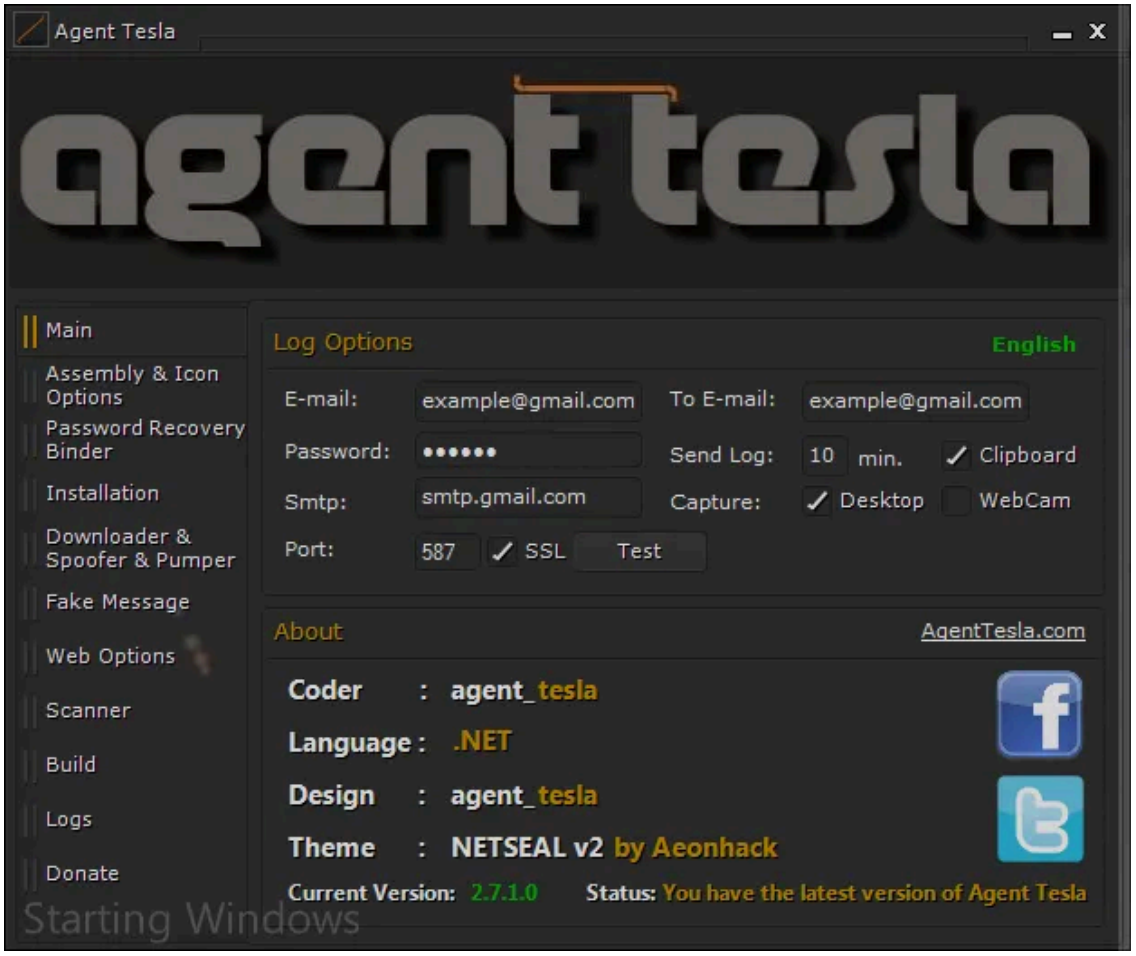
Decrypting AgentTesla strings and config

By NexusFuzzy

Published: 2020-09-03 · Archived: 2026-04-05 18:21:04 UTC



The nefarious malware AgentTesla has been around for some years and remains a serious threat

Working in cyber security is all about being faster than your adversary to limit or prevent damage to the systems and users you are about to protect.

This naturally leads to the need to automate as much as possible to be able to react quickly to new threats. As you can see from the stats of Any.run AgentTesla is still one of the most popular malware families when it comes to commodity malware:

Press enter or click to view image in full size

AgentTesla remains a threat

Due to the number of samples to analyze I created a little tool called Edison which is able to decrypt all strings used by AgentTesla with the fact in mind that the exfiltration methods are also present in the strings. Equiped with this knowledge let’s dive in!

