Initial analysis of PasswordState supply chain attack backdoor code

lordx64.medium.com/initial-analysis-of-passwordstate-supply-chain-attack-backdoor-code-aaff1df389e4

taha aka lordx64

April 24, 2021

tah
a aka

taha aka lordx64

Apr 23, 2021

·

7 min read

Photo by on

UPDATE — 24 April 2021: The second stage payload was shared with me by Peter Kruse (@PeterKruse) thanks to him. Please find the second stage payload analysis at the end of this document.

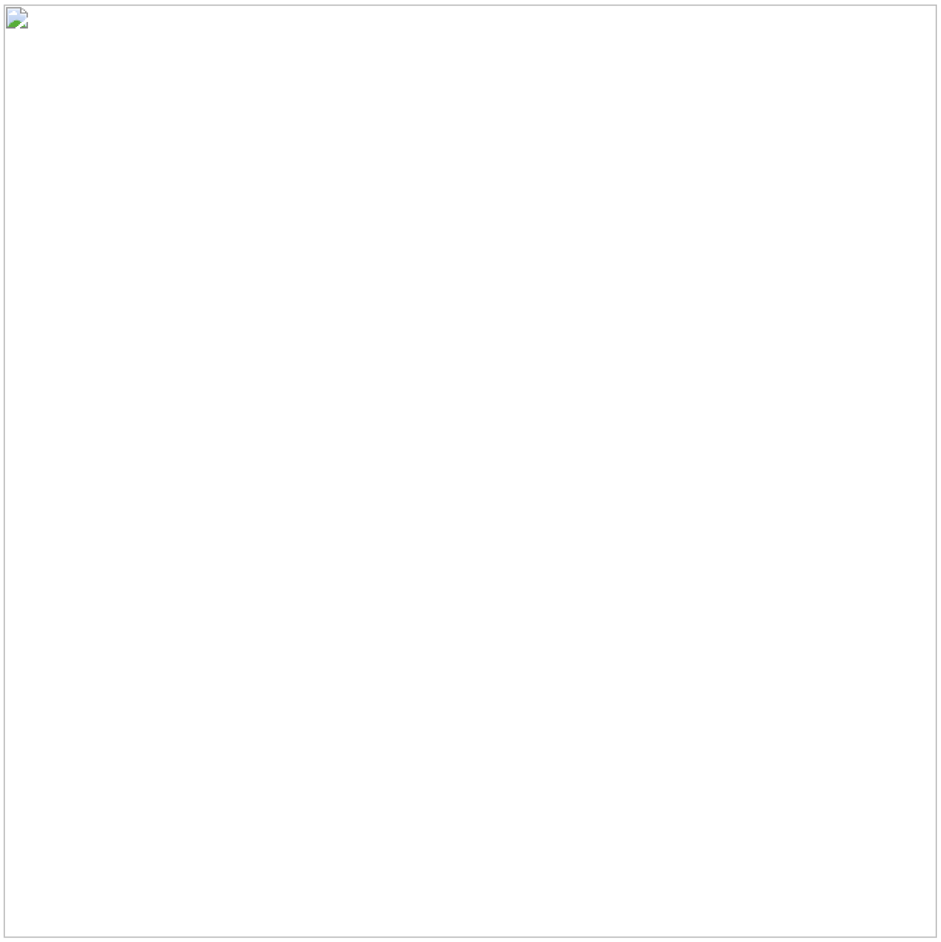
One hour ago, I became aware of this supply chain attack from a tweet by Kim Zetter. If you don't follow her please do and subscribe to her blog for amazing cyber espionage related content : zetter@substack.com

Why this quick analysis? well it's Friday, and this news will worry a lot of people, and many would want to know what is this about and fast.

> Note: Based on ClickStudios customers, this attack might be impacting 29 000 enterprise customers including many Fortune 500 clients.

Based on the initial information we have, I will explain in details how this legitimate software: PasswordState from ClickStudios, was backdoored, as well as how to detect this backdoor.

There was an official statement from CSIS group on their website, saying that ClickStudios recently notified their customers about a breach resulting in a supply chain attack conducted via an update of the password manager PASSWORDSTATE. the attack was noticed between 20th of April 2021 8:33 PM UTC and 22nd of April 2021 00.30am UTC



source:
CSIS group shared IOC’s, I confirmed them and below is the analysis of the backdoor code.

