Habr Login

Blaukovitch
Apr 10 at 12:17

Article summary

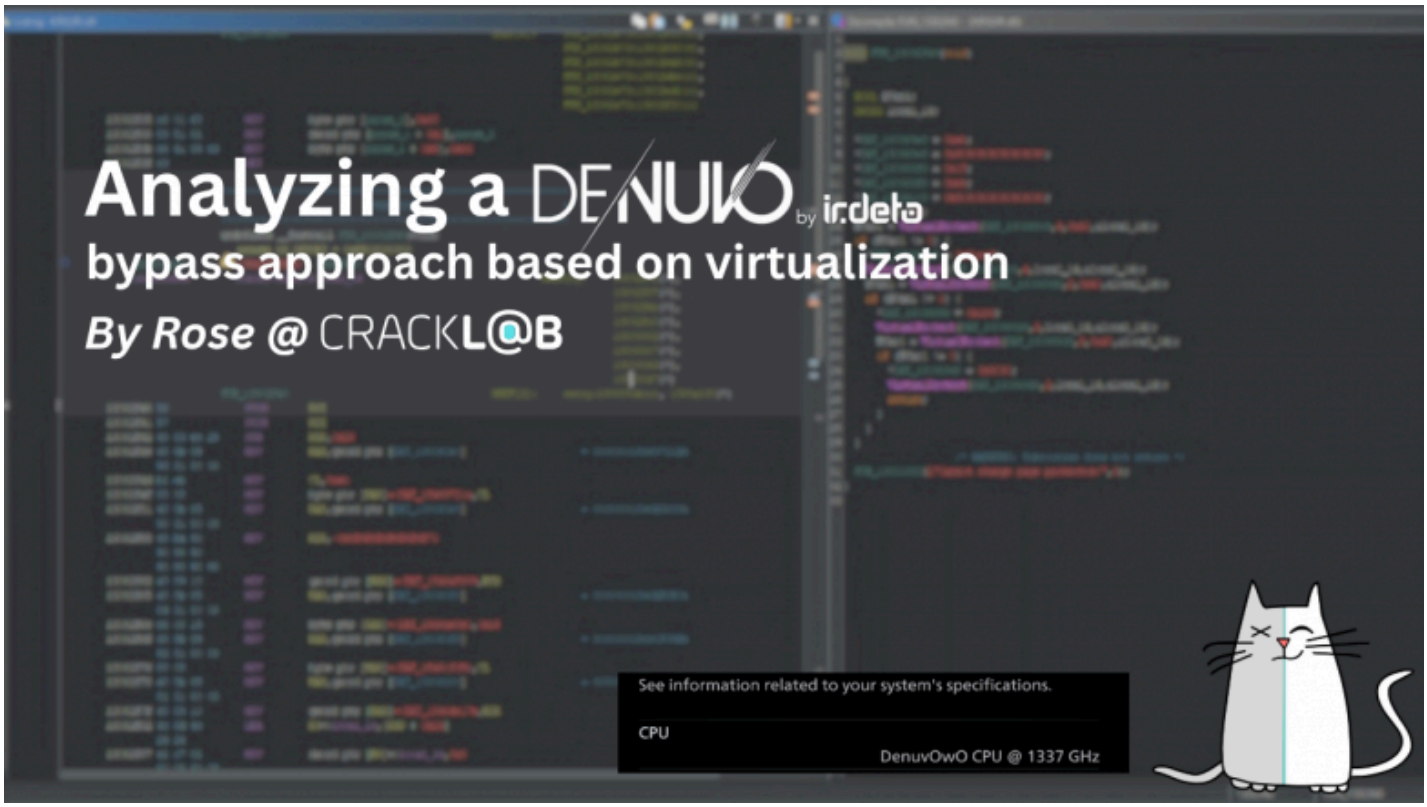
DENUVO Hypervisor. How does it work?

Medium 7 min 14K

Assembler*, Development for Windows*, Information Security*, Game development*, Reverse engineering*

Review

Everything you were afraid to know about the latest method for bypassing DENUVO: how it works behind the scenes and how safe it is for your computer.



CRACKLAB team present

Disclaimer

The author of the article spoke directly with the developers of the DENUVO protection system. It is only because of his location that DENUVO's management is unable to hold him accountable for disclosing sensitive information.



BlaBlaukovitch and DENUVO Sofware Solutions GmbH approves this article.

Analyzing a Denuvo bypass approach based on virtualization.

Hello everyone, since lately there has been a surge of hypervisor-based releases that aim to bypass Denuvo's DRM, I decided to make a write-up describing what this approach looks like and how it works from a technical perspective. This write-up is brought to you fully by `Rose / Natasha 0x80000003`

This article serves more or so to analyze this bypass approach and how to circumvent it on Denuvo's side.

Why this method was chosen

Why was this method chosen in the first place? The main reason is that it is a relatively low-effort way to get a protected title running without doing a full reverse engineering effort against the protection itself. It can be shipped quickly, which matters because early release timing can impact sales in the first week. But this comes at the price of bad performance, instability, security concerns and many other issues.