The sample

My prefered site to get interesting samples is bazaar.abuse.ch so you may also use this site to get the following sample to follow along with theses steps — in a secured VM of course (We are about to execute the malware)

d29da0500ff7aecab3d24397cb745554f399dce5ab59f4ed7a95f6f959b62584

Required tools

As already explained you **need a VM** to run our little experiment since we will execute this sample. If you aren’t already using it, this is a good moment to start using FlareVM which is maintained by FireEye.

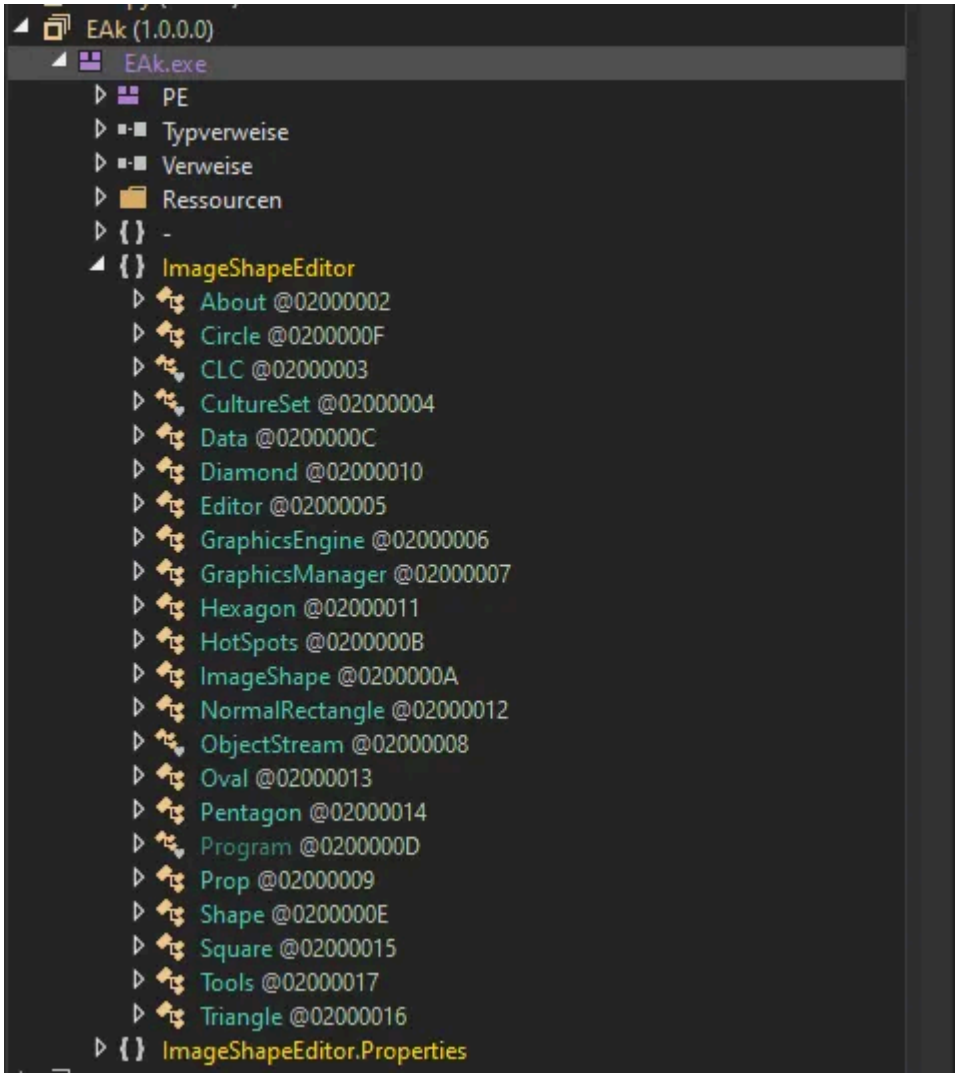
Once you got your VM up and running it’s time to get Hollows Hunter. The reason for this is, that most if not all malware comes “packed”. This means that the true intention of the malware is not visible at first glance. Only if the packer deems your system worthy to get infected (No VM, only specific regions etc.) it will unpack the malware and pass control to it. Going in too much details is out of the scope of this post so we chose the automatic way with Hollows Hunter. Once the malware is unpacked, Hollows Hunter will dump it to disk, fix some things and you are good to go.

Not necessary but really helpful in our analysis is dnSpy. If you are using FlareVM, it should already be installed.

