

# Anti-Cheat Detections

- [Virtual Method Tables](#)
- [Control Flow Analysis](#)

# Virtual Method Tables

R5AC is ensuring that VMTP's are pointing within expected bounds, and that read-only VMT related data has not been tampered with.

---

## Pointer Verification

Every time the script calls `sub_1DCDD1`, the virtual method table pointer (VMTP) for the filesystem interface is verified by `VTP_GetFilesystemInterface`. This function performs an integrity check to ensure the interface has not been tampered with or injected. There are numerous other variants of this VMTP related detection, all on different interfaces and virtual method tables, or object instances.

## How the detection works

### 1. VMTP location check

- The function calculates the address of `FilesystemInterfaceVMTP` and checks whether it lies within a valid PE section of the main game module.
- If the pointer is **outside expected sections**, this indicates possible tampering or injection.

### 2. Module and export verification

- It scans the loaded modules for `KERNEL32.DLL` and finds the export `K32GetMappedFileName`.
- This function is used to retrieve the **file path of the module** where the VMTP resides.

### 3. Violation reporting

- If the VMTP is outside a legitimate module section, `R5AC::PushViolation` is called.
- This flags a potential anti-cheat violation.

In more recent builds, R5AC has started verifying that the backing segment of the VMTP's linear address are having a plausible characteristics. it must contain `IMAGE_SCN_CNT_INITIALIZED_DATA/IMAGE_SCN_MEM_READ`

---

### Conclusion:

`VTP_GetFilesystemInterface` is an anti-tamper routine that validates the location of the filesystem VMTP and ensures it points to legitimate code within the game's module. Any deviation triggers a violation report, preventing injected or modified interfaces from being used.

Here is an example of how this detection might look like.

```
“ // Everytime sub_1DCDD1 is called, the check (VTP_GetFileSystemInterface) is executed.
```

```
char __fastcall sub_1DCDD1()
{
    FileSystemInterface = (__int64 *)R5::VTP_GetFileSystemInterface();
    *((_QWORD *)v2 + 16) = FileSystemInterface;
    gpFileSystemInterface[0] = FileSystemInterface;
    return 1;
}

void **R5::VTP_GetFileSystemInterface()
{
    _LIST_ENTRY* lpK32GetMappedFileName = nullptr;
    void* VmtTablePtr = R5::FileSystemInterfaceVMTP;
    void* VmtTablePtr2 = R5::FileSystemInterfaceVMTP;
    __int64 ImageBaseAddress = (__int64)NtCurrentTeb()-
>ProcessEnvironmentBlock->ImageBaseAddress;

    // Plaintext strings (replacing encrypted runtime decoding)
    const char ModuleNameStr[] = "KERNEL32.DLL";
    const char ExportFuncStr[] = "K32GetMappedFileName";
    const char InterfaceStr[] = "VTP_GetFileSystemInterface";

    if (*(WORD*)ImageBaseAddress == 0x5A4D) // Check for 'MZ' PE signature
    {
        _IMAGE_NT_HEADERS64* lpNTH =
        (_IMAGE_NT_HEADERS64*)(ImageBaseAddress + *(int*)(ImageBaseAddress +
        0x3C));
        if (lpNTH->OptionalHeader.Magic == 0x20B) // PE32+
        {
            unsigned int NumberOfSections = lpNTH-
            >FileHeader.NumberOfSections;
            char* lpFirstSectionContentStart = (char*)&lpNTH->OptionalHeader +
            lpNTH->FileHeader.SizeOfOptionalHeader;

            if (NumberOfSections && lpFirstSectionContentStart)
            {
```



```

if (Flink != p_InMemoryOrderModuleList)
{
    while (_wcsicmp(NeedleForSomeModule, (_WCHAR*)Flink[5].Flink) != 0)
    {
        Flink = Flink->Flink;
        if (Flink == p_InMemoryOrderModuleList)
            goto ON_DETECTED;
    }

    _LIST_ENTRY* lpModuleBase = Flink[2].Flink;
    if (lpModuleBase)
    {
LABEL_22:
        if (LOWORD(lpModuleBase->Flink) == 0x5A4D) // 'MZ'
        {
            __int64 lpExportDirectory = (__int64)lpModuleBase +
SHIDWORD(lpModuleBase[3].Blink);
            if (*(WORD*)(lpExportDirectory + 24) == 0x20B) // PE32+
            {
                unsigned int* v27 = (unsigned int*)(lpExportDirectory + 0x88);
                if (!v27)
                    v27 = nullptr;

                if (v27)
                {
                    unsigned int nExportEnumIdx = 0;
                    _DWORD* v29 = (_DWORD*)((char*)lpModuleBase + *v27);
                    __int64 AddressOfFunctions = (__int64)lpModuleBase + (unsigned
int)v29[7];
                    __int64 AddressOfNames = (__int64)lpModuleBase + (unsigned
int)v29[8];
                    unsigned int NumberOfExports = v29[6];
                    __int64 AddressOfNameOrdinals = (__int64)lpModuleBase +
(unsigned int)v29[9];

                    while (nExportEnumIdx < NumberOfExports)
                    {
                        char* lpExportRecordMaybe = (char*)lpModuleBase +
*(unsigned int*)(AddressOfNames + 4i64 * nExportEnumIdx);
                        if (strcmp(lpExportRecordMaybe, ExportFuncStr) == 0)
                        {
                            unsigned int dwFunctionRVA =
*(_DWORD*)(AddressOfFunctions
+ 4i64 * *(unsigned __int16*)(AddressOfNameOrdinals +
2i64 * nExportEnumIdx));