Introduction

In this write-up, we will analyze the main bypass dll from the recent Resident Evil: Requiem release on cs.rin.ru and go into details on how hypervisors such as **HyperDBG** can be used to spoof hardware values and in effect allow for a **Denuvo DRM** bypass.

Requirements and constraints

For this approach to work, core Windows kernel protections typically need to be weakened so an unsigned or non-standard kernel driver can be loaded. Commonly mentioned requirements in public releases include:

- Disabling Driver Signature Enforcement DSE
- Disabling PatchGuard
- Disabling Windows Defender
- Modification of the Windows Kernel
- At this point disable windows just for the game to run XD

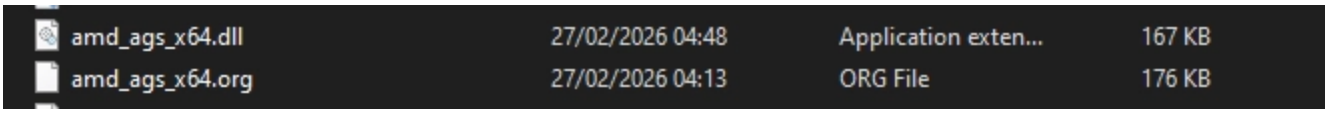
You can use EfiGuard (github)to make your whole system as insecure as possible.

Analyzing Resident Evil: Requiem Hypervisor dll

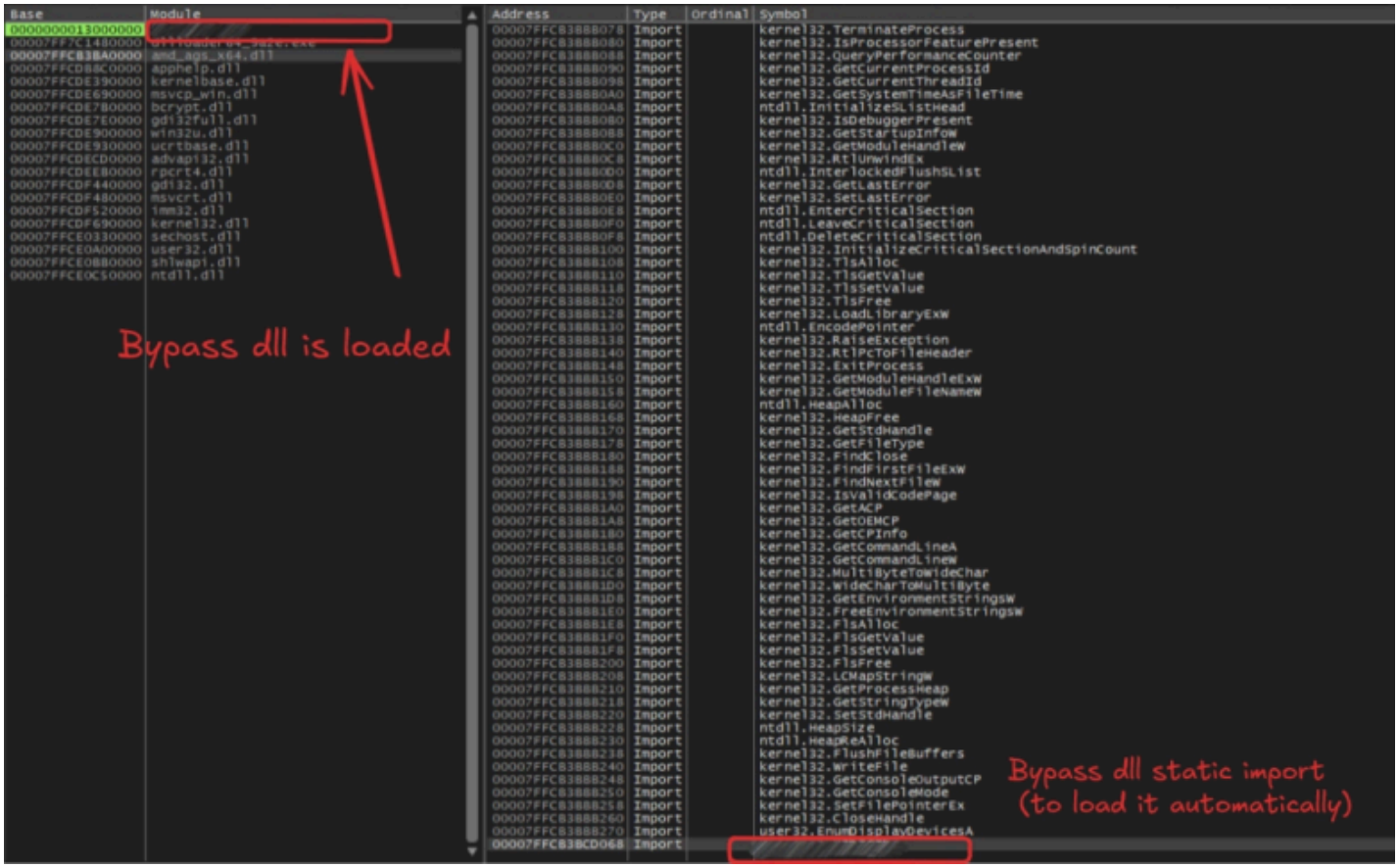
Initial load

In the sample shown here, `amd_ags_x64.dll (dllfiles)` is replaced with a patched proxy dll included in the release. The proxy dll then loads:

- The main bypass dll
- The original legitimate AMD GPU Services SDK dll 4, renamed as amd_ags_x64.org .