Last but not least you'll need Edison to automatically extract the strings from the sample.

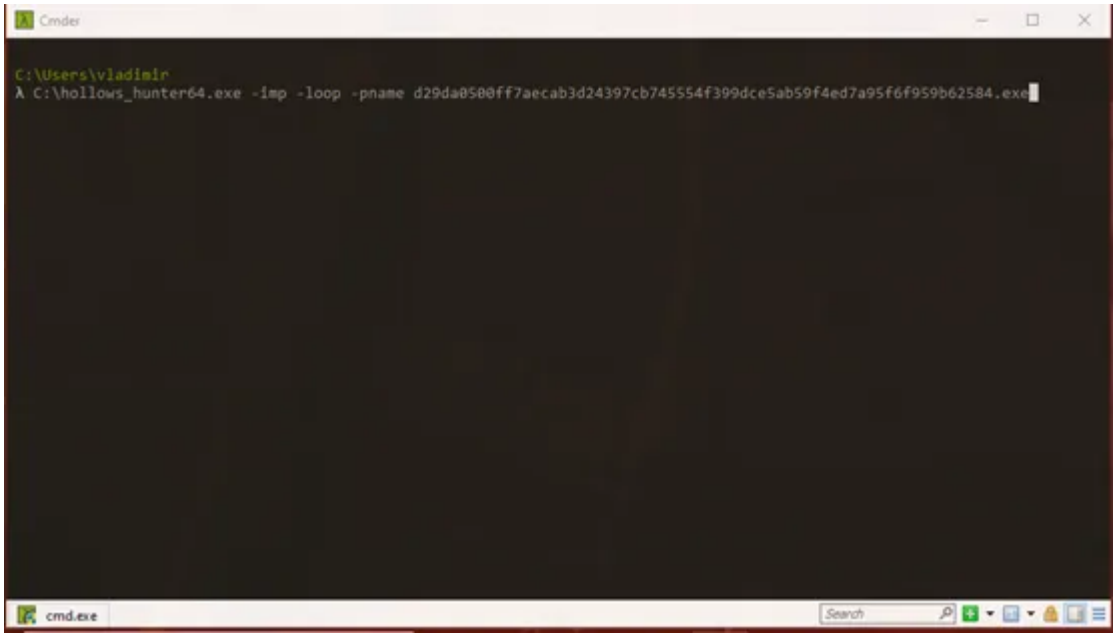
Let's infect ourselves

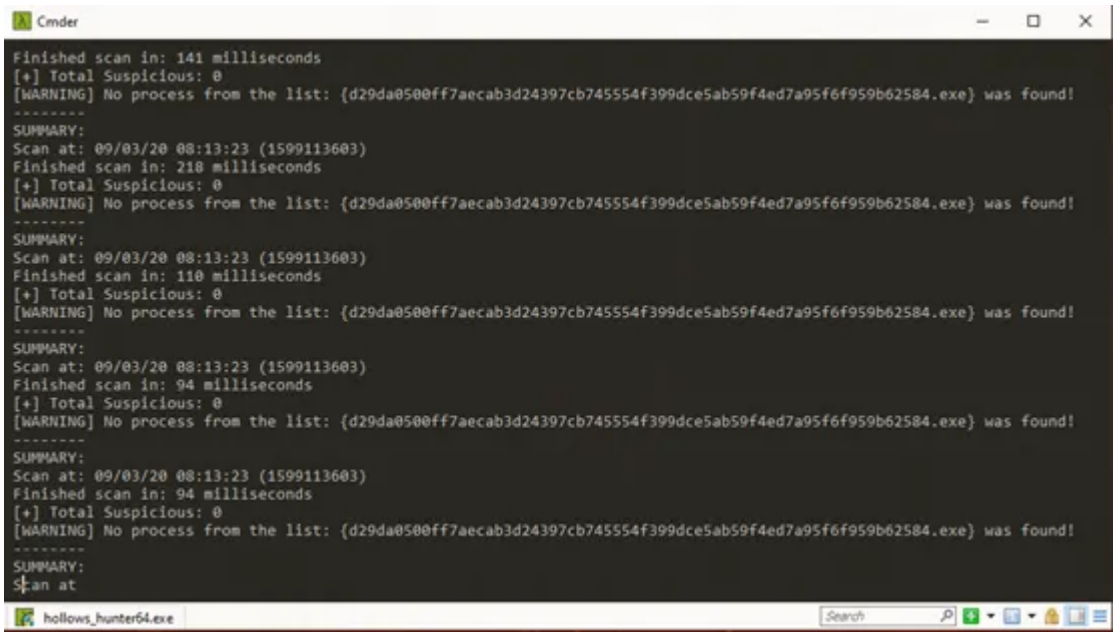
Once you downloaded the sample, we open it up in dnSpy:



That doesn't really look like AgentTesla to be honest

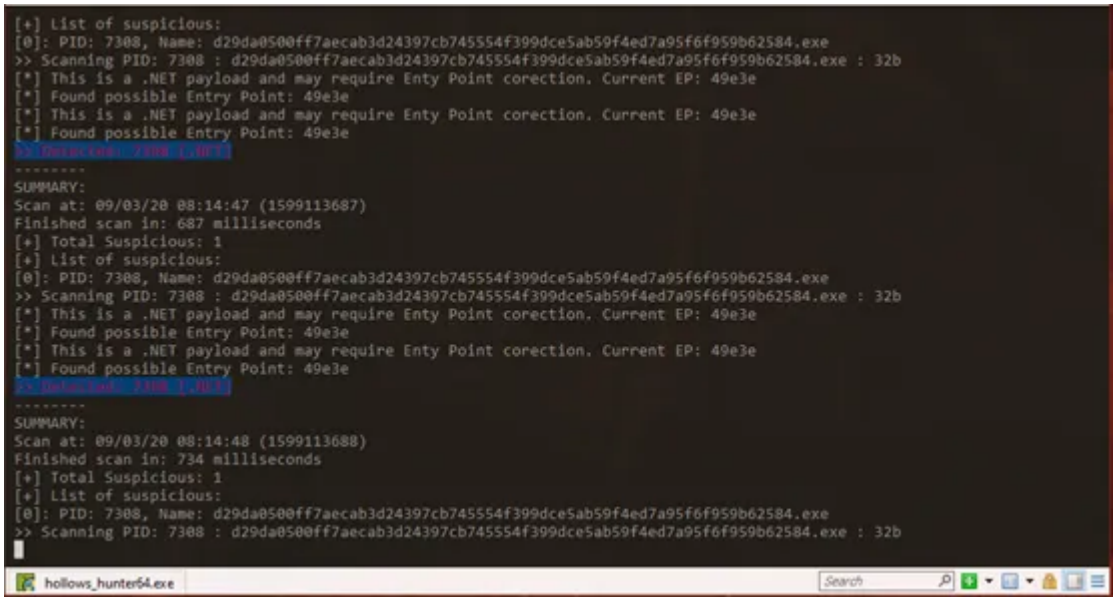
As we can see when we go through the methods it doesn't look like AgentTesla so at this point we assume that it's packed. We now use Hollows Hunter to watch for injections and automatically dump them to disk:





If Hollows Hunter is working, you should see output similar to this

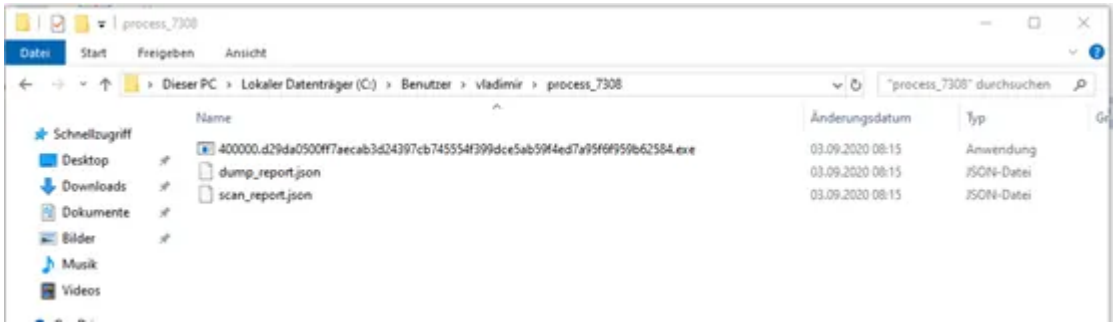
After we got Hollows Hunter in place, we start the malware itsself and wait until a new process is spawned. As expected Hollows Hunter catches this:



Gotcha!

