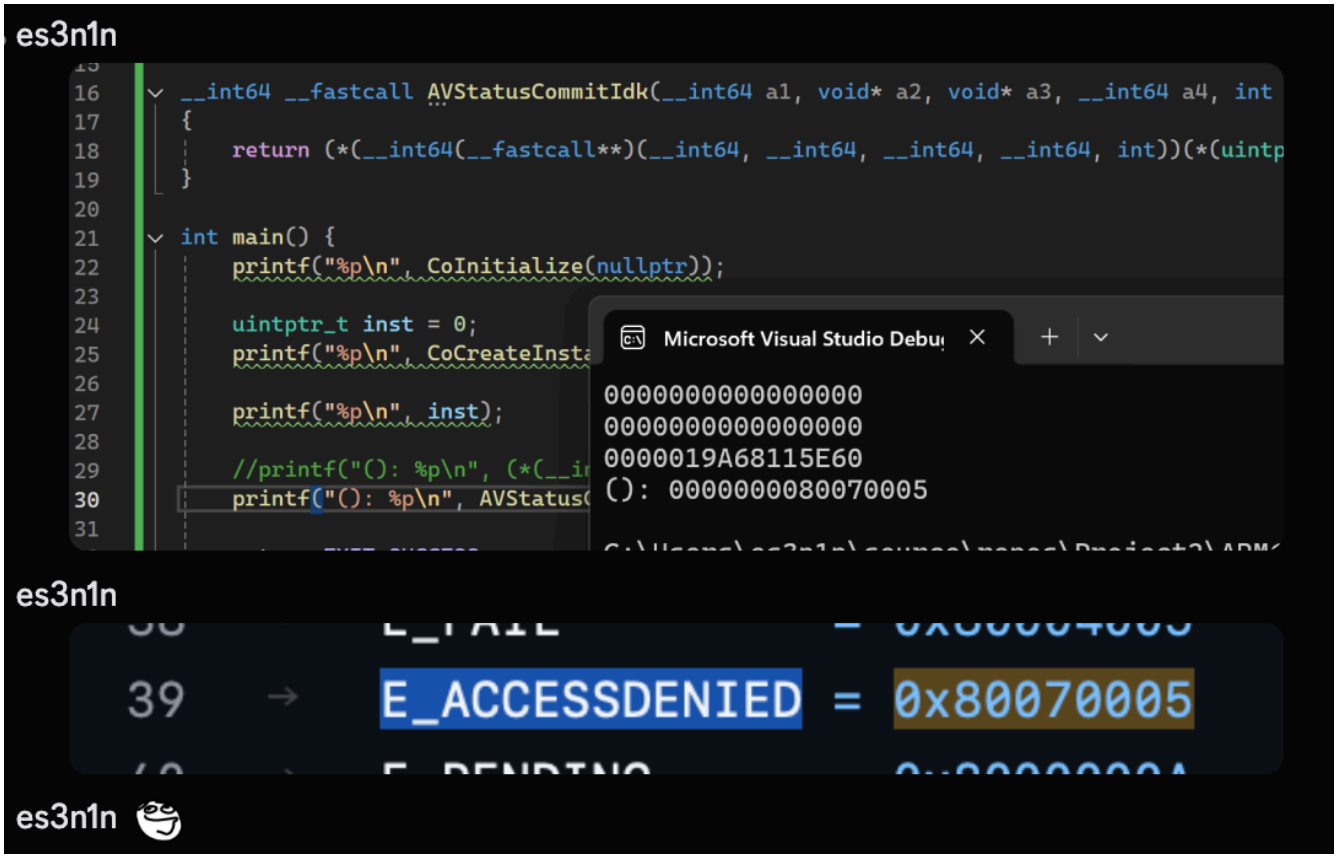
es3n1n's Blog

blog.es3n1n.eu/posts/how-i-ruined-my-vacation

May 8, 2025



Reverse engineering, programming and some other boring stuff.

In this post I will briefly describe the journey I went through while implementing defendnot, a tool that disables Windows Defender by using the Windows Security Center (WSC) service API directly.

Even though this is most likely not what you expected to see here, but rather than going into full technical details on how everything works, I will describe what rabbitholes I went through and how painful everything was due to my ✨special✨ environment.

Beware, most likely this post will be too informal unlike the previous posts of mine, I am pretty sure that all the other posts with `irl` tag will be written in a style like this. If you are looking for a more detailed technical description of how everything works, a writeup like this will be released a bit later by someone else and I will link it here.

A one-year step back

Almost exactly one year ago I released a tool no-defender, a project that was disabling windows defender using the special windows api made for antiviruses to let the system know that there is an another antivirus so there is no need to run defender scans.

The part of the system that manages all this mess is called Windows Security Center - WSC for short. The way how my project worked is that it was using a thirdparty code from some already existing antivirus and forced that av to register the antivirus in WSC.

Then, after a few weeks after the release, the project blew up quite a bit and gained ~1.5k stars, after that the developers of the antivirus I was using filed a DMCA takedown request and I didn't really want to do anything with that so just erased everything and called it a day.

How it started

Currently, even while writing this article, I am sitting in an airbnb we rented in Seoul. After numerous trips to other parts of the planet for CTFs *(CTF is short for capture the flag)* and stuff, me and a friend of mine decided that we want to visit Seoul and arrived a few months after that.

My current main machine for non-ctf things is an M4Pro MacBook, and usually, when I am going for a CTF I bring an another x86 laptop with me to do some extensive reverse engineering/pwn stuff as it is usually built for the x86 cpus. Emulation would kind of work for this task but it is pretty painful so I just use an another laptop for all the x86 stuff.

And, as you might have guessed, for this trip I did not bring that x86 laptop with me, but I did bring my macbook with me to do some other development stuff in my free-free time. So, I did not have any x86 machine with me to do the x86 reversing.

And, on May 4th, after a few days spent in South Korea meeting my favorite South Korean CTF friends and drinking alcohol with them, I received a message from MrBruh where they said that they were looking at no-defender and were looking into whether it would be possible to create a "clean" implementation of my project without using any AVs.