Windows Explorer. target DLLs



x64dbg Windows debugger. List of loaded modules.

Since the bypass dll is set as a static PE import, it will be loaded automatically by the Windows loader without needing a separate injector.

Hypervisor back-end selection (Intel / AMD)

Once the main bypass dll is present, it selects the hypervisor back-end based on CPU vendor. The CPUID vendor string is read and a back-end is selected based on the vendor's ID. In the sample:

- `"AuthenticAMD"` → `SimpleSvm.sys`
- `"GenuineIntel"` → `hyperkd.sys`

```
if (((zx.q(temp2_1) ^ 0x444d4163)
        | (0x69746e6568747541 ^ temp1_1.q)) == 0)
    pszMore = u"SimpleSvm.sys"
else
    pszMore = nullptr

    if (((zx.q(temp2_1) ^ 0x6c65746e)
            | (0x49656e69756e6547 ^ temp1_1.q)) == 0)
        pszMore = u"hyperkd.sys"

memset(_Dst: &var_228, _Val: 0, _Size: 0x208)
```

IDA Pro (hex-rays view). Select from back-end driver.

Service creation and driver start

After selecting the driver, a service is created and started for that driver. Because hardware virtualization is typically exclusive, the logic often includes checks to ensure no other hypervisor is already occupying **VT-x** or **AMD-V**. If one is present, it may be stopped so the chosen driver can initialize. Some screenshots of this logic :

```
if (rax_7 == 0 && rax_8 != ERROR_SERVICE_MARKED_FOR_DELETE)
    rcx_20 = u"Cannot delete existing Denuvo service."
else
    CloseServiceHandle(hSCObject: rax_4)
label_13001f5b:
    PWSTR pszMore

    if (((zx.q(temp2_1) ^ 0x444d4163)
            | (0x69746e6568747541 ^ temp1_1.q)) == 0)
        pszMore = u"SimpleSvm.sys"
    else
        pszMore = nullptr

        if (((zx.q(temp2_1) ^ 0x6c65746e)
                | (0x49656e69756e6547 ^ temp1_1.q)) == 0)
            pszMore = u"hyperkd.sys"

    memset(_Dst: &var_228, _Val: 0, _Size: 0x208)

    if (GetModuleFileNameW(hModule: data_13009250,
            lpFilename: &var_228, nSize: 0x104) == 0)
        rcx_20 = u"Cannot get current directory path."
    else
        PathRemoveFileSpecW(pszPath: &var_228)
        PathAppendW(pszPath: &var_228, pszMore)
        int128_t lpLoadOrderGroup
        __builtin_memset(dest: &lpLoadOrderGroup, ch: 0, count: 0x20)
        int128_t lpdwTagId
        SC_HANDLE rax_13 = CreateServiceW(hSCManager: rax_3,
            lpServiceName: u"[redacted]",
            lpDisplayName: u"Denuvo Hypervisor Driver",
            dwDesiredAccess: 0x10,
            dwServiceType: SERVICE_KERNEL_DRIVER,
            dwStartType: SERVICE_DEMAND_START,
            dwErrorControl: SERVICE_ERROR_NORMAL,
            lpBinaryPathName: &var_228, lpLoadOrderGroup, lpdwTagId,
            lpDependencies: &data_130050fe)
```

Start "DENUVO Hypervisor Driver" with call standard Windows API **CreateServiceW**

```
int64_t rbx
int64_t var_20 = rbx

if (sub_13001bfb() != 0)
    enum MESSAGEBOX_RESULT result = MessageBoxW(hWnd: nullptr,
        lpText: u"A hypervisor is already running!\n\nTo continue using the loaded "
    "hypervisor, click 'OK'. Otherwise please ",
        lpCaption: u"Hypervisor detected", uType: 0x41031)

    if (result == IDOK)
        return result

    ExitProcess(uExitCode: 0x1337)
    noreturn

int32_t temp0_2
int32_t temp1_1
int32_t temp2_1
int32_t temp3_1
temp0_2, temp1_1, temp2_1, temp3_1 = __cpuid(0, 0)
int32_t var_230_1 = temp3_1
int32_t rax_2 = sub_13001806(0)
SC_HANDLE rax_3 = OpenSCManagerW(lpMachineName: nullptr,
    lpDatabaseName: u"ServicesActive", dwDesiredAccess: 3)
wchar16 const* const rcx_20
```

Verify loaded service.