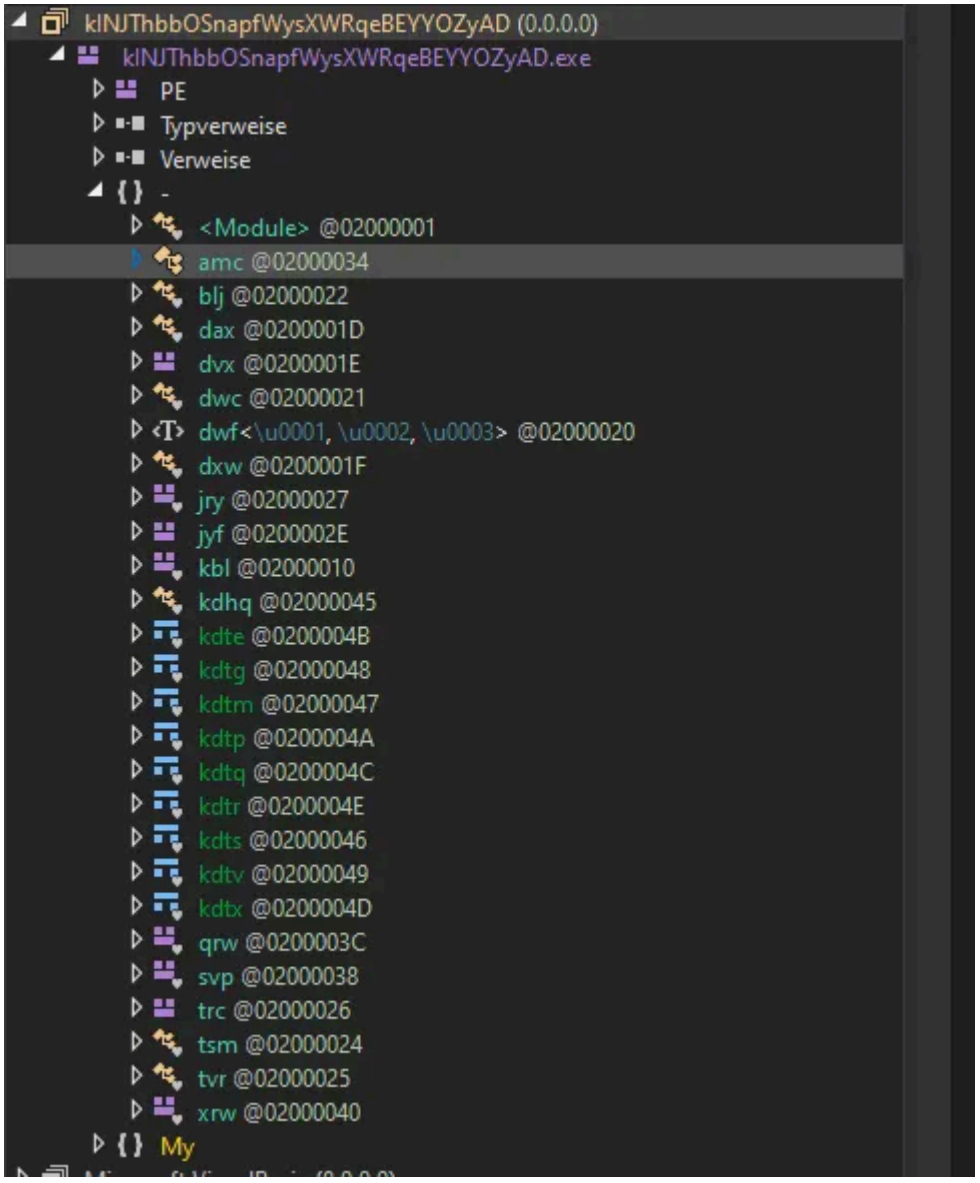
Once we made sure that we got a dumped sample we can kill the malware itsself and Hollows Hunter:

Press enter or click to view image in full size



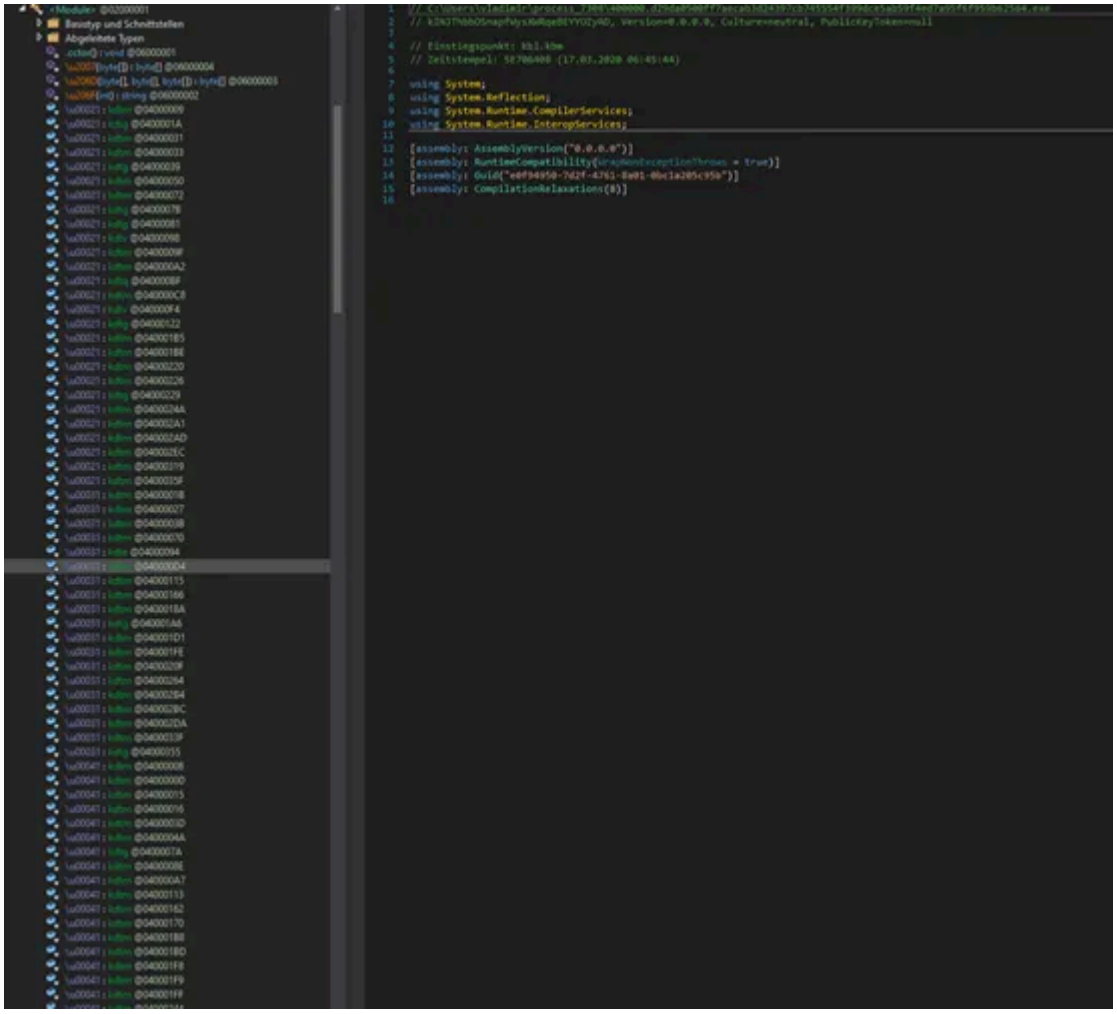
The dumped file

To confirm our assumptions we open this file in dnSpy again:



Hello, AgentTesla!

One characteristic thing about AgentTesla is a huge object array of uint[] which turn out to be the encrypted strings:



Note the huge list of uint[] at the left

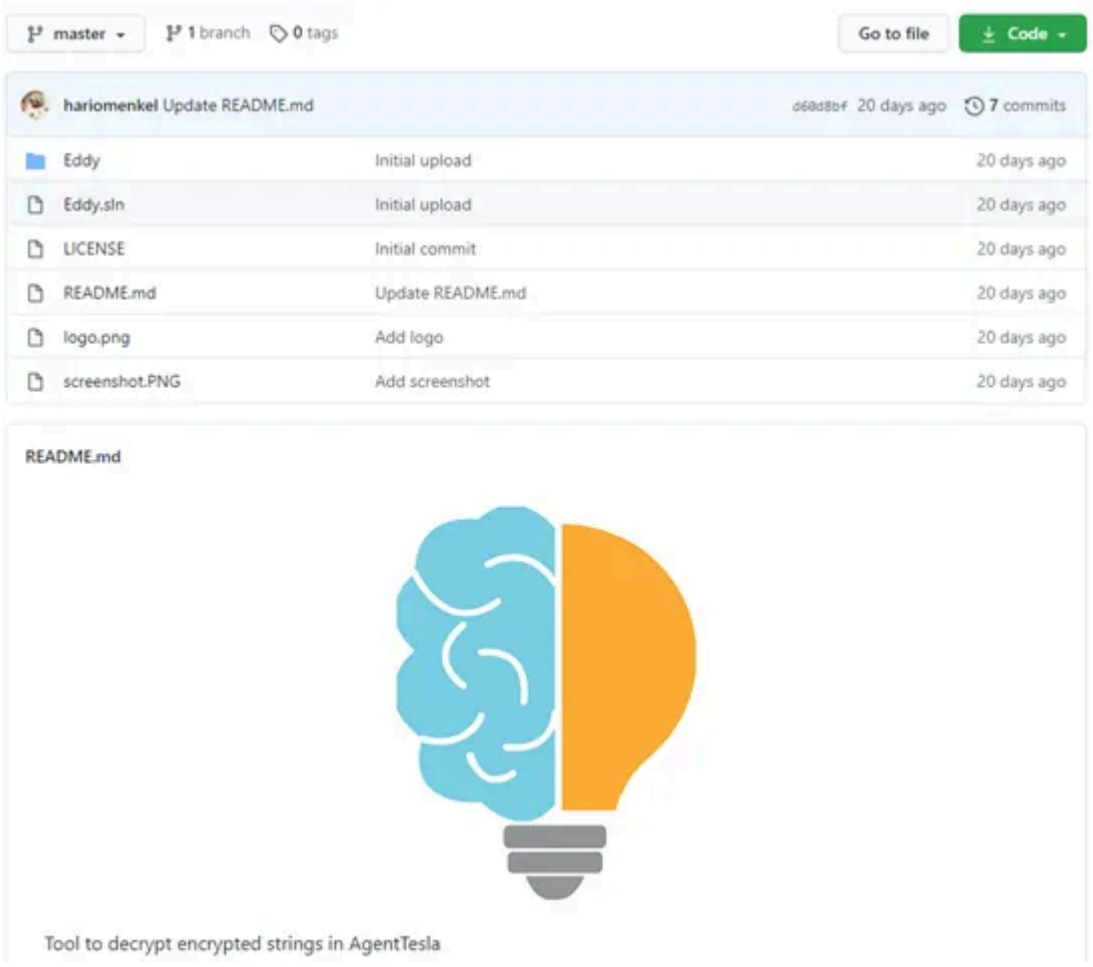
A good training for your fingers if you decide to scroll through the list

```
        return "";
    }

    // Token: 0x06000003 RID: 3 RVA: 0x00016AB8 File Offset: 0x00014CB8
    internal static byte[] \u206D(byte[] A_0, byte[] A_1, byte[] A_2)
    {
        Rijndael rijndael = Rijndael.Create();
        rijndael.Key = A_1;
        rijndael.IV = A_2;
        return rijndael.CreateDecryptor().TransformFinalBlock(A_0, 0, A_0.Length);
    }
```

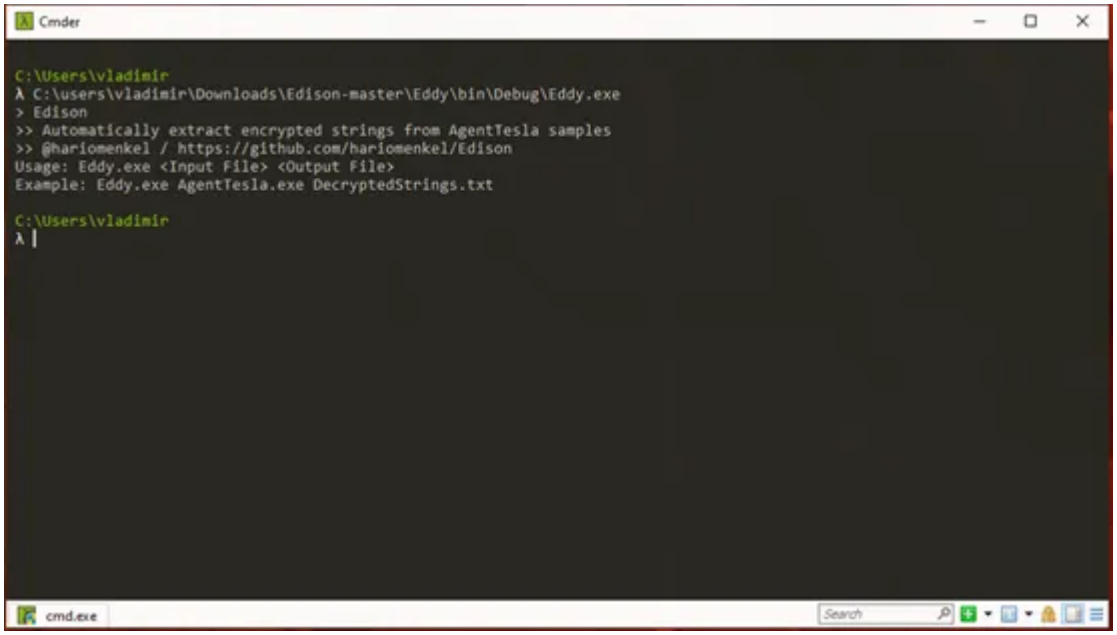
Key? IV? CreateDecryptor? Looks like we found the juicy parts!

You could now dynamically analyze the malware, set breakpoints, work your way to the obfuscated control flow **OR** we fire up Edison to just decrypt all strings for us:



If not already downloaded, this is your chance!

Once downloaded you can start it without arguments to see the usage:



No rocket science. Really.

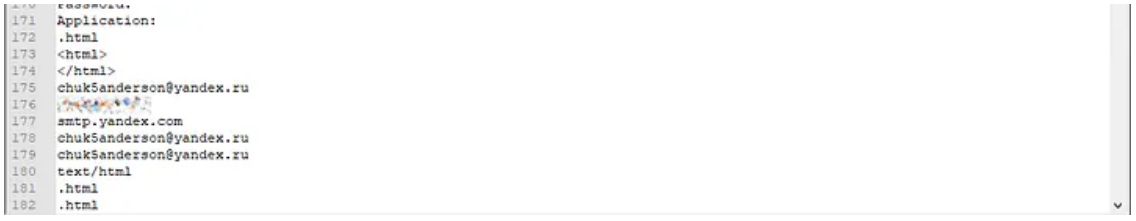
We do as advised:

Just point Edison to the dumped binary and specify an output textfile

Once it’s finished we can open up the text file:

All the strings

Press enter or click to view image in full size



That’s what we were interested in — the address used for exfiltration!

Teach me master!

You can browse the repository to see the complete source code but for reference purposes you can see the interesting part here:

Press enter or click to view image in full size

```
byte[] array_Key = new byte[32];
byte[] array_IV = new byte[16];
int keyLength = 32;
int ivLength = 16;

Assembly a = null;
try
{
    a = Assembly.LoadFile(input);
}
catch (Exception ex)
{
    Console.WriteLine("Error while loading the file: " + ex.Message);
    return;
}

Module[] modules = a.GetModules();
var fields = modules[0].GetFields();
foreach (var field in fields)
{
    var objArr = field.GetValue(null);
    var values = (object[])objArr;

    for (int i = 0; i < values.Length; i++)
    {
        try
        {
            uint[] encryptedValue = (uint[])values[i];
            byte[] arrEncryptedValue = new byte[encryptedValue.Length * 4];
            Buffer.BlockCopy(encryptedValue, 0, arrEncryptedValue, 0, encryptedValue.Length * 4);
            byte[] arrPayload = arrEncryptedValue;
            int offsetKeyAndIV = arrPayload.Length - (keyLength + ivLength);
            byte[] array_EncryptedValue = new byte[offsetKeyAndIV];
            Buffer.BlockCopy(arrPayload, 0, array_Key, 0, keyLength);
            Buffer.BlockCopy(arrPayload, keyLength, array_IV, 0, ivLength);
            Buffer.BlockCopy(arrPayload, keyLength + ivLength, array_EncryptedValue, 0, offsetKeyAndIV);

            using (System.IO.StreamWriter file = new System.IO.StreamWriter(output, true))
            {
                Console.WriteLine(":: Success :: " + Encoding.UTF8.GetString(Decrypt(array_EncryptedValue, array_Key, array_IV)));
                file.WriteLine(Encoding.UTF8.GetString(Decrypt(array_EncryptedValue, array_Key, array_IV)));
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("!! Error !! " + ex.Message);
        }
    }
}
```

No rocket science either

I am using reflection to load the AgentTesla sample and afterwards I’m getting all fields of the first module which turn out to be the uint[] elements.

Through reversing the sample I found out that the Key and IV for every uint[]/string differs and is found at the end of every uint[]. Since we know the size of the key and the IV we can use theses values to pass those values to the

decrypt function we saw earlier.