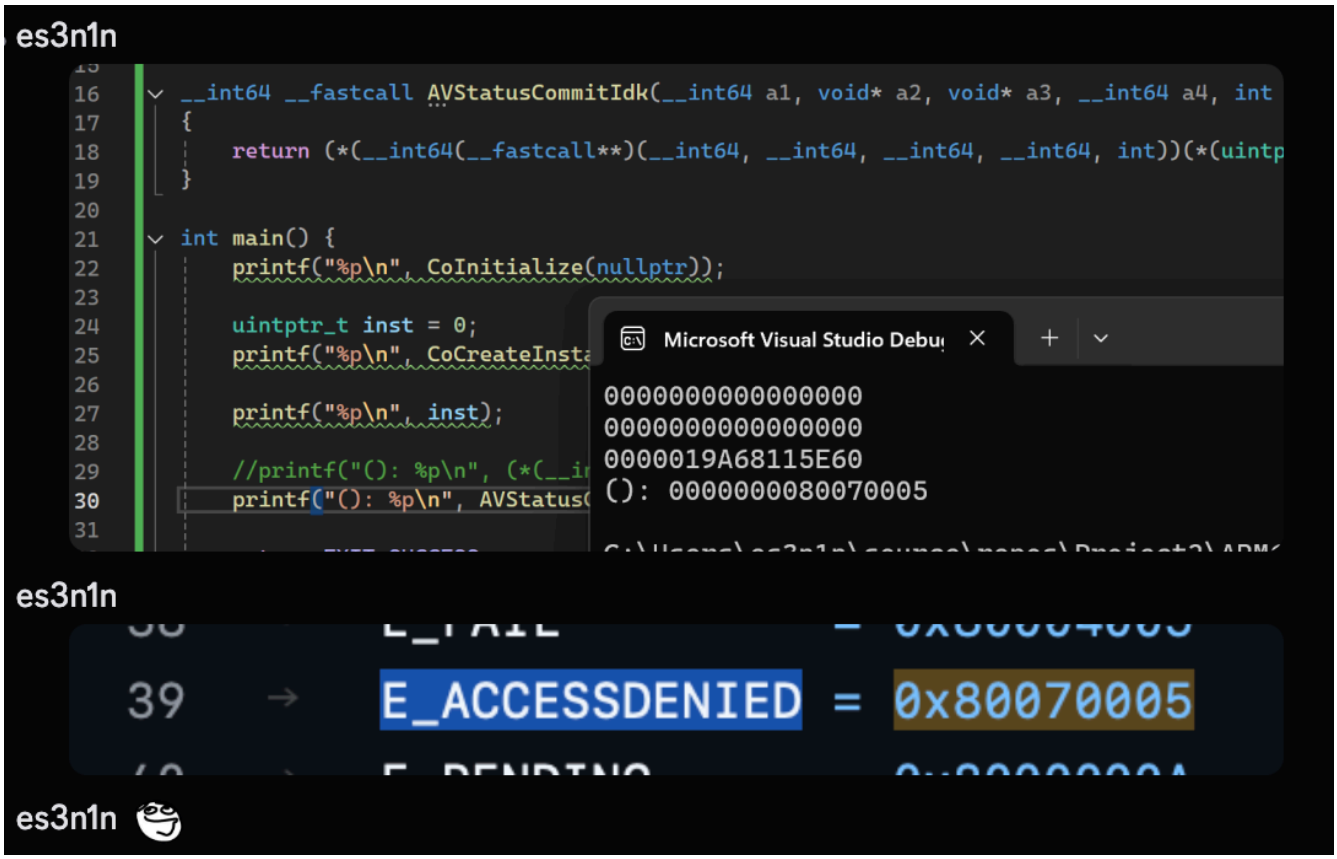
Initial research (Day 1)

I am having *some* troubles with my sleep schedule and I woke up a bit earlier than my friends so I decided to take a look at this while I am waiting for my friends to wake up.

MrBruh provided me the latest binaries of wsc because I was too lazy to spin up my parallels vm to get the binaries and I started looking into what we got.

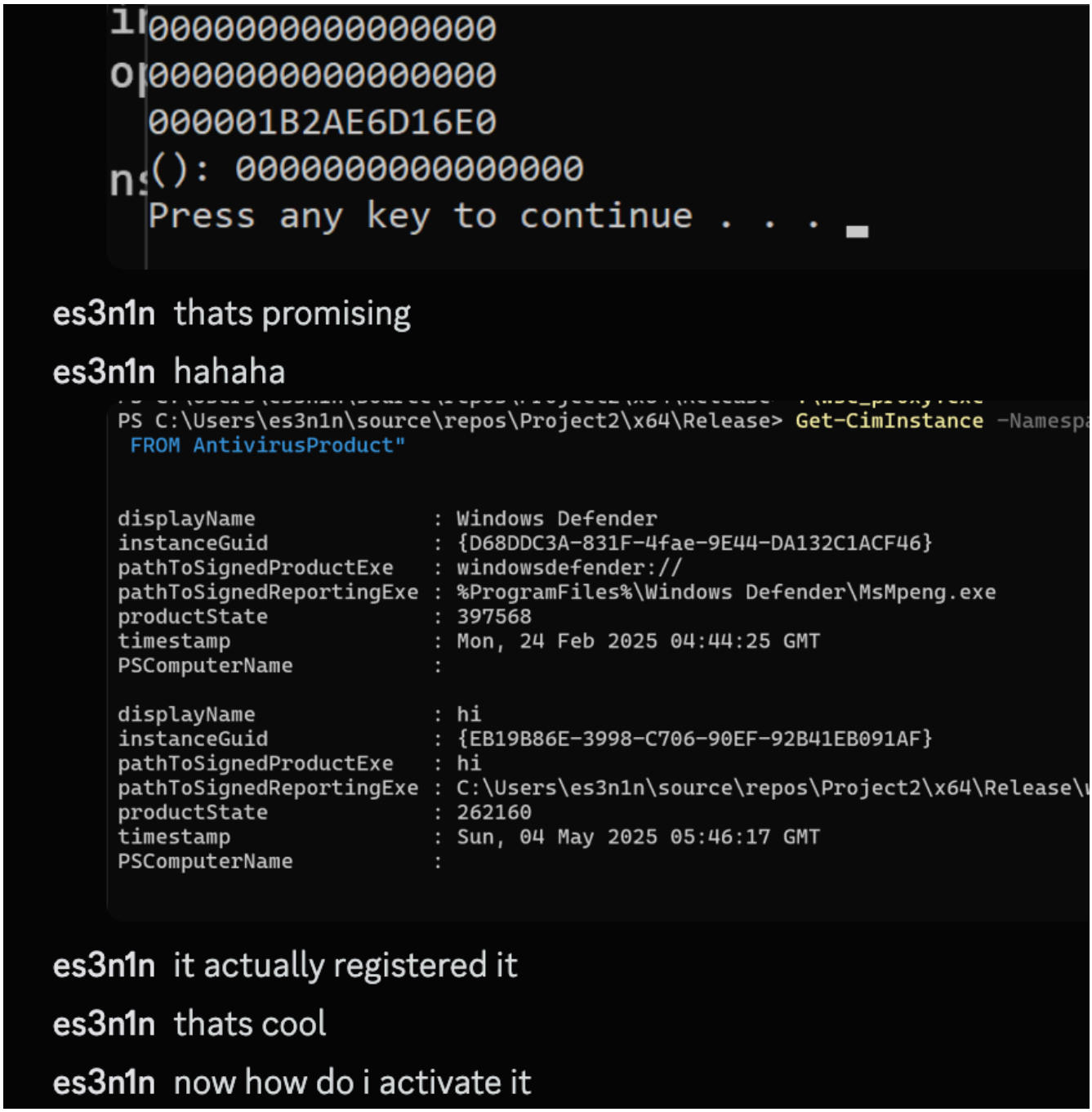
As a reference implementation, I took the WSC registration implementation made by the same AV I was using a year ago. I was somewhat familiar with the internals of their thing and it was a great call.