```



# Control Flow Analysis

R5AC is planting control flow enumeration helpers into important game mode.

Just because a module is signed, doesn't mean that R5AC will give it a pass.

It follows a strict whitelist which at the time of this writing, consisted of the following ranges.

r5apex_dx12.exe	(+1000 → +13d2000) ; '.text'
KERNEL32.DLL	(+1000 → +7f4bb) ; '.text'
ntdll.dll	(+1000 → +119f1e) ; '.text'
ntdll.dll	+11a000 → +11a592) ; 'PAGE'
ntdll.dll	(+11b000 → +11b1f9) ; 'RT'
steamnetworkingsockets.dll	(+1000 → +2f63a8) ; '.text'
steam_api64.dll	(+1000 → +243de) ; '.text'
EOSSDK-Win64-Shipping.dll	(+1000 → +d8117c) ; '.text'

Their whitelist is using a relatively primitive structure for describing what they deem a whitelisted memory range:

```
“ struct WHITELISTED_REGION
{
    u64 Begin;
    u64 End;
};
```

In more recent builds of **Apex Legends**, this list is now more complex to interpret and seems to have switched to using some computed hash of whitelisted module bases, and checking for that instead. R5AC only uses RtlCaptureStackBackTrace for examining control flow.

In 2026, R5AC has switched to manual stack tracing via RtlCaptureContext + RtlLookupFunctionEntry + RtlVirtualUnwind.

