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)
        {
            _IMAGE_SECTION_HEADER* dwTableRVA =
(_IMAGE_SECTION_HEADER*)((char*)R5::FilesystemInterfaceVMTP -
ImageBaseAddress);
            _DWORD* sectionHeaders = (_DWORD*)(lpFirstSectionContentStart +
8);
            unsigned int nSectionIdx = 0;

            while (true)
            {
               _IMAGE_SECTION_HEADER* dwSegmentRVA =
(_IMAGE_SECTION_HEADER*)(unsigned int)sectionHeaders[1];
               if (dwSegmentRVA <= dwTableRVA &&
                  (unsigned int)((_DWORD)dwSegmentRVA + *sectionHeaders) >
(unsigned __int64)dwTableRVA)
               {
                  break; // Found containing section
               }

               ++nSectionIdx;
               sectionHeaders += 10;

               if (nSectionIdx >= NumberOfSections)
               {
                  // Outside module section — use default
                  VmtTablePtr = VmtTablePtr2;
                  goto ON_DETECTED_OUTSIDE_INGAME_SECTION;
               }
            }
        }
     }
  }

ON_DETECTED_OUTSIDE_INGAME_SECTION:

  char lpMappedFileName[272] = {0};
  wchar_t NeedleForSomeModule[264] = {0};
  mbstowcs_s(nullptr, NeedleForSomeModule, ModuleNameStr,
sizeof(ModuleNameStr));

  // Search loaded modules
  _LIST_ENTRY* p_InMemoryOrderModuleList = &NtCurrentTeb()-
>ProcessEnvironmentBlock->Ldr->InMemoryOrderModuleList;
  _LIST_ENTRY* Flink = p_InMemoryOrderModuleList->Flink;
```

```
    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));
```

```
                        lpK32GetMappedFileName =
(_LIST_ENTRY*)((char*)lpModuleBase + dwFunctionRVA);
                        break;
                    }
                    ++nExportEnumIdx;
                }
            }
        }
    }
  }

ON_DETECTED:

   // Get the module file name of where this VMTP is residing in.
   if (lpK32GetMappedFileName)
      ((void(__fastcall*)(__int64, void*, char*,
__int64))lpK32GetMappedFileName)(
         -1,
         VmtTablePtr2,
         lpMappedFileName,
         sizeof(lpMappedFileName)
      );

   // Report violation if needed
   R5AC::PushViolation(nullptr, 2, (__int64)VmtTablePtr2,
(__int64)lpMappedFileName);

   return &R5::FilesystemInterfaceVMTP;
}
```

Revision #21
Created 2026-02-12 23:12:41 UTC by Admin
Updated 2026-02-28 16:51:38 UTC by Admin