Essentially, WSC has a COM API that all antiviruses are using, so I quickly rebuilt everything that AV was doing with it in ~1hr, booted an arm64 windows in parallels and tested the thing. I was greeted with an access denied error.

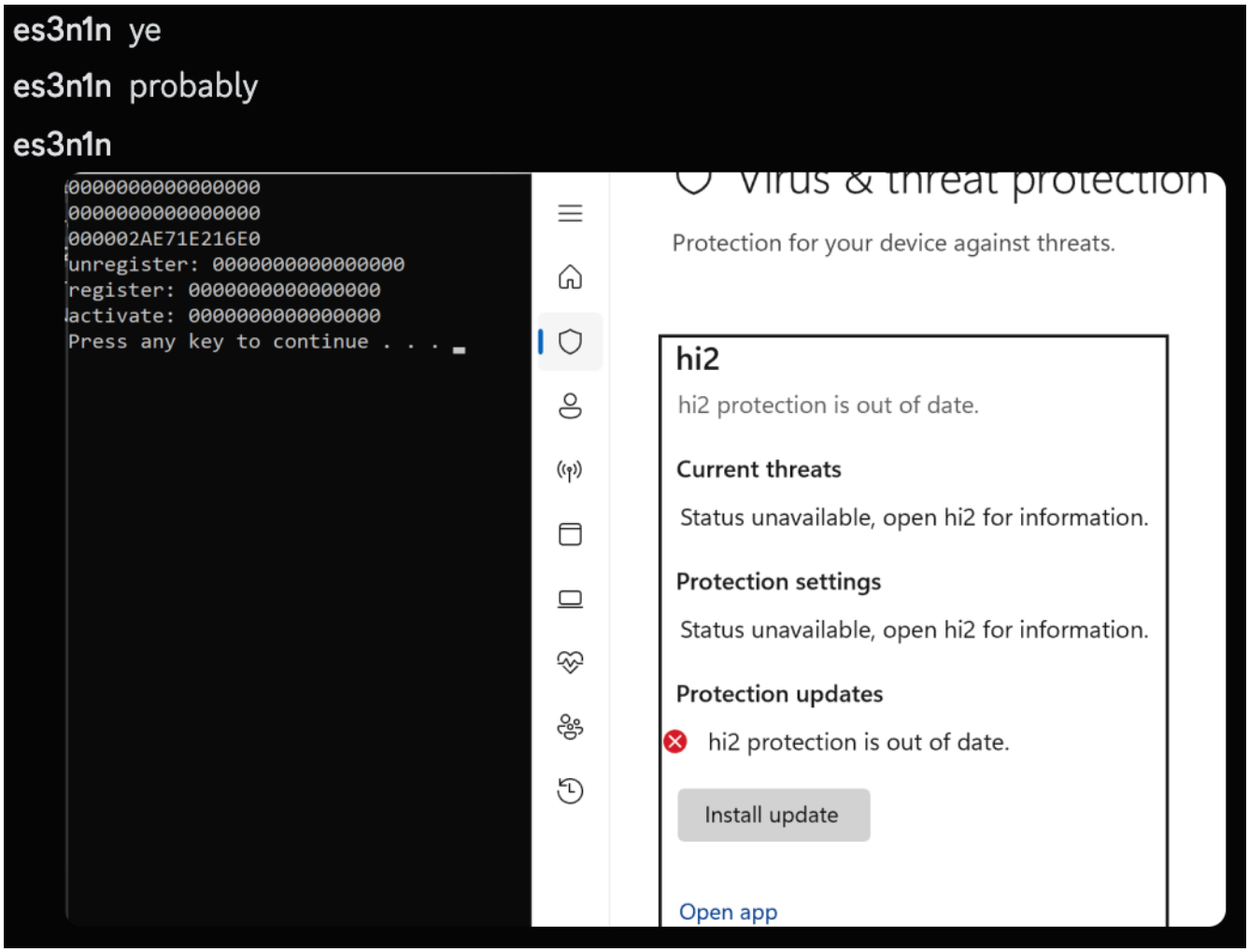


But from my last year's courtesy I knew that WSC was somehow validating the process that calls these APIs, my guess was that they are validating the signatures, which was indeed a correct guess but I didn't know that for sure yet.