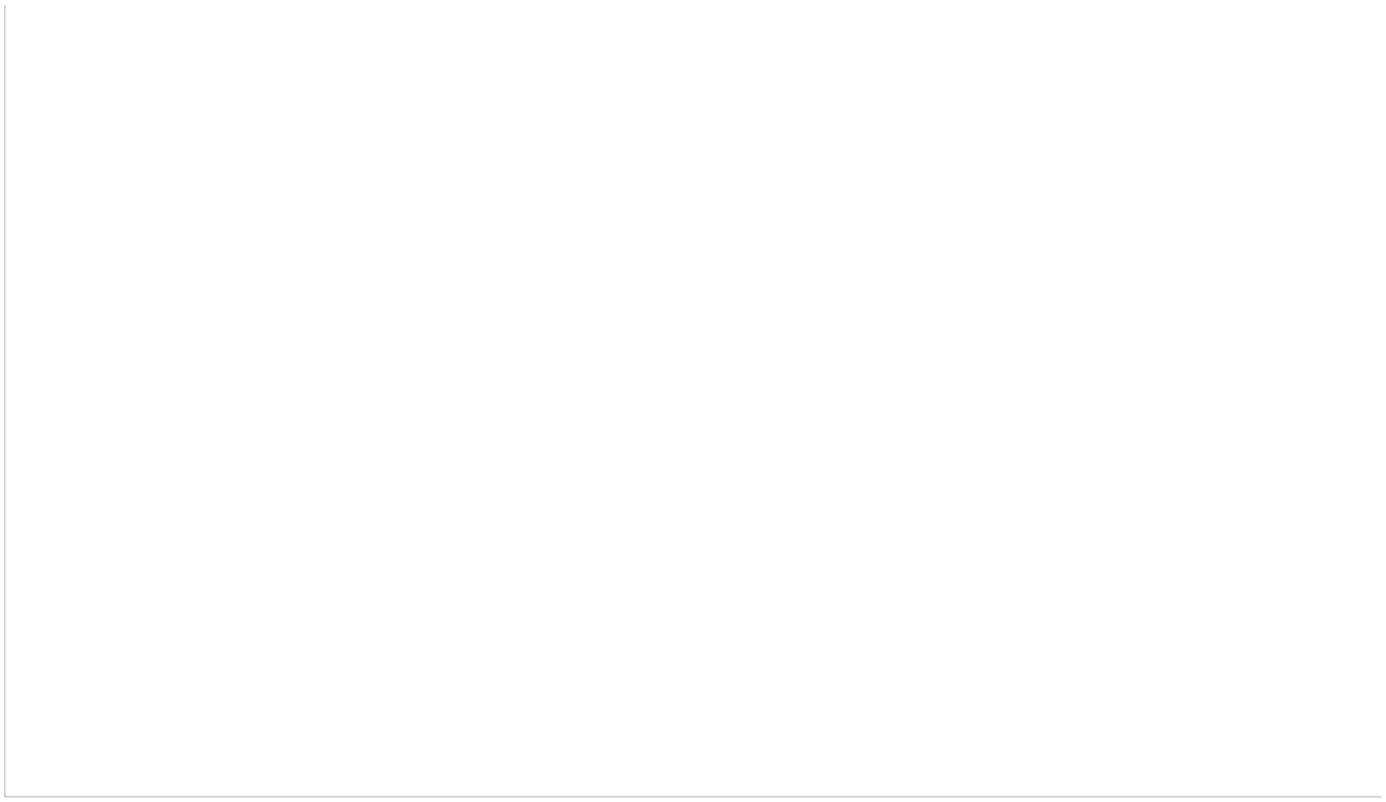
Sample in question is a rogue DLL file, downloaded during an update of PasswordState (I am not sure yet which version of PasswordState exactly): f23f9c2aaf94147b2c5d4b39b56514cd67102d3293bdef85101e2c05ee1c3bf9 Moserware.SecretSplitter.dll

The backdoor code was inserted in the assembly, via a Loader method see below :

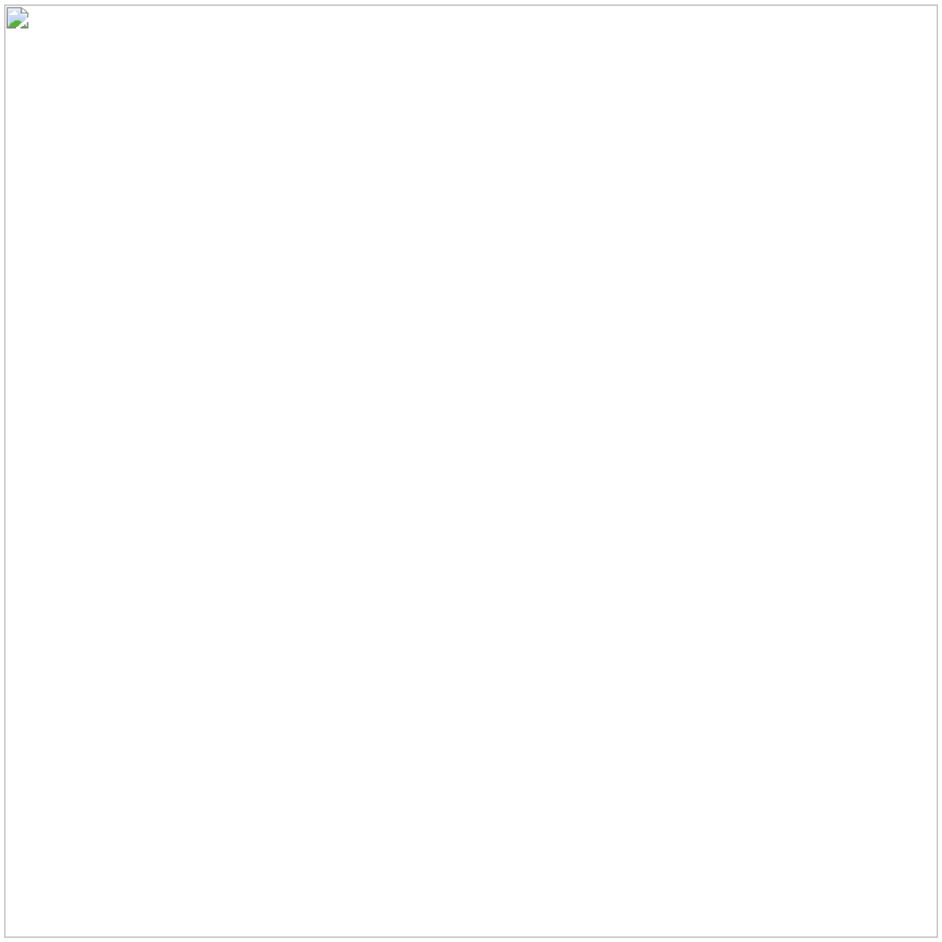
clean vs backdoored
(compared it with a clean version of this dll: 1ee0f14c44058e3d0d1c19b4713d573c81b49c28ed58bd41c72832c78f7d1464)

Next let’s jump into the backdoor code and how it’s been called from the legitimate code:

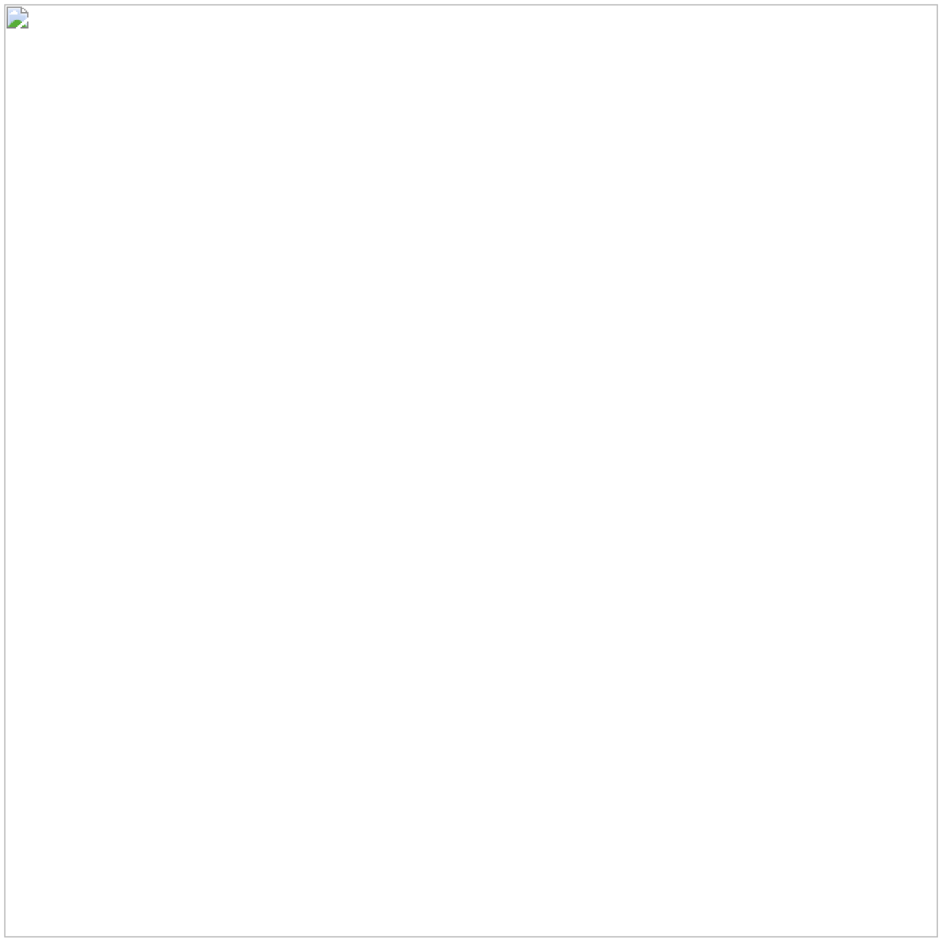
Inside the namespace Moserware.Security.Cryptography, there’s class named Diffuser, containing a protected method called Diffuse() which is the constructor of this virtual class:


clean vs backdoored version

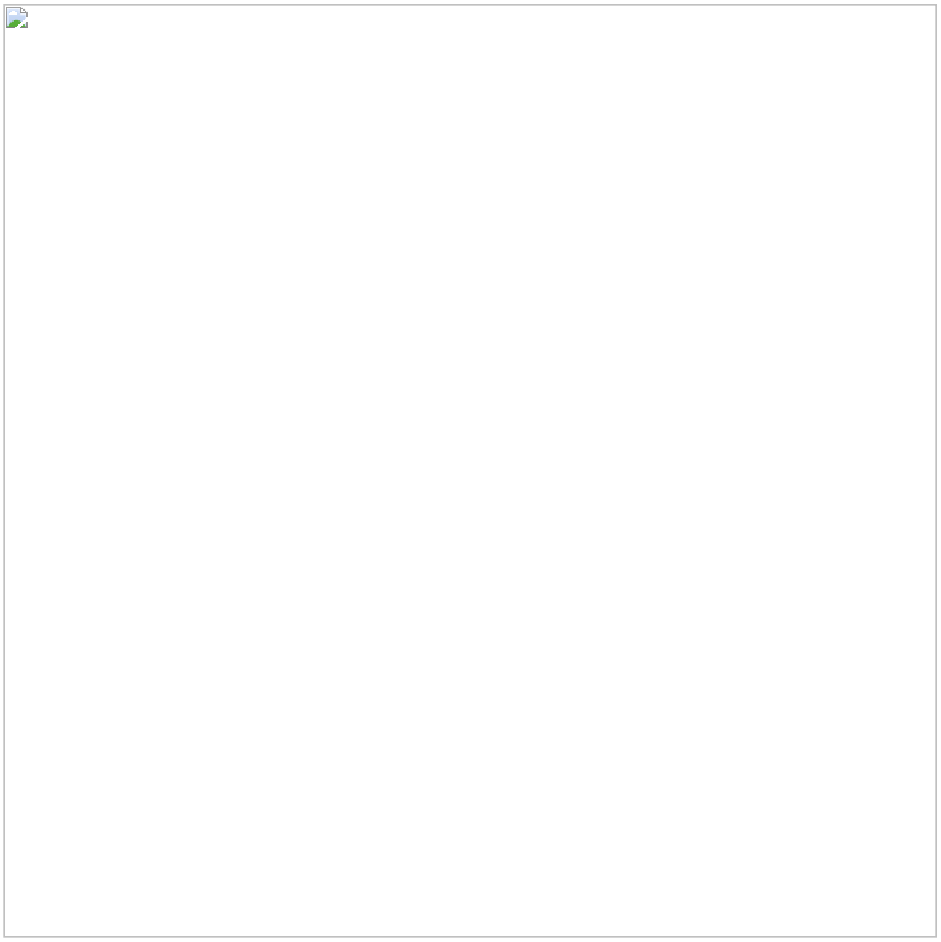
I checked how this class Diffuser() was called, it is indeed used by 3 other classes, amongst them XteaDiffuser that **inherit** from Diffuser.



Which means every time a object XteaDiffuser is instantiated the constructor Diffuser() will be called and the backdoor code will run. XteaDiffuser object is used by SsssDiffuser() Class as an attribute and used for every Xtea encryption/decryption operations (kinda like a wrapper). which in turn called by SecretSplitter as the default diffuser. It seems PasswordState uses Xtea for splitting the secret password (based on SecretSplitter class name, didn’t look in the code in details):

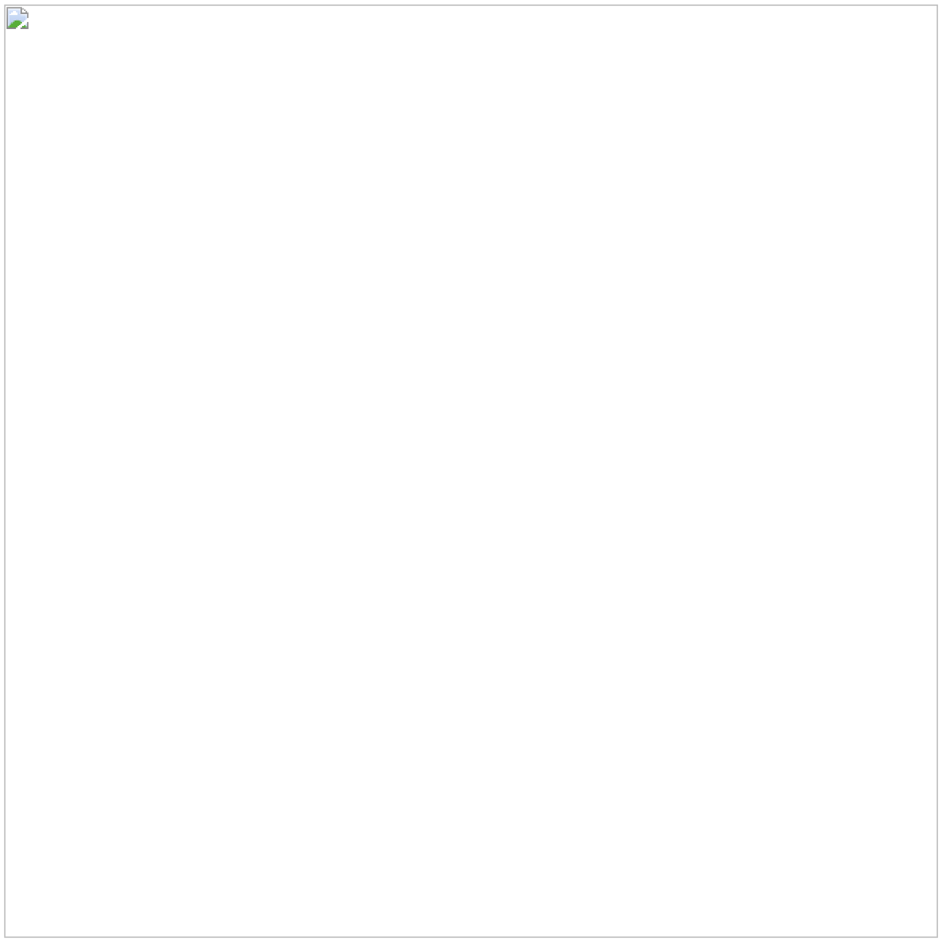


Back to the diffuser() constructor:



