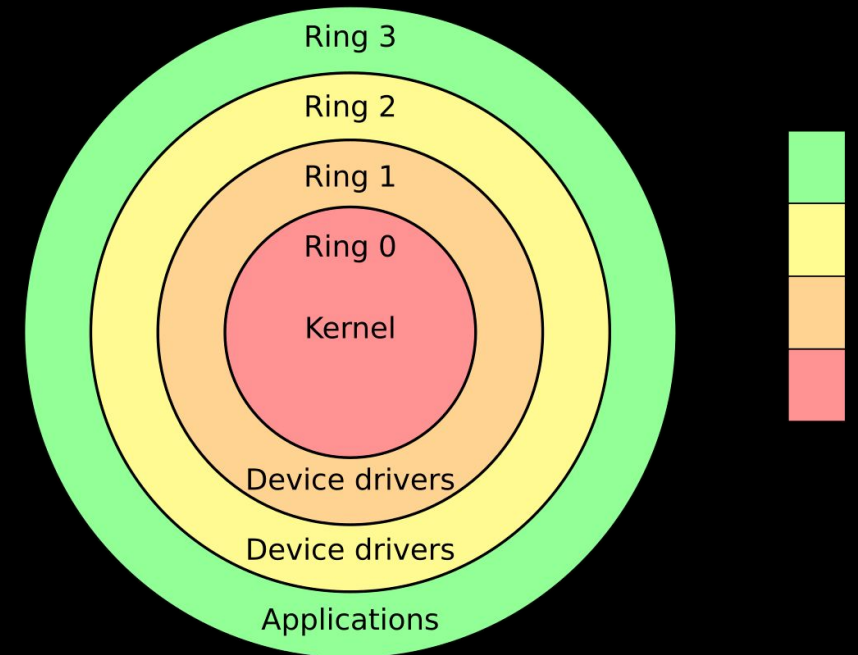
But how do they get caught?



The Privilege Ring Ladder

- Ring 3 (User Mode): Vac
- Ring 0 (Kernel Mode): EAC, BattleEye
- Ring 0 + Boot: Vanguard
- Ring -1/-2: Covered in GH-I (hypervisors, SMM)

We're climbing this ladder today.



VAC: The Classic Anti-Cheat

- Valve Anti-Cheat: purely user-mode (Ring 3)
- Gabe Newell: prefers security through obscurity combined with delayed ban waves
- No BSODs. No driver conflicts. Cross-platform.
- Trade off: can't see kernel-level cheats.



VAC Architecture

Three-tier design:

- Valve Servers -> steam encrypted modules
- steamservice.dll -> orchestrator (32-bit)
- VAC Modules -> run in dedicated threads

Notes:

- Modules are not embedded in steam binaries
- Streamed dynamically at runtime
- Each module owns its own thread
- All VAC code is 32-bit (x86)



The VAC Module System

Modules encrypted with RSA + ICE cipher

- ICE key derived from LCG seeded by `__rdtsc()`
- Custom VLV PE header format
- All modules export `_runfunc@20`

4 known modules types:

- SystemInfo - hardware fingerprinting
- ProcessHandleList - handle enumeration
- ProcessMonitor - process scanning
- DriverInfo - driver inspection



VAC Module Loader

- VAC_LoadModule @ 0x1005bab0
- Two loading paths:
 - Manual mapping (anti-dump)
 - Standard LoadLibrary
- Both resolve “__runfunc@20” via GetProcAddress
- steamservice.dll - 3.6MB, thousands of imports, no obfuscation

```
PE ▾ Linear ▾ High Level IL ▾
void* __stdcall VAC_LoadModule(char* arg1, char arg2)
1005bbf3          var_28 = nullptr
1005bbf3
1005bbfa          int32_t var_24_1 = 0
1005bbfa
1005bc01          sub_100efe50(
1005bc01          (*(data_10475bf8 + 0x1c))(arg1, 0)
1005bc0f
1005bb60      else
1005bb69      int32_t* eax_7 = VAC_ManualMapModule(*(esi + 0x18), 0, 1)
1005bb71      *(esi + 8) = eax_7
1005bb71
1005bb76          if (eax_7 == 0)
1005bb9d          *(esi + 0x10) = 0x16
1005bb76      else
1005bb7e          int32_t eax_8 = VAC_ManualMap_GetExport(eax_7, "_runfunc@20")
1005bb86          *(esi + 0xc) = eax_8
1005bb86
1005bb8b          if (eax_8 == 0)
1005bb91          *(esi + 0x10) = 0x19
1005bb91
1005bce8          if (*(esi + 0xc) == 0)
1005bcd8          sub_1005c0d0(esi)
1005bcf3          void* eax_17
1005bcf3          eax_17.b = 0
1005bcf9          return eax_17
1005bcf9
1005bcfc          bool cond:4_1 = *(esi + 0x18) == 0
1005bd05          int32_t ecx_9 = data_10471f6c
1005bd0b          *(esi + 0x20) = data_10471f68
1005bd0e          *(esi + 0x24) = ecx_9
1005bd0e
1005bd11          if (not(cond:4_1))
1005bd13          sub_100efe50(
1005bd21          (*(data_10475bf8 + 0x1c))* (esi + 0x18), 0)
1005bd24          *(esi + 0x18) = 0
1005bd24
1005bd2c          eax.b = 1
1005bd32          return eax
```