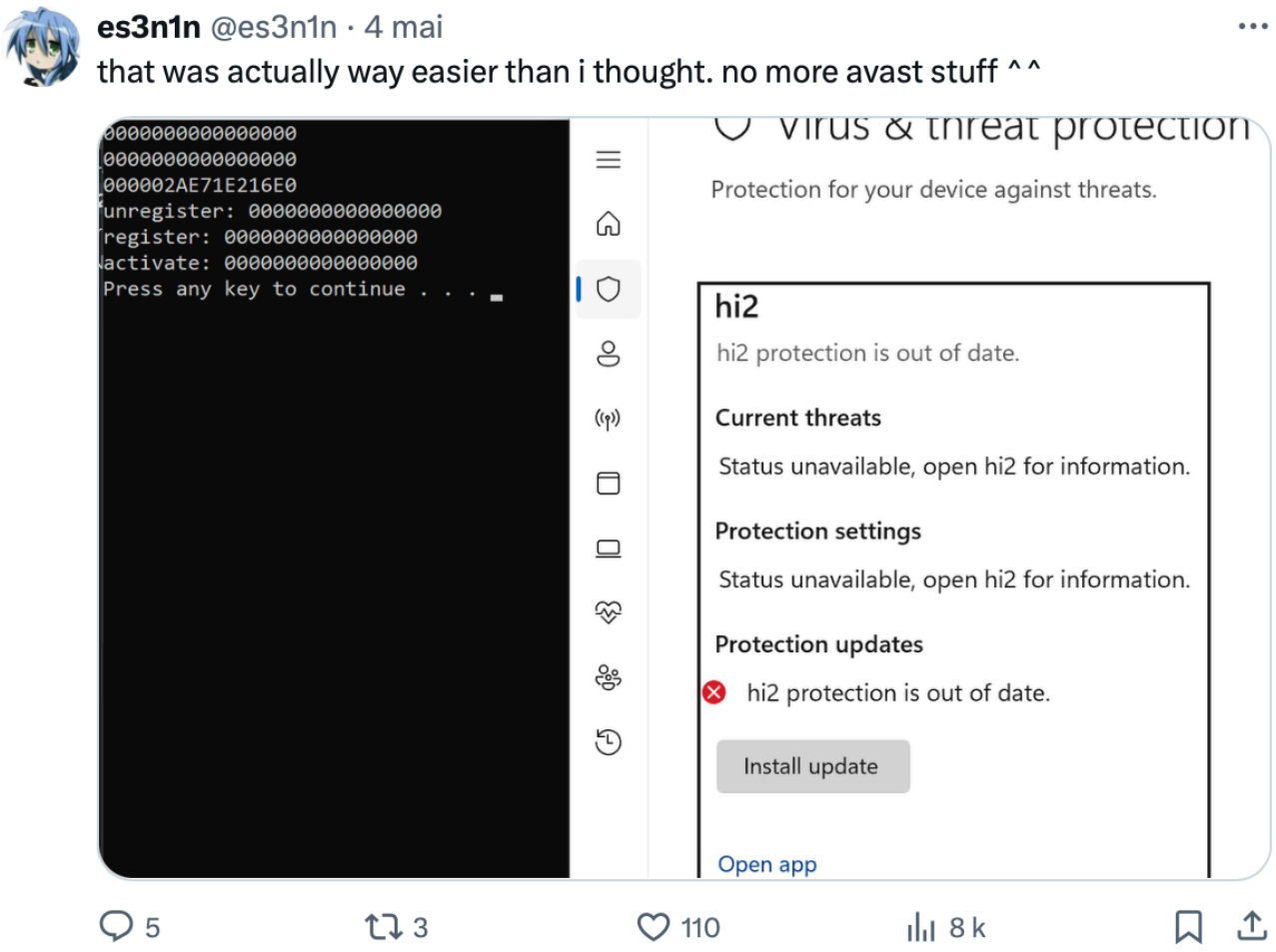
My move then was to inject my code into the same process that is doing all the WSC stuff for that AV and register my AV from there, when I did that this is what come out:



Then, I recreated an another COM call to update the status of my fresh-new antivirus I registered and everything worked like a charm as well!



As you might have guessed, this is exactly the image I posted on twitter to let my beloved followers know that I might have something cooking:



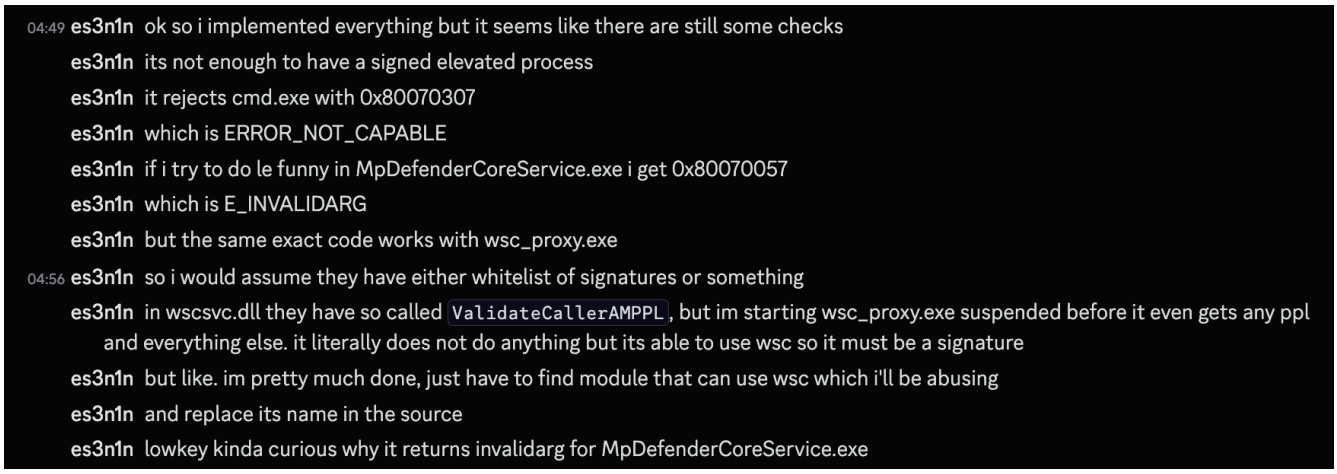
Trying to get rid of the AVs binary (Day 1)

After my initial research, I spent many hours actually enjoying life and arrived back to airbnb late at night and started tinkering with this again.

My first idea was to create a legit-signed process, inject my module in it, and execute my shenanigans from there, the exact same thing I was doing except I would use system-provided binaries and not AV's ones *(because I didn't want my new project to be removed from github by that AV)*.

As a first victim process I chose `cmd.exe` for no particular reason, just the first thing that came to my mind. However, to my surprise the api rejected my calls and I had to actually dig into the implementation to find out what was causing it.

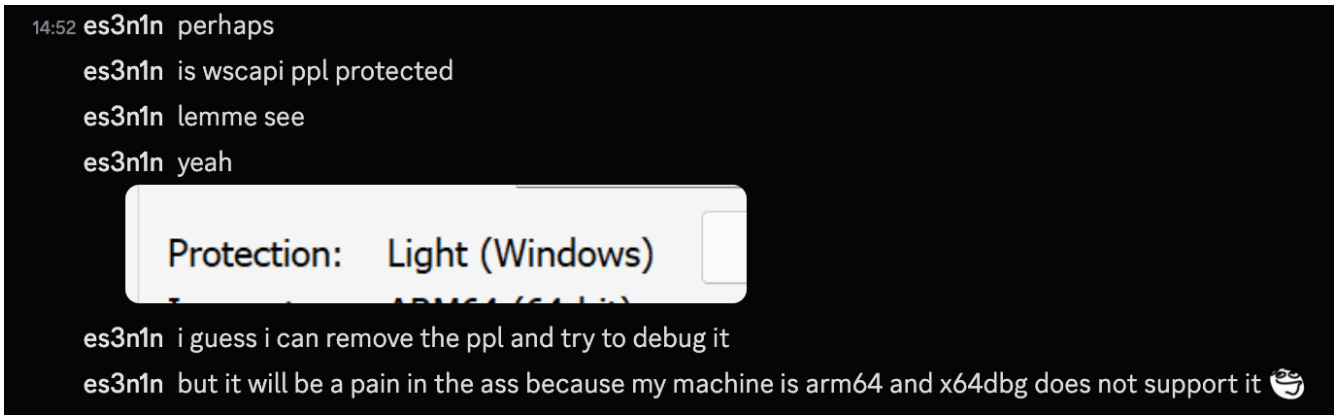
After a quick look at `wscsvc.dll`, I found out that the binary was doing some calls to check the caller process for PPL, but, the binary I was running was created using just simple `CreateProcessA` call, there is no way it was PPL protected *(and it indeed was not)*.



It was already pretty late in the morning so I went to sleep.

Setting up environment (Day 2)

When I woke up, I tried a bunch of other system processes, but nothing really worked, so I decided that it is actually time to properly reverse engineer this service, debug it and find out what is the reason behind all this - something I was really trying to avoid because I did not have an x86 machine.



As you might still remember, I was working on an arm64 macbook and there currently is no sane solutions how to emulate x86 windows on arm macbooks. I didn't feel like dealing with arm64 stuff and not being able to use my favorite x64dbg, so I asked a good friend of mine to lend me their pc while they are asleep, so that I can debug the wsc service in a virtual machine that will be running on their pc.

Luckily my friend pindos agreed to this almost instantly, shared an access to their pc through parsec and went to sleep. A rather notable thing about all this is that while I am in South Korea, they were in the USA so an average latency from my pc to theirs was around 210ms. Not very convenient, but bearable.

So, at that time, my setup looked like this:

- Build the module in windows arm64 running in parallels using MSVC
- Share the build artifacts with my host using shared folders
- Copy build artifacts using anydesk to the virtual machine running on pindos' pc
- Debug the service using parsec with 210ms latency

As you can probably guess, this was **extremely** painful to proceed with, so the speed of development/researching was really poor.

Debugging WSC service (Day 2)

Essentially, WSC service is just a dll that is being run by svchost, the only thing that blocks us from attaching debugger to it right ahead is the PPL protection, which very conveniently can be removed with a few lines of code in kernel mode. So I enabled test mode on the vm, spinned up a driver that removed PPL from a process and we were good to go.

After looking into what happened after `cmd.exe` requested to register an antivirus in WSC, I traced down that the function it fails in was a so-called `WscServiceUtils::CreateExternalBaseFromCaller`.

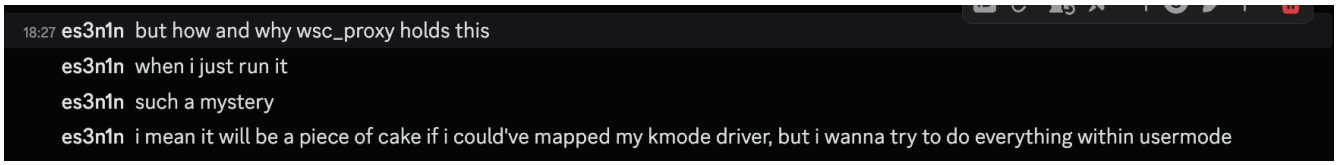
Here's a quick preview of that function:

```
v10 = RpcImpersonateClient(ClientBinding);
if ( ConvertStringSidToSidW(L"S-1-5-80-1913148863-3492339771-4165695881-2087618961-410
{
    if ( !CheckTokenMembership(0, Sid[0], &IsMember) )
    {
        IsMember = 0;
    }
    LocalFree(Sid[0]);
}
if ( IsMember )
{
    v8 = (*(__int64 (__fastcall **)(void *, struct CExternalBase **))(*(_QWORD *)a2 + 8
}
else
{
    // a bunch of some other stuff
}
```

So what it's doing is checking whether the process that is calling the RPC methods has a `WinDefend` SID on a token.

Back then, I had no idea how any of the token-related stuff worked so I spent some time to get a grasp of how it is exactly working and came to a conclusion right ahead that if we impersonate `WinDefend`, we will pass all these checks and we should be good.

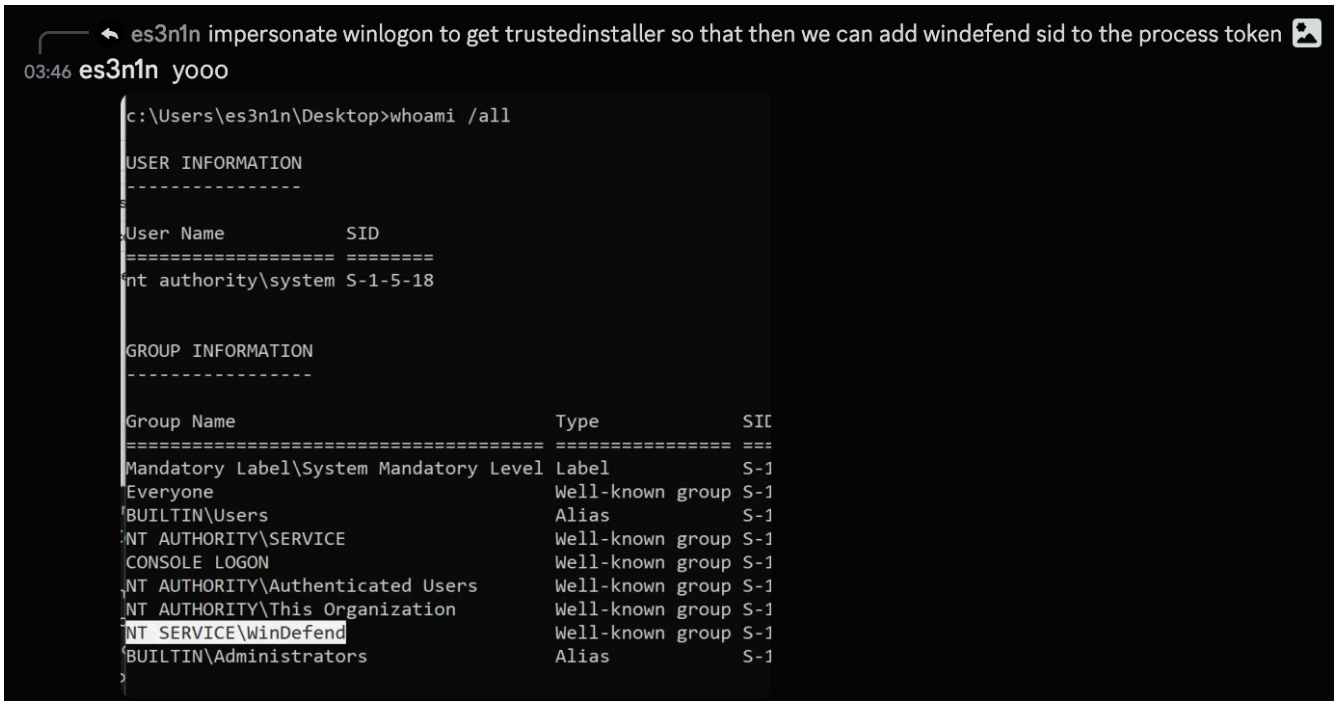
It was already evening and I wanted to see what happened in that function for the legit av binary I was initially testing my code in, and *for some reason, probably due to sleep deprivation* I came to the conclusion that IsMember was equals 1 for that binary. Now looking back on this, I have literally no clue why I assumed that, but I guess such things really tend to happen when you're trying to speedrun stuff.