Resolving syscall numbers from ntdll

Syscall numbers are taken from `ntdll.dll` and are then used in other functions.

```
56                  push    rsi {__saved_rsi}
57                  push    rdi {__saved_rdi}
4883ec28            sub     rsp, 0x28
488d0dff150000      lea     rcx, [rel data_1300441d]  {"ntdll.dll"}
ff1574260000        call    qword [rel GetModuleHandleA]
4885c0              test    rax, rax
743b                je      0x13002e64

4889c6              mov     rsi, rax
488d15d1150000      lea     rdx, [rel data_13004404]  {"NtQuerySystemInformation"}
488b3d66260000      mov     rdi, qword [rel GetProcAddress]
4889c1              mov     rcx, rax
ffd7                call    rdi
8b4004              mov     eax, dword [rax+0x4]
890510640000        mov     dword [rel data_13009258], eax
488d152e160000      lea     rdx, [rel data_1300447d]  {"NtQueryFullAttributesFile"}
4889f1              mov     rcx, rsi
ffd7                call    rdi
8b4004              mov     eax, dword [rax+0x4]
8905ff630000        mov     dword [rel data_1300925c], eax
4883c428            add     rsp, 0x28
5f                  pop     rdi {__saved_rdi}
5e                  pop     rsi {__saved_rsi}
c3                  retn     {__return_addr}

488d0d73190000      lea     rcx, [rel data_130047de]  {u"Cannot get module handle for ntdll.dll."}
31d2                xor     edx, edx  {0x0}
e811eaffff          call    sub_13001883
```

IDA Pro. Get native syscall numbers.

IAT hooking

IAT hooking is used to redirect selected imports in the game's executable.

Hooked imports are: `ntdll.dll`, `kernel32.dll`, `kernelbase.dll`, `user32.dll`

```
4157              push    r15 {__saved_r15}
4156              push    r14 {__saved_r14}
4155              push    r13 {__saved_r13}
4154              push    r12 {__saved_r12}
56                push    rsi {__saved_rsi}
57                push    rdi {__saved_rdi}
55                push    rbp {__saved_rbp}
53                push    rbx {__saved_rbx}
4883ec48          sub     rsp, 0x48
488b350d2f0000    mov     rsi, qword [rel VirtualAlloc]
ba00102000        mov     edx, 0x201000
31c9              xor     ecx, ecx  {0x0}
41b800301000      mov     r8d, 0x103000
41b940000000      mov     r9d, 0x40
ffd6              call    rsi
4889c7            mov     rdi, rax
ba00102000        mov     edx, 0x201000
31c9              xor     ecx, ecx  {0x0}
41b800301000      mov     r8d, 0x103000
41b940000000      mov     r9d, 0x40
ffd6              call    rsi
4989c5            mov     r13, rax
ba00102000        mov     edx, 0x201000
31c9              xor     ecx, ecx  {0x0}
41b800301000      mov     r8d, 0x103000
41b940000000      mov     r9d, 0x40
ffd6              call    rsi
4889442430        mov     qword [rsp+0x30 {var_58}], rax
ba00102000        mov     edx, 0x201000
31c9              xor     ecx, ecx  {0x0}
41b800301000      mov     r8d, 0x103000
41b940000000      mov     r9d, 0x40
ffd6              call    rsi
4889442428        mov     qword [rsp+0x28 {var_60}], rax
48897c2438        mov     qword [rsp+0x38 {var_50}], rdi
4885ff            test    rdi, rdi
0f8444060000      je      0x13002c69
```

IDA Pro. Hooks from IAT

```
130025c6    int16_t* rax = VirtualAlloc(lpAddress: nullptr, dwSize: 0x201000,
130025c6        flAllocationType: MEM_COMMIT | MEM_RESERVE | 0x100000,
130025c6        flProtect: PAGE_EXECUTE_READWRITE)
130025de    int16_t* rax_1 = VirtualAlloc(lpAddress: nullptr, dwSize: 0x201000,
130025de        flAllocationType: MEM_COMMIT | MEM_RESERVE | 0x100000,
130025de        flProtect: PAGE_EXECUTE_READWRITE)
130025f6    int16_t* rax_2 = VirtualAlloc(lpAddress: nullptr, dwSize: 0x201000,
130025f6        flAllocationType: MEM_COMMIT | MEM_RESERVE | 0x100000,
130025f6        flProtect: PAGE_EXECUTE_READWRITE)
13002610    int16_t* rax_3 = VirtualAlloc(lpAddress: nullptr, dwSize: 0x201000,
13002610        flAllocationType: MEM_COMMIT | MEM_RESERVE | 0x100000,
13002610        flProtect: PAGE_EXECUTE_READWRITE)
1300261f    wchar16 const* const rcx_29
```

IDA Pro. WinAPI VirtualAlloc sets

```
41b800020000        mov     r8d, 0x200
4c89e9              mov     rcx, r13  {data_13009270}
4c89fa              mov     rdx, r15
e858150000          call    memcpy
488b442438          mov     rax, qword [rsp+0x38 {var_50}]
4889059c6a0000      mov     qword [rel data_130092a0], rax
4c893d656a0000      mov     qword [rel data_13009270], r15
4c893d666a0000      mov     qword [rel data_13009278], r15
498b07              mov     rax, qword [r15 {_PEB_LDR_DATA::InLoadOrderModuleList.Flink}]
4c896808            mov     qword [rax+0x8 {_LDR_DATA_TABLE_ENTRY::InLoadOrderLinks.Blink}], r13  {data_13009270}
498b4708            mov     rax, qword [r15+0x8 {_PEB_LDR_DATA::InLoadOrderModuleList.Blink}]
488d88000000ed      lea     rcx, [rax-0x13000000]
4881f901000001      cmp     rcx, 0x1000001
7304                jae     0x13002831

488b4008            mov     rax, qword [rax+0x8 {_LDR_DATA_TABLE_ENTRY::InLoadOrderLinks.Blink}]

4c8928              mov     qword [rax {_LDR_DATA_TABLE_ENTRY::InLoadOrderLinks.Flink}], r13  {data_13009270}
40b701              mov     dil, 0x1
e9be000000          jmp     0x130028fa

41b800020000        mov     r8d, 0x200
488d35276c0000      lea     rsi, [rel data_13009470]
4889f1              mov     rcx, rsi  {data_13009470}
4c89fa              mov     rdx, r15
e8fc140000          call    memcpy
488b442440          mov     rax, qword [rsp+0x40 {var_48}]
488905406c0000      mov     qword [rel data_130094a0], rax
4c893d096c0000      mov     qword [rel data_13009470], r15
4c893d0a6c0000      mov     qword [rel data_13009478], r15
498b07              mov     rax, qword [r15 {_PEB_LDR_DATA::InLoadOrderModuleList.Flink}]
48897008            mov     qword [rax+0x8 {_LDR_DATA_TABLE_ENTRY::InLoadOrderLinks.Blink}], rsi  {data_13009470}
498b4708            mov     rax, qword [r15+0x8 {_PEB_LDR_DATA::InLoadOrderModuleList.Blink}]
488d88000000ed      lea     rcx, [rax-0x13000000]
4881f901000001      cmp     rcx, 0x1000001
7304                jae     0x1300288d
```

IDA Pro. IAT hooking. Continue.

```
488b4018          mov     rax, qword [rax+0x18 {_PEB::Ldr}]
4c8d7810          lea     r15, [rax+0x10] {_PEB_LDR_DATA::InLoadOrderModuleList}
488b6820          mov     rbp, qword [rax+0x20 {_PEB_LDR_DATA::InMemoryOrderModuleList.Flink}]
488b5830          mov     rbx, qword [rax+0x30 {_PEB_LDR_DATA::InInitializationOrderModuleList.Flink}]
31ff              xor     edi, edi  {0x0}
4c8d2d6b6b0000    lea     r13, [rel data_13009270]
31f6              xor     esi, esi  {0x0}
4531f6            xor     r14d, r14d  {0x0}
4531e4            xor     r12d, r12d  {0x0}

4d8b3f            mov     r15, qword [r15 {_PEB_LDR_DATA::InLoadOrderModuleList.Flink}]
450fb74758        movzx   r8d, word [r15+0x58]
498b4f60          mov     rcx, qword [r15+0x60]
488d15721e0000    lea     rdx, [rel data_13004592]  {u"ntdll.dll"}
e81b160000        call    _wcsnicmp
85c0              test    eax, eax
0f84ba000000      je      0x130027e7

450fb74758        movzx   r8d, word [r15+0x58]
498b4f60          mov     rcx, qword [r15+0x60]
488d159d1e0000    lea     rdx, [rel data_130045da]  {u"kernel32.dll"}
e8fe150000        call    _wcsnicmp
85c0              test    eax, eax
0f84f2000000      je      0x1300283c

450fb74758        movzx   r8d, word [r15+0x58]
498b4f60          mov     rcx, qword [r15+0x60]
488d154c1e0000    lea     rdx, [rel data_130045a6]  {u"kernelbase.dll"}
e8e1150000        call    _wcsnicmp
85c0              test    eax, eax
0f8435010000      je      0x1300289c

450fb74758        movzx   r8d, word [r15+0x58]
498b4f60          mov     rcx, qword [r15+0x60]
488d154d1e0000    lea     rdx, [rel data_130045c4]  {u"user32.dll"}
e8c4150000        call    _wcsnicmp
85c0              test    eax, eax
0f8576010000      jne     0x130028fa
```

IDA Pro. Select from hooking dynamic library's.

With the IAT patched, calls that would normally go directly to these modules are routed through the bypass dll's handlers.