VAC Detection Techniques

- Signature scanning of known cheats in process memory
- Process handle enumeration
- IAT/VMT hook detection
- File hash checking
- ConVr manipulation detection (CS2-specific)
- DNS cache scanning (yes, really - more on this next)



The DNS Cache incident (2014)

- VAC scanned DNS cache to check if players visited known cheat DRM servers.
- Public outcry. Privacy concerns.
- Worked for ~13 days before cheat devs adapted
- Even user-mode scanning has privacy implications

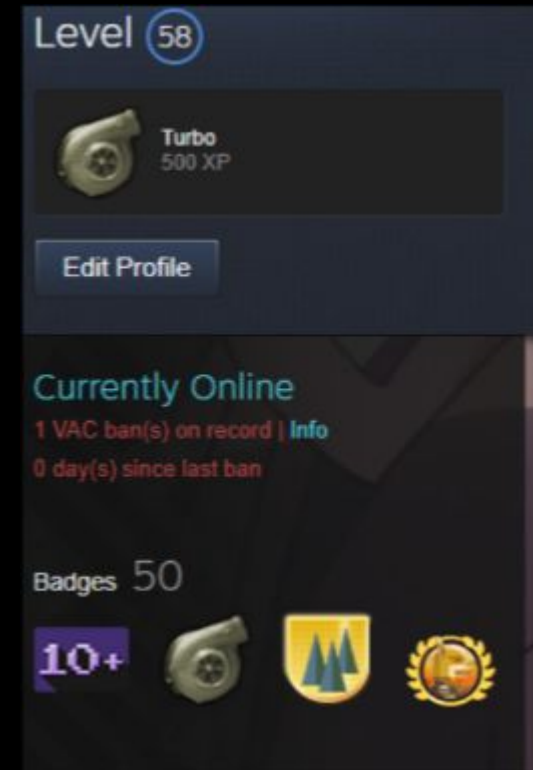


Why VAC is “Weak”

- No kernel access = can't see kernel-level cheats
- Modules can be blocked from loading
- Known bypasses at `steamservice.dll` + `0x590DE`
- Delayed bans (3-4 weeks): cheats play free

But: catching up!

- VACnet - ML-powered detection (2017)
- VAC Live - real-time, cancels matches mid-game (2024)



EAC: The Industry Standard

- EasyAntiCheat: kernel-level (Ring 0)
- Finnish origin -> acquired by Epic Games (2018)
- Free via Epic online Services (200+ games)
- Fortnite, Apex Legends, Rust, Elden Ring, The Finals, ...
- Most widely deployed kernel anti-cheat



EAC Architecture

Three-tier kernel design:

- Ring 0: EasyAntiCheat.sys (kernel driver)
 - Callbacks, memory scanning, driver validation
 - Ring 3: EasyAntiCheat_EOS.exe (SYSTEM)
 - Bridge between driver and EAC servers
 - Ring 3: EasyAntiCheat.dll (injected into game)
 - Game integration, user-mode checks
-
- Bootstrapper downloads fresh client modules from CDN on every launch



Kernel Callbacks: The Big Six

- PsSetCreateProcessNotifyRoutine
 - sees every process creation
- PsSetLoadImageNotifyRoutine
 - sees every DLL/driver load
- PsSetCreateThreadNotifyRoutine
 - sees every thread creation
- ObRegisterCallbacks
 - intercepts handle operations!
- Minifilter
 - filesystem monitoring
- CmRegisterCallbackEX
 - registry monitoring