Impersonating WinDefend (Day 2)

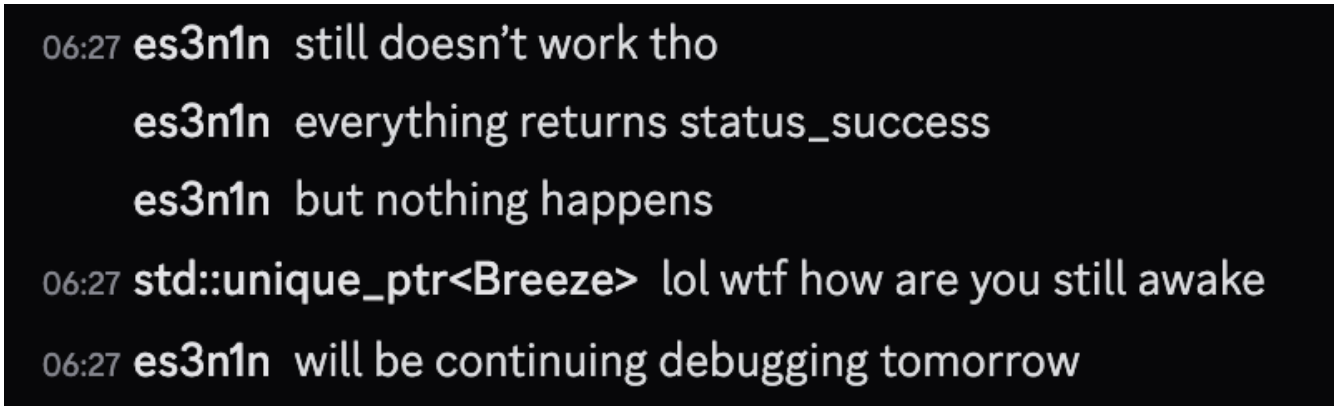
After three more hours of learning how tokens in windows work, I came up with this theory:

I spent some time doing IRL things, then came up with an implementation of the described algo:



Right ahead, after chatting with my friends I wanted to test whether my code will work if I run it within the cmd that has WinDefend sid on its token.

Surprise surprise, while all the COM calls returned `STATUS_SUCCESS`, nothing really happened. It didn't register any new AV, it didn't do anything.



Rebuilding validation algorithm (Day 3)

While I was asleep, the owner of PC I was using told me that they are going to sleep, but they didn't shutdown the pc so I can connect to it anytime. I proceeded doing silly thingies IRL and once I was tired from that, went back to analyzing what was actually happening.

A first thing I did was verifying whether the checks I thought passed for that binary, indeed passed - mostly because I don't trust myself, especially when I am sleep deprived. Man, I really tend to overlook things and misinterpret stuff sometimes.

As it turned out, in fact, the SID check did not pass for the legit AV binary as I thought it did. And it also, turned out, that if you pass that check you will be operating on the WSC object of windows defender, but this is rather useless because you can't disable it with WSC calls just like that.

So I removed everything I had for the WinDefend impersonation and started looking into the second branch of the code.

And what it revealed to me is that if the windefend check didn't pass, the service will check whether the calling binary is elevated, and then check for the signature and specific dll characteristics flag in the `CSecurityVerificationManager::CreateExternalBaseFromPESettings`, the function that gets called later on in within this branch:

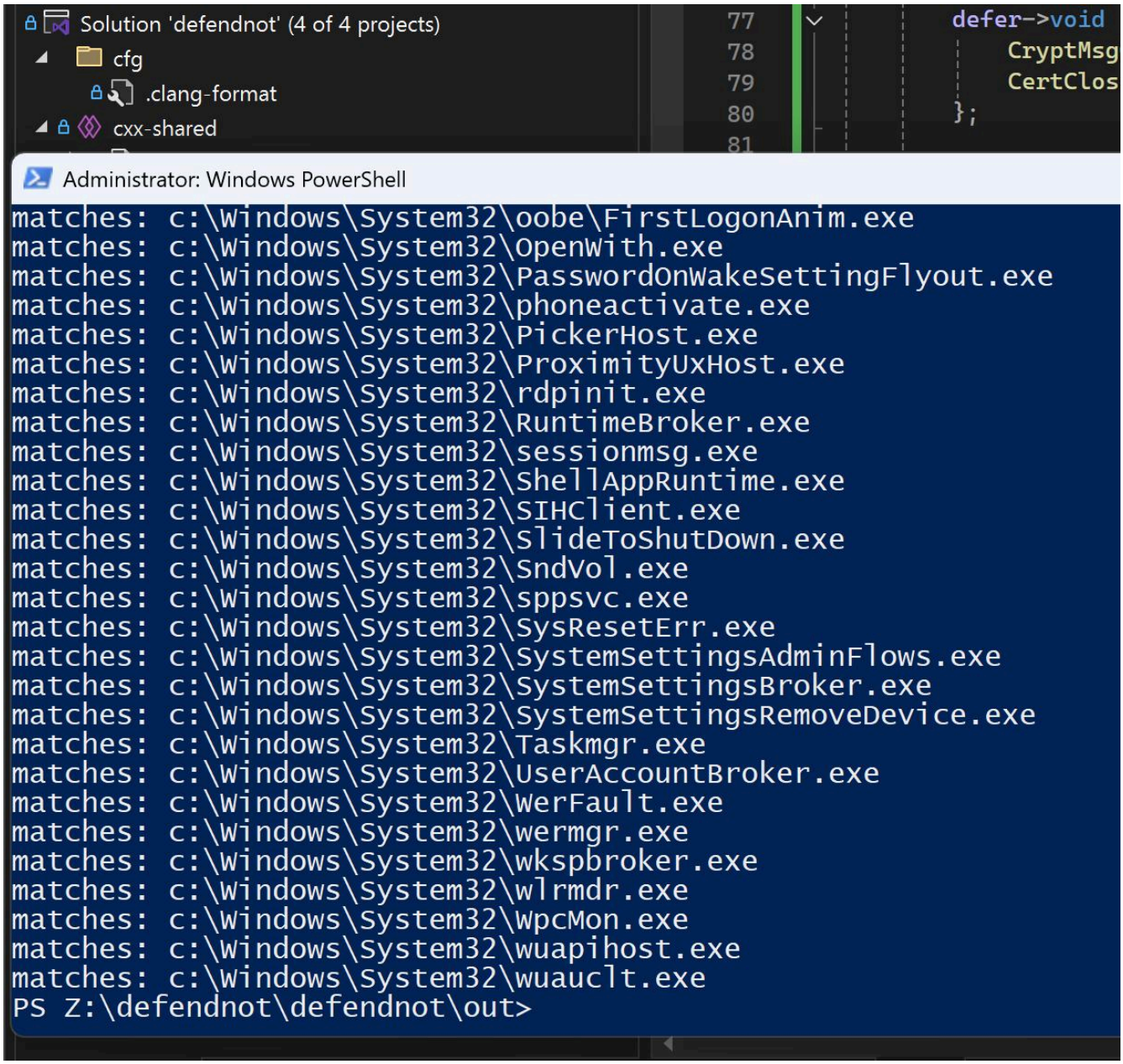
```
/// Check DllCharacteristics flag
FileW = CreateFileW(a2, 0x80000000, 1u, 0, 3u, 0x8000000u, 0);
FileMappingW = CreateFileMappingW(FileW, 0, 2u, 0, 0, 0);
v12 = MapViewOfFile(FileMappingW, 4u, 0, 0, 0);
v14 = ImageNtHeader(v12);
v6 = SLOBYTE(v14->OptionalHeader.DllCharacteristics) < 0;
UnmapViewOfFile(v13);

/// Check the signature
CertContext = GetCertContext(a2, &pCertContext);
if ( CryptHashPublicKeyInfo(
        0,
        0x8003u,
        0,
        1u,
        &pCertContext->pCertInfo->SubjectPublicKeyInfo,
        (BYTE *)&SystemTime,
        &pcbComputedHash) )
{
    /// Store signature info
}

/// ...
```