License file / Token handling

Inside the main bypass dll, there are 2 different pre-generated tokens that are assigned depending on the host's CPU architecture and the corresponding hypervisor (as shown below). These tokens are written to a `.bin` file on first launch and are then used by the hypervisor as Denuvo tokens.

```
if (GetModuleFileNameW(hModule: data_13009250, lpFilename: &var_218, nSize: 0x104)
        == 0)
    rcx_14 = u"Cannot get current directory path."
else
    PathRemoveFileSpecW(pszPath: &var_218)
    PathAppendW(pszPath: &var_218, pszMore: u"[redacted]")
    HANDLE rax_2 = CreateFileW(lpFileName: &var_218, dwDesiredAccess: 0xc0000000,
        dwShareMode: FILE_SHARE_NONE, lpSecurityAttributes: nullptr,
        dwCreationDisposition: CREATE_ALWAYS,
        dwFlagsAndAttributes: FILE_ATTRIBUTE_NORMAL, hTemplateFile: nullptr)

    if (rax_2 == -1)
        rcx_14 = u"Cannot create Denuvo license file [redacted]."
    else
        uint8_t* lpBuffer

        if (((zx.q(temp2) ^ 0x444d4163) | (0x69746e6568747541 ^ temp1.q)) == 0)
            lpBuffer = "8c10fd301c7b40a8c40898a1b4c9365a036faad298bf7150c44998a1b5c83857003eaddc87ef285ec91dc1aebbd175070f63fd9a899d7f
        else
            lpBuffer = nullptr

            if (((zx.q(temp2) ^ 0x6c65746e) | (0x49656e69756e6547 ^ temp1.q)) == 0)
                lpBuffer = "8c438ac29c793362c4202686b4278d6903bdaa4b98fe5c41c4252686b5230e5d00f86b7387bf0a79c97470babb475d1c0f8d7f6a89

        uint32_t numberOfBytesWritten = 0

        if (WriteFile(hFile: rax_2, lpBuffer, nNumberOfBytesToWrite: 0x1867,
                lpNumberOfBytesWritten: &numberOfBytesWritten,
                lpOverlapped: nullptr) != 0)
            return CloseHandle(hObject: rax_2)

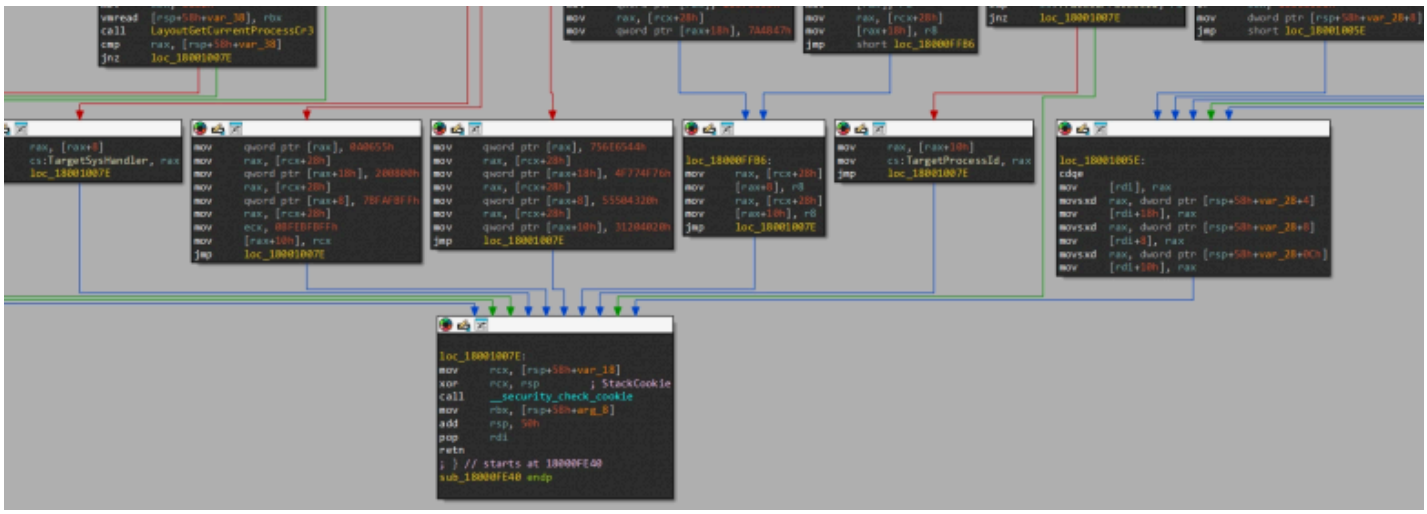
        rcx_14 = u"Cannot write to Denuvo license file [redacted]."

sub_13001883(rcx_14, NO_ERROR)
noreturn
```

IDA Pro. Make DENUVO license file (with secret tokens).