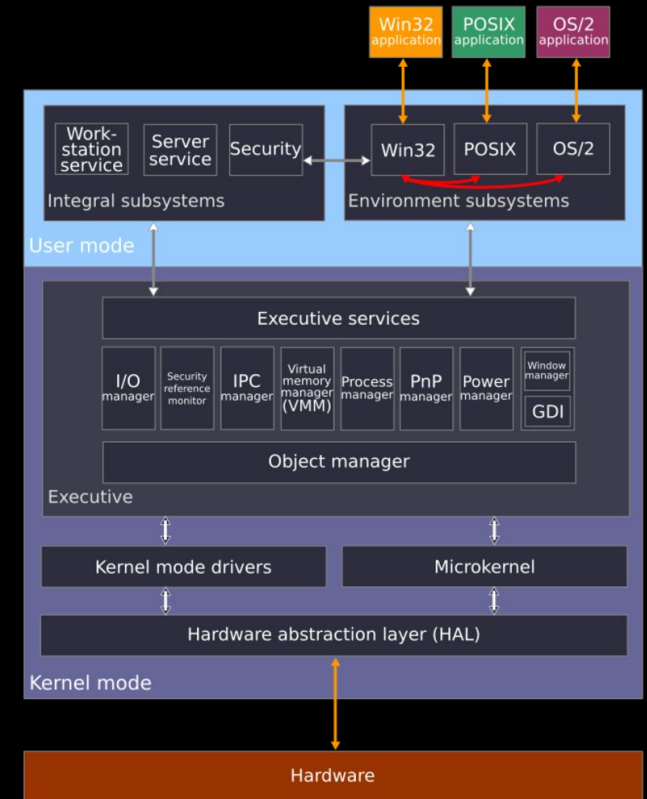
ObRegisterCallbacks: Handle Stripping

The most important anti-cheat mechanism

- Cheat calls `OpenProcess(game)`
- Kernel sends request to `ObRegisterCallbacks`
- EAC's callback strips access rights:
 - `PROCESS_VM_READ` -> removed
 - `PROCESS_VM_WRITE` -> removed
 - `PROCESS_CREATE_THREAD` -> removed
- Cheat gets a useless handle

Result: `ReadProcessMemory` / `WriteProcessMemory` fail.

- External cheats are dead.



EAC Detections

- Memory integrity:
 - Copies own driver to pool, compares every 10-40s
- DLL injection detection:
 - Scans for PE headers + compiler patterns (FF 25 stubs)
- Stack walking via RtlVirtualUnwind:
 - Checks 32 frames for suspicious return addresses
- SSDT integrity via IA32_LSTAR MSR:
 - Verified syscall table hasn't been hooked
- Driver enumeration:
 - PiDDBCacheTable, MmUnloadedDrivers, pool tag 'SldT'



Custom VM Obfuscation

- EAC uses custom virtual machine obfuscation
- Only 2 imports:
FtlRegisterFilter + __chkstk
- 100+ APIs resolved at runtime
- .text section: 1.3MB (real code)
- _3 section: 38MB (VM bytecode)
- No public deobfuscation tooling exists

```
-----  
_3 section started {0x14023f000-0x142742000}  
-----  
14023f000  int64_t j_sub_1402d8ef1(void* arg1 @ r12)  
-----  
14023f000  return sub_1402d8ef1(arg1) __tailcall  
-----  
14023f005  int512_t sub_14023f005(int128_t arg1, int64_t arg2, int128_t arg3, int64_t arg4, int128_t arg5, int128_t arg6, int128_t arg7, int128_t arg8, int128_t arg9, int128_t arg10, int64_t arg11, int64_t arg12, int64_t arg13, int64_t arg14, int128_t arg15, int128_t arg16, int128_t arg17, int64_t arg18, int64_t arg19, int64_t arg20, int128_t arg21, int64_t arg22, int64_t arg23, int64_t arg24, int64_t arg25, int64_t arg26, int128_t arg27, int128_t arg28, int64_t arg29, int128_t arg30, int64_t arg31) __pure  
-----  
14023f005  int512_t zmm5  
14023f005  zmm5.o = arg1  
14023f00f  int512_t zmm9  
14023f00f  zmm9.o = arg3  
14023f01a  int512_t zmm6  
14023f01a  zmm6.o = arg5  
14023f01f  int512_t zmm12  
14023f01f  zmm12.o = arg6  
14023f025  int512_t zmm3  
14023f025  zmm3.o = arg7  
14023f02a  int512_t zmm8  
14023f02a  zmm8.o = arg8  
14023f030  int512_t zmm13  
14023f030  zmm13.o = arg9  
14023f036  int512_t zmm14  
14023f036  zmm14.o = arg10  
14023f05f  int512_t zmm11  
14023f05f  zmm11.o = arg15  
14023f068  int512_t zmm15  
14023f068  zmm15.o = arg16  
14023f071  int512_t zmm4  
14023f071  zmm4.o = arg17  
14023f091  int512_t zmm1  
14023f091  zmm1.o = arg21  
14023f0c1  int512_t zmm7
```