After taking a look at the structure of `DllCharacteristics`, I realized that the flag it checks for is `ForceIntegrity`.

In debugger, the failing check I saw is the DllCharacteristics, so to get a new victim process, I recreated the checks wsc is doing on a binary (`wsc-binary-check` folder in defendnot repo) and tested all the System32 binaries against it.



Using Taskmgr as a victim process (Day 3)

At that time my friend has already woken up and had to do some stuff for the uni on their PC, so I connected directly to the vm using Parsec and my setup immediately got even worse because on top of the latency issues, now the encoding was being done in software which was super slow.

I tried to just replace `cmd.exe` with `Taskmgr.exe` in my code and unfortunately something was still missing, I was still getting errors from RPC.

After an attempt to debug this on the same vm, I realized that it was pratically impossible to debug anything because everything was lagging super hard, some keys weren't pressing,

sometimes when I pressed a key once it was pressed two times on the machine - something that is super difficult to put up with.

So what I did, is I spent $30 on a shadow.tech subscription after a friend recommended me this service, as it gives you a bare-metal access to the machine which was something I needed.

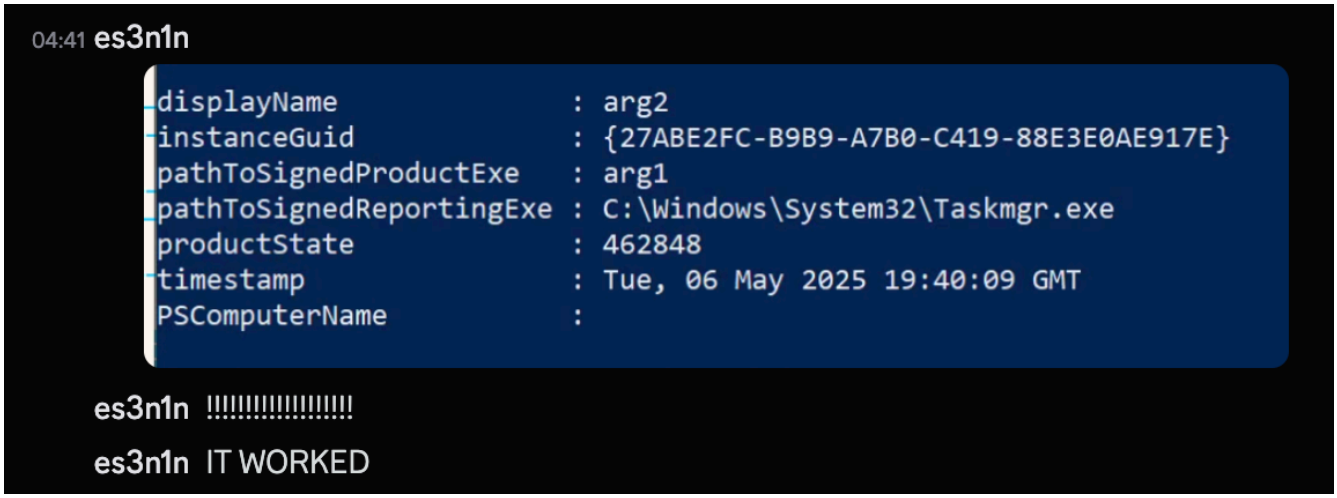
After waiting an hour, my VM was created, I hopped in and noticed that because the windows version they are using is much older than the latest one, the code I was interested in wscsvc wasn't inlined to a single function and the overall "quality" of decompiled code was much better. Too bad I already had everything figured out, but if you are taking a look at wsc and confused about something due to the fact that everything was inlined in a single function, you can take a look at the older versions.