Small Notes

- Detanup01's GBE fork was used to load the game and also emulate steam API services
- HyperEvade was used to also further hide the hypervisor (I will talk about this more later)

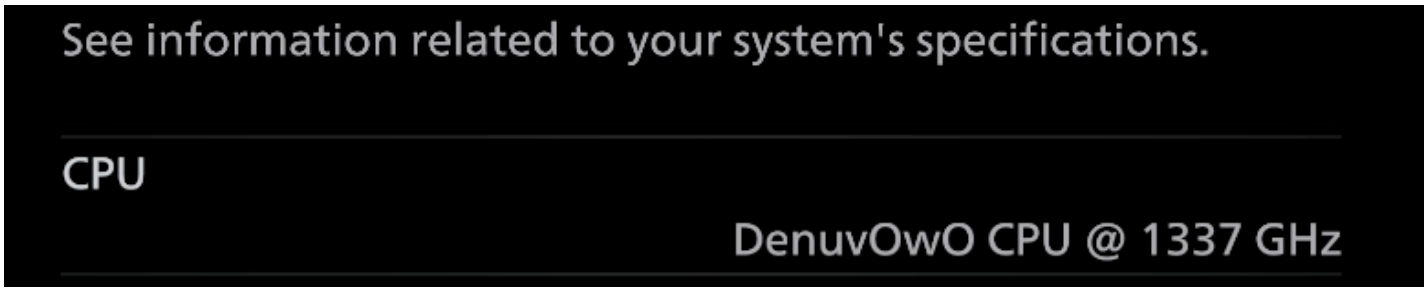


IDA Pro. Emulation logic.

The **CPU brand** string in the sample I analyzed is set to

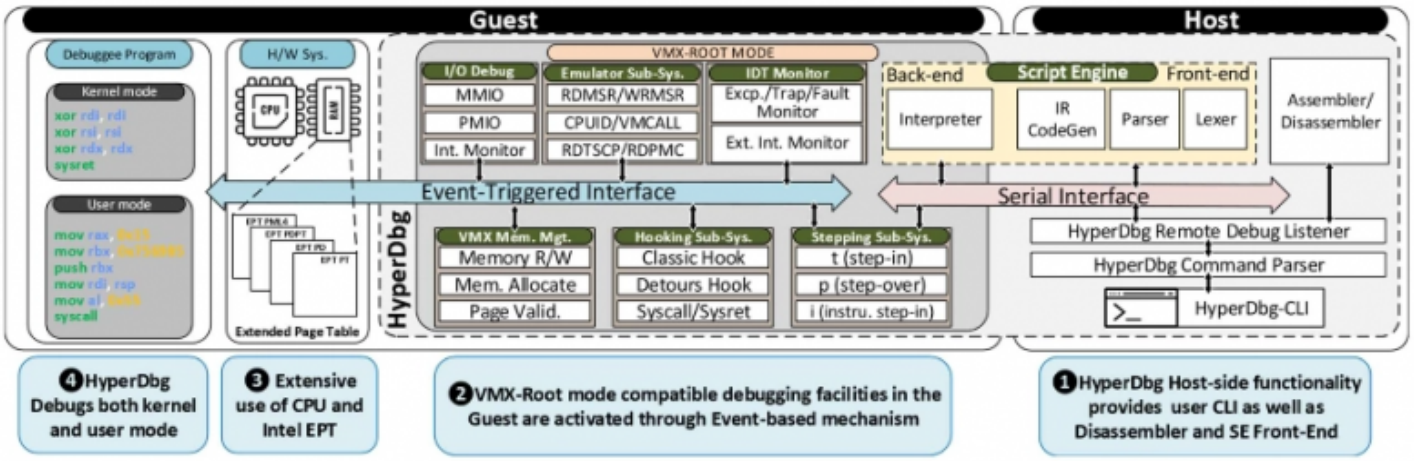
`DenuvOWO CPU @ 1337 GHz`. XD

(I only included this to showcase that some info doesn't have to be fully real and that environments can be modified to allow for custom values to be used in the hypervisor)



`DenuvOWO CPU @ 1337 GHz`

Let's dive more into the details!



Hypervisor workflow architecture.

HyperDBG is a very advanced debugger and thus I won't analyze every part of it and more or so focus on the main part of hardware spoofing and how it can be done in the context of Denuvo. All the commands here can be used with **HyperDBG's** scripting system for testing. A custom implementation can be done later to optimize this process for an actual bypass release.

Hardware Spoofing

`SEE at FULL article at official CRACKLAB site:`

https://cracklab.team/index.php?threads/1601/

Important Note

Denuvo checks for many other values in `KUSER_SHARED_DATA` , `WinAPI` and many other checks pre and post `OEP`. Want to find them ? Suit yourself and analyze the protection like a normal human being instead of trying to find “vulnerabilities” to bypass it. Though I’d like to say that all of Denuvo DRM checks can be bypassed this way (**As of February 2026**).

VMX-Hypervisor

There are also many MANY hypervisors that are available publicly on github, check out **Marius’** hypervisors as a base:



VMX-Hypervisor

These were used for some pretty cool shenanigans and are a very good base if you want to study hypervisors and maybe create your own

Finishing up

After we’re done, we can go ahead and generate the Denuvo token matching our **CPU** (if you have the CPU) or simply generate a new Denuvo token for these specific values in a **VM** or some kind of **Hypervisor** (figure it out yourself, there’s a million approaches for this). Though I would recommend that you generate it inside an environment with these values and then use the same values from the environment to avoid any problems with the bypass.