HWID Collection & The ALGS Incident

- EAC collects hardware IDs from 10+ registry paths
- Hardware bans
 - New account + same hardware = banned again
- HWID spoofers vs. EAC adding more fingerprint sources
- ALGS Tournament Incident
 - Pro players got aimbot mid-match on live stream. RCE was in Apex itself, not EAC



BattlEye: The Veteran

- Created by Bastian Suter (2004)
- Kernel Driver since 2014-2015
- PUBG, Escape from Tarkov, R6 Siege, DayZ, Arma3, GTA Online



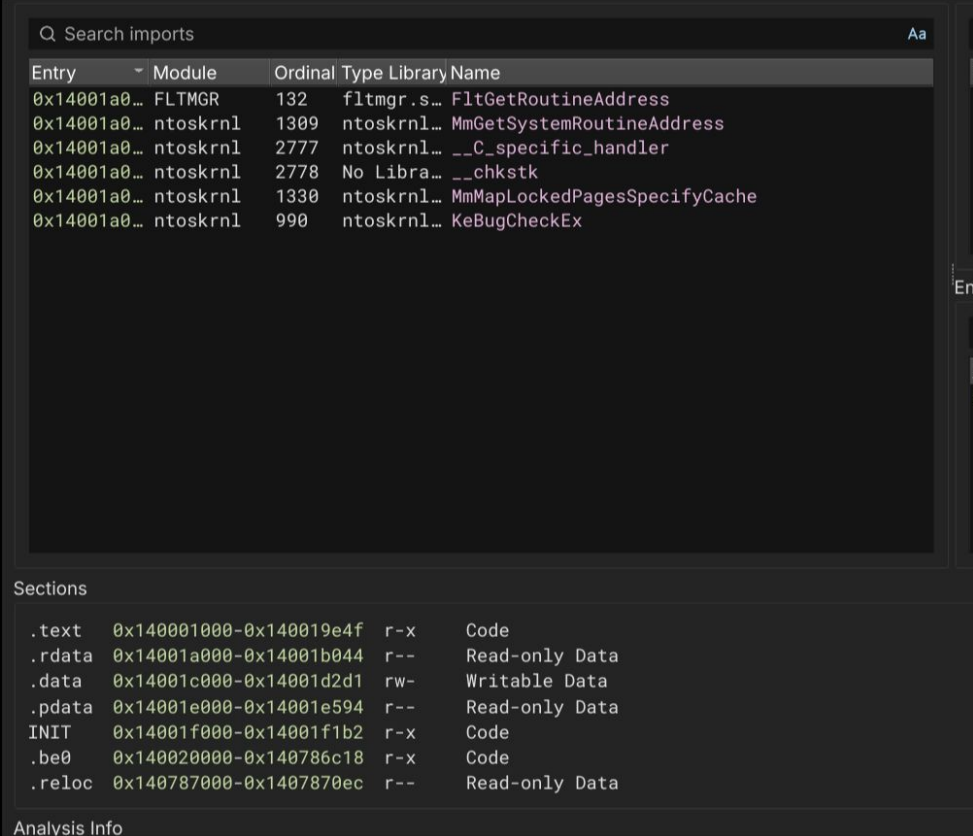
The Four Components

- BEService.exe (NT service, SYSTEM)
 - Orchestrator, server comms, loads driver
- BEDaisy.sys (Ring 0)
 - Preventative: kernel callbacks, blocks access
- BEClient.dll (injected into game)
 - Offensive: the actual scanner
- BEServer (cloud)
 - Telemetry, signatures, bans