An example of this instance of detection can be found in C\_ViewRender::GetMostRecentClipTransform.

```
“ __int64 C_ViewRender::GetMostRecentClipTransform(__int64 a1, char
frameIndex)
{
    v152 = frameIndex; // Prime RNG (seems unrelated to stack walker)
    R5AC::Prime = 214013 * R5AC::Prime + 2531011; // Early exit: check stack
walker enablement
    if (!R5AC::IsStackWalkerEnabled)
        return *(_QWORD *)(a1 + 8i64 * v152 + 1155920); __int64
stackFrames[2] = { 0 };
    __int128 stackData[2] = { 0 }; unsigned short numFrames =
R5AC::Imp::RtlCaptureSBT(1, 5, stackData);
    if (!numFrames)
        return *(_QWORD *)(a1 + 8i64 * v152 + 1155920); // Compute simple
hash of the captured stack
    int hashA = 5381, hashB = 0;
    char* p = reinterpret_cast<char*>(stackData);
    for (size_t i = 0; i < 8 * numFrames; ++i)
    {
        unsigned char val = *p++;
        hashA = val + 33 * hashA;
        hashB = val + 65599 * hashB;
    }
    int stackHash = hashA ^ hashB; // Lock to check hash
    MEMORY[0x7FFE8D571760](&R5AC::SoemLock, 0);
    char hasChanged =
R5AC::HashmapHasChangedData((__int64)&R5::HashedCodeExecutionFrames,
&stackHash);
    MEMORY[0x7FFE8D571920](&R5AC::SoemLock); if (hasChanged != -1)
    {
        // If hash is known, return cached transform
        return *(_QWORD *)(a1 + 8i64 * v152 + 1155920);
    }
}
```

```

} // Iterate captured frames
for (unsigned int i = 0; i < numFrames; ++i)
{
    __int64 frameAddr = *reinterpret_cast<__int64*>(stackData + i);
    if (!frameAddr)
        break; // Check if frame is in whitelisted regions
    bool inWhitelist = false;
    for (int j = 0; j < R5AC::WhitelistedSectionCount; ++j)
    {
        _WHITELISTED_SEGMENT_* region = &R5AC::WhitelistedSections[j];
        if (frameAddr >= region->Begin && frameAddr <= region->End)
        {
            inWhitelist = true;
            break;
        }
    }
    if (inWhitelist)
        continue; // If not whitelisted, fetch mapped module and push
violation
    char moduleName[272] = { 0 };
    ((void(__fastcall*)(__int64, __int64*, char*, __int64))0)(-1, frameAddr,
moduleName, 260); if (moduleName[0])
    {
        R5AC::PushViolation(nullptr, 1, frameAddr,
reinterpret_cast<__int64>(moduleName));
        return *(_QWORD*)(a1 + 8i64 * v152 + 1155920);
    }
} // If no violations, store new hash
MEMORY[0x7FFE8D5790A0](&R5AC::SoemLock,
R5AC::WhitelistedSectionCount, numFrames, R5AC::WhitelistedSections);
sub_2345E0(&R5::HashedCodeExecutionFrames, &stackHash);
MEMORY[0x7FFE8D562C70](&R5AC::SoemLock); return *(_QWORD*)(a1 +
8i64 * v152 + 1155920);
}

```

## A little fun Fact

Some cheat developers have actually been using the aforementioned functionality against apex. See, because the API respawn uses for stack walking is resolved into an unchecked pointer in .data, anyone can just swap it out and fake the back trace buffer, or even cheaper: pretend like nothing on the stack was unwindable. As if that wasn't enough, respawn also have decided to making calls by referencing it's value across the game's entire code base.



```

features::on_movement(me);
features::on_trigger(me);
}
}

g.inputs.cmove.tick_end();
}

return 0; /* no traces are available */
}

```

```

“ bool disarm_stack_walker()
{
    size_t num_handled = 0;
    uint64_t needle = gctx->winapis.capture_stack_back_trace;
    if (needle)
    {
        uintptr_t offset = 0;
        auto data = (u8*)gctx->game_data_base;
        auto end = (u8*)(gctx->game_data_base + gctx->game_data_len);

        while (true)
        {
            auto result = stl::search(data + offset, end, (uint8_t *)&needle,
sizeof(uintptr_t));
            if (result != 0)
            {
                #if LOGGING_IS_ENABLED == 1
                LMsg(256, "[R5AC] disarming stack walk at index %i and pointer
%p", num_handled, result);
                #endif
                *reinterpret_cast<void **>(result) = hk_capture_stack_back_trace;
                ++num_handled;
                offset = result - (uintptr_t)data + 1;
            }
            else
            {
                break; // No more occurrences found
            }
        }
    }
}

```

```
g.r5ac_num_stack_walks_disarmed = num_handled;  
  
return (num_handled > 0);  
}
```

Above code was tested on retail apex legends, on both windows and linux. Since the flaw is in R5AC itself, platform mostly does not matter.