We love you Blaukovitch <3

Detecting Hypervisor bypass attempts

The various techniques to detect hypervisor bypass attempts and patching such methods. To be honest, there isn't much to do on user mode level (that I can talk about here / publicly), thus a custom driver would need to be implemented to add most of these checks. In this section I will discuss general techniques to detect hypervisors, I will later on explain why some stuff won't work in the case of Denuvo.

EPT hook detection

Enforcing DSE on

Timing Checks

`SEE at FULL article at official CRACKLAB site:`

https://cracklab.team/index.php?threads/1601/

Explaining the current Denuvo situation

There are many checks that can be used to detect hypervisors, but the issue is most of them can be spoofed one way or another. Checks that must be used have to be very reliable and very hard to spoof. An obvious check is already implemented in the bypass dll itself.

```
13001bfb    int64_t sub_13001bfb()

13001c05        int32_t temp0_1
13001c05        int32_t temp1
13001c05        int32_t temp2
13001c05        int32_t temp3
13001c05        temp0_1, temp1, temp2, temp3 = __cpuid(0x40000001, 0)
13001c0f        int32_t result
13001c0f        result.b = temp0_1 == 0x30237648
13001c12        return result
```

DENUVO CPUID check code example.

Here the dll checks for the **Microsoft Hyper-V** interface signature using `cpuid`, and Denuvo could theoretically just start checking every **`cpuid`** value and using it for token generation, but that would just end up with another cat and mouse game where Denuvo adds more checks and hypervisor releases keep adding more spoofs which is not reliable at all as a long term solution for hypervisors.

Timing checks are unreliable and thus those also cannot be used.

The solution that Denuvo will use needs to be:

- Reliable
- Not so easy to bypass
- Doesn’t affect performance
- Doesn’t rely on a kernel driver
- Doesn’t ruin the experience for people that actually bought the game

Overkill Approaches

`SEE at FULL article at official CRACKLAB site:`

https://cracklab.team/index.php?threads/1601/

Risks

“Bu-but what risks are there ?” Are we serious ?

Does executing kernel level code on your pc from a random guy on the internet sound like a good idea to you ?

If they (the people releasing these “cracks”) really wanted to, they can obfuscate the driver with the excuse of “*hiding their techniques from Denuvo”* and ship it with whatever they want. They already tell you to disable all security settings so it shouldn’t be hard for them to execute anything they want on your computers anyway. You can go ahead and keep trusting retards but don’t be surprised if greed blinds them one day and they decide to release a “crack” with info stealer code attached.

”But that’s the case with every software on the internet!”

Well, most software on the internet doesn’t require you to run it with higher permissions than your own administrator account, making this a special case scenario. Sadly this isn’t like the old scene where if your crack has the slightest bit of an issue it would get nuked to smithereens. These people couldn’t care less about their reputation because everyone can release these types of cracks, it’s just a time competition on who will release it first.

Overall, it’s really hard trying to convince anyone in the scene that it’s a bad idea to use any of these hypervisor releases, because frankly, they really don’t care. They don’t care if their data and personal information ends up on some random Chinese forum as long as they can enjoy their little game and shitpost about “finally beating Denuvo”.

Summary

When the game is ran, a custom bypass dll is loaded to attach a hypervisor layer that could be used to spoof various values that are checked by Denuvo, this implementation allows the user to run the game under an already generated token configured to the hypervisor environment.

Closing statement

From a purely technical standpoint, this is a pretty cool approach, but **I personally don’t think any crack/bypass release** of any game should contain code that requires kernel-level access and attaches itself to very low level layers just to operate, especially at the cost of performance and security concerns. Instead of wasting your time with this, study the protection and try to analyze it yourself from a research perspective.



On the right is the CTO of Denuvo, On the left is his beautiful girlfriend. (Sarcasm)

Contact

`@unpackme` on Discord (official site), no other contact info will be provided unless personally given.

Big thanks to:

Marius - Analysis of Denuvo and helping with this article. (https://github.com/Nitr0G)

********* - Releasing the bypass for Resident Evil : Requiem

Momo5502 - EPT hooks detection analysis (https://github.com/momo5502/ept-hook-detection)

SinaKarvandi - Research on Hypervisors and Development of HyperDBG SinaKarvandi - Overview) (https://github.com/SinaKarvandi/Hypervisor-From-Scratch)

if you need more about Sony DADC AG SecuROM:

https://cracklab.team/PAunlock

Tags: DENUVO, hypervisors, research, cpuid, cracklab, SecuROM, ring0, resident evil, drm

Hubs: Assembler, Development for Windows, Information Security, Game development, Reverse engineering

+3 5 0



72
Karma

Max S. @Blaukovitch

80_PA SecuROM keygen (Sony DADC AG)

Follow

Leave a comment

Articles

TOP OF THE DAY SIMILAR ARTICLES

Show the best of all time

Your account

Log in

Sign up

Sections

Articles

News

Hubs

Authors

Sandbox

Information

How it works

For authors

For companies

Documents

Agreement

Confidential

Services

Corporate blogs

Advertising

Native advertising

Education programs

Startups

Language settings

Support

© 2006–2026, Habr

On this platform we use recommendation systems