BEDaisy.sys Internals

- Same kernel callbacks as EAC, plus:
- 80+ runtime-resolved API imports
 - Not in IAT (uses MmGetSystemRoutineAddress)
- VMProtect obfuscation
 - .text: 100KB, .be0: 7.7MB
- Injects shellcode into lsass.exe and csrss.exe
 - For monitoring from privileged system processes
- Device: \\Device\\BattlEye
- IOCTLs: 0x222000, 0x2D1400, 0x7C088 (HWIDs)



The screenshot shows a debugger window with two panes. The top pane, titled 'Search imports', displays a table of imported functions. The bottom pane, titled 'Sections', displays a table of section names, addresses, permissions, and characteristics.

Entry	Module	Ordinal	Type	Library Name
0x14001a0...	FLTMGR	132		fltMgr.s... FltGetRoutineAddress
0x14001a0...	ntoskrnl	1309		ntoskrnl... MmGetSystemRoutineAddress
0x14001a0...	ntoskrnl	2777		ntoskrnl... __C_specific_handler
0x14001a0...	ntoskrnl	2778	No Libra...	__chkstk
0x14001a0...	ntoskrnl	1330		ntoskrnl... MmMapLockedPagesSpecifyCache
0x14001a0...	ntoskrnl	990		ntoskrnl... KeBugCheckEx

Section Name	Address Range	Permissions	Characteristics
.text	0x140001000-0x140019e4f	r-x	Code
.rdata	0x14001a000-0x14001b044	r--	Read-only Data
.data	0x14001c000-0x14001d2d1	rw-	Writable Data
.pdata	0x14001e000-0x14001e594	r--	Read-only Data
INIT	0x14001f000-0x14001f1b2	r-x	Code
.be0	0x140020000-0x140786c18	r-x	Code
.reloc	0x140787000-0x1407870ec	r--	Read-only Data



BEClient: The Real Scanner

- Memory pattern scanning with report IDs
- Window/process enumeration with blacklists
- Shellcode scanning in game process memory
- Module validation
- Guard page detection
- Screenshot capture and upload
- Server pushes new scan patterns dynamically

- Blacklisted:
 - nvToolsExt64_1.dll -> 0x5A8
 - ws2detour_x96.dll -> 0x5B5



BattlEye: Historical Vulnerabilities

- CVE-2022-27095
 - Privilege escalation vulnerability
- FakeEye
 - Emulated launcher elevation to obtain unstripped process handle
- badeye
 - Handle elevation exploit
- ObRegisterCallbacks disable
 - Technique to remove callback protection (patched)



Vanguard: Boot or Don't Play

- Riot Games: Valorant + League of Legends
- The most invasive mainstream anti-cheat
- Loads at BOOT TIME. Runs 24/7.
- The one that made gamers actually angry.
- Cannot run under Virtual Machine



Why Boot-Time?

- EAC / BattlEye:
 - Load when game launches
 - Cheat driver could already be loaded
- Vanguard:
 - Loads at Windows boot
 - Sees every driver load from the start
- Writes VGKbootstatus.dat to C:\Windows\ during DriverEntry to prove clean boot.
- Disable it = must reboot
 - Chain of trust is broken



The Component Triad

- vgk.sys (Ring 0, boot)
 - Kernel driver, passive monitoring always-on
- vgc.exe (Ring 3, SYSTEM)
 - User-mode service, issues scan commands
- vgtray.exe (Ring 3)
 - System tray icon
- vgk.sys has no network access.
 - All server comms go through vgc.exe
 - Driver only answers binary true/false queries



Kernel Callbacks + Hooks

Standard callbacks (same as EAC/BE) Plus:

- HalPrivateDispatchTable hooks:
 - +0x400 -> SwapContext (context switch monitor)
 - +0x248 -> Syscall hooking
- IAT hooks MmCopyVirtualMemory:
 - Intercepts ALL memory copy operations
- Device: \\Device\\VGK_PLZNOHACK
 - (yes, really)



vgk.sys

- 42MB driver. 22 imports
- .text: 600KB (real code)
- .grfn1: 42MB (PacMan + VMProtect 3)
- .stub0: 291KB (hook stubs)
- _ENTROPY: 64B (randomness seed)
- Pool tag: VGKT
- HAL.dll import in INIT section

```
PE Bytes
1400a391c #####
1400a396b #####
1400a39ba #####
1400a3a09 #####
1400a3a58 #####
1400a3aa7 #####
1400a3af6 #####
1400a3b45 #####
1400a3b94 #####
1400a3be3 #####
1400a3c32 #####
1400a3c81 #####
1400a3cd0 #####
1400a3d1f #####
1400a3d6e #####
1400a3dbd #####
1400a3e0c #####
1400a3e5b #####
1400a3eaa #####
1400a3ef9 #####
1400a3f48 #####
1400a3f97 #####
1400a3fe6 Protected by Riot Vanguard.
1400a4035 #####
1400a4084 #####
1400a40d3 #####
1400a4122 #####
1400a4171 #####
1400a41c0 #####
1400a420f #####
1400a425e #####
1400a42ad #####
1400a42fc #####
1400a434b #####
1400a439a #####
1400a43e9 #####
1400a4438 #####
1400a4487 #####
1400a44d6 #####
1400a4525 #####
```

Section	Address Range	Permissions	Description
text	0x140001000-0x140096680	r-x	Code
	0x140097000-0x140098000	r-x	Code
ENTROPY	0x140098000-0x140098040	r-x	Code
rdata	0x140099000-0x1400a7ee0	r--	Read-only Data
data	0x1400a8000-0x1400bc4bc	rw-	Writable Data
pdata	0x1400bd000-0x1400bf424	r--	Read-only Data
nt	0x1400c0000-0x1400c4000	rw-	Writable Data
edata	0x1400c4000-0x1400c40c6	r--	Read-only Data
INIT	0x1400c5000-0x1400c59d6	rw-	Writable Data
stub0	0x1400c6000-0x14010d120	r-x	Code
reloc	0x14010e000-0x14010e438	r--	Read-only Data
rsrc	0x14010f000-0x140114db8	r--	Read-only Data
grfn1	0x140115000-0x1429442c4	r-x	Code

alvsis Info



Guarded Regions

- Clones CR3/PML4 page table entries to create shadow memory regions
- Game memory mapped into alternate address space only Vangaurd controls
- External cheats using ReadProcessMemory see FAKE DATA (e.g., fake UWorld Pointer)
- Even kernel-level reads get the decoy unless you know the real mapping



Anti-Everything

- Debuggers:
 - Clears DR7 via IPI across ALL cores, RDTSC timing
- Hypervisors:
 - MSR analysis, CPUID checks (refuses to run in VMs)
- DMA:
 - PCIe device scanning, chipset analysis
 - HalpDmaGetIommuInterface import
- Screen capture:
 - Win32k syscall hooks block overlay/capture tools



Self-Protection

- Packman packer + VMProtect 3 virtualization on critical code sections
- stub.dll API hooking with return address verification (calling from unexpected location = flagged)
- Heartbeat:
 - vgk.sys <-> vgc.exe <-> Riot servers
- BCrypt imports for signature verification of all loaded code.



Vanguard Exploits

- Van1338: Handle leak via callback timing race
 - \$6k bug bounty
- Sleeping Bouncer: UEFI vulnerability
 - 4 CVEs across ASUS/Gigabyte/MSI/ASRock
 - Attackers load before Vanguard at BIOS level
- DMA evolution: PCIe cards reading physical memory
 - Vanguard now scans for them



What Actually Works?

- User-mode (VAC):
 - Catches skids, misses kernel cheats
- Kernel (EAC/BE):
 - Stops most cheats, vulnerable to BYOVD + DMA
- Boot (Vanguard):
 - Hardest to bypass, most invasive
- The real trend:
 - Server-side detection + ML
 - Client-side is always an arms race
 - Sadly, you eventually lose



The Privacy Question

- These are rootkits that millions of gamers voluntarily install
 - Vanguard runs 24/7 from boot
 - EAC collects 10+ hardware fingerprints
 - VAC scanned DNS caches
 - BattlEye takes screenshots
-
- Where's the line?



Next Meetings

2026-02-19 • This Thursday

- Password Cracking
- Learn how to store passwords securely and crack them hashed with insecure algorithms!

2026-02-22 • Next Sunday

- Elliptic Curves and Sigs
- Learn some newer forms of cryptography and their benefits.



ctf.sigpwny.com

sigpwny{v4ngu4rd_c4n7_c47ch_m3}

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