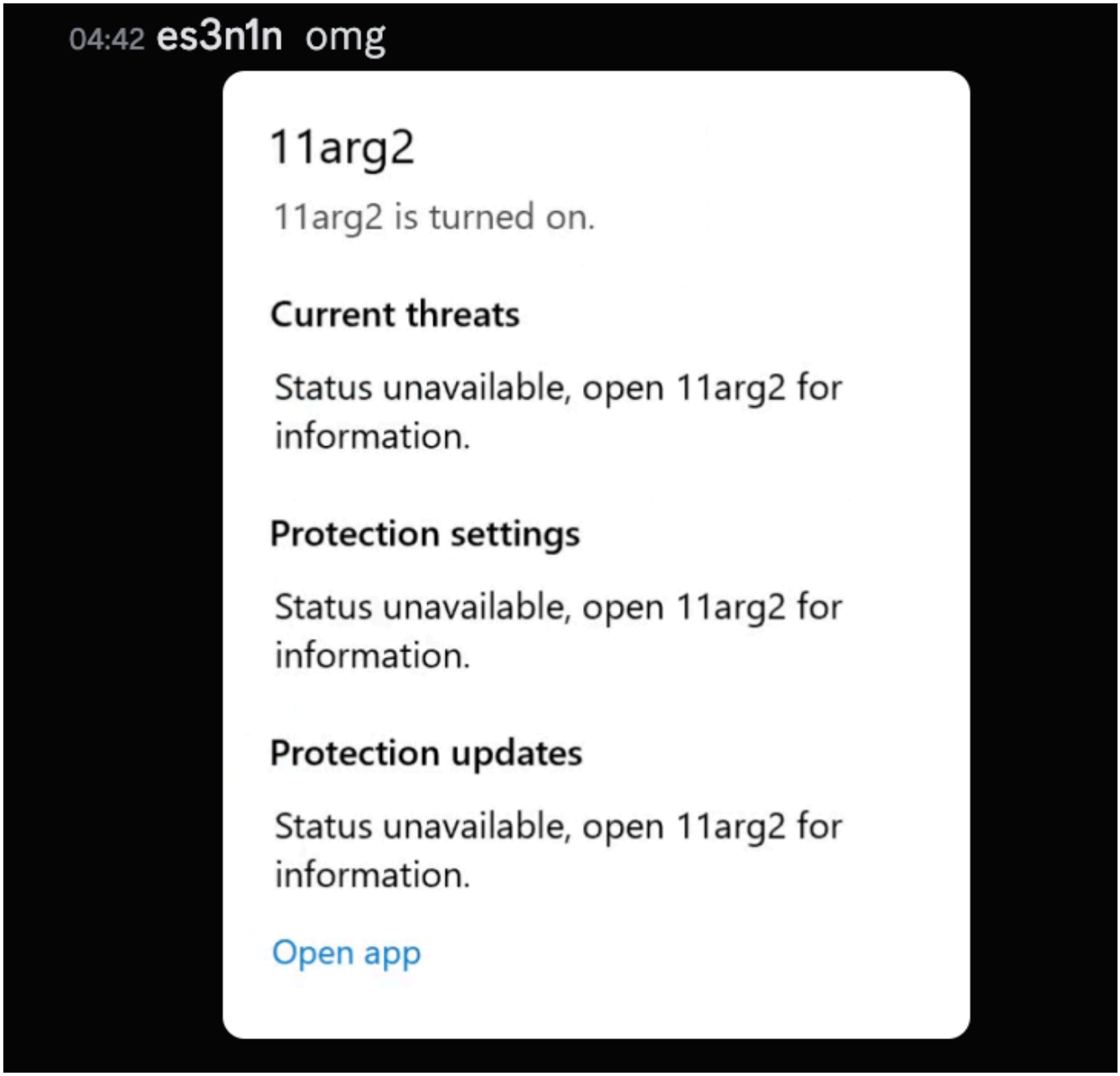
To cut the chase, the error that was happening on a VM was due to invalid name I passed as the AV name.

You see, the way how defendnot transfers data from defendnot-loader to defendnot.dll is by creating a `ctx.bin` file with serialized parameters. While I added a proper IPC mechanism in the project for state tracking, I was too lazy to port this context thingy to use the same IPC buffer and left the config still using this `ctx.bin` thing. It was a leftover from the initial no-defender code after all.

So what happened is that my code was reading an invalid `ctx.bin` file, because the function that deduced the path of path of this file was broken (it used the base folder of the module of `Taskmgr.exe`, not of the `nodefend.dll`), it read a bunch of null bytes and passed that as a name of the AV. Of course wsc rejected this buffer and returned an error.

After figuring it out, I fixed the issue, tested and here is what happened:





Cleaning up code (Day 3)

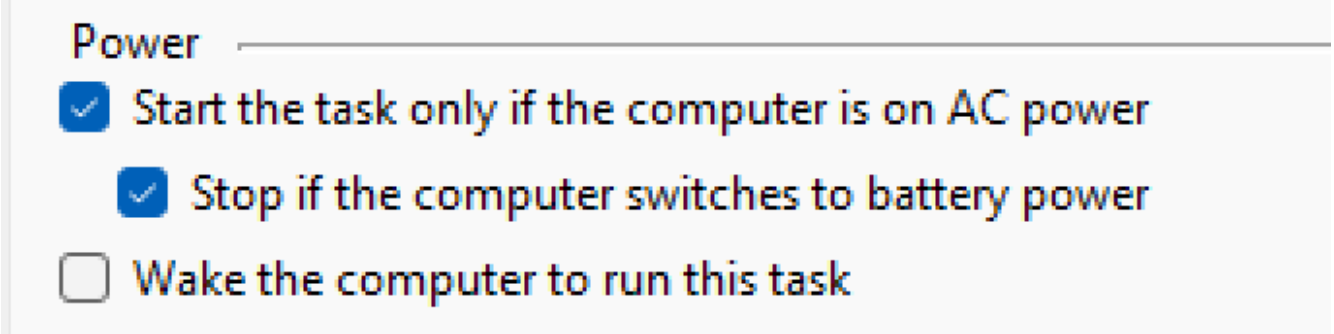
I wanted to finish everything on this day so I stayed up until 8 am cleaning up code and implementing some other features, such as adding itself to autorun.

At 8 am, I had everything done except the autorun part because it was just not working. I tested numerous ways how to do it, but nothing really worked so I just went to sleep.

Implementing autorun (Day 4)

When I woke up, I immediately started working on the autorun and realized that while I was creating a task, the reason why my autorun code did not work is because of these two check

boxes:



This is exactly what was happening. My laptop was not on the AC power and the task simply was not executing. After unsetting these two flags, it started working.

I spent a few more hours cleaning up the code, and that was it.

Conclusion

While that was a fun experience, I don't think I would want to repeat everything I went through these past few days. Considering only that diabolical environment I had, it was already enough to make me lose my mind.

Thanks for reading, a more technical documentation of wsc will be released a bit later by someone else.

Acknowledgements

- Pindos for heating up their room by the pc running at night so that I can debug the WSC service
- MrBruh for poking me into researching this and listening to my mad ideas while I was working on this
- Everyone else i was texting during these few days
- I love you kimchi
- The graffiti artist that VANDALIZED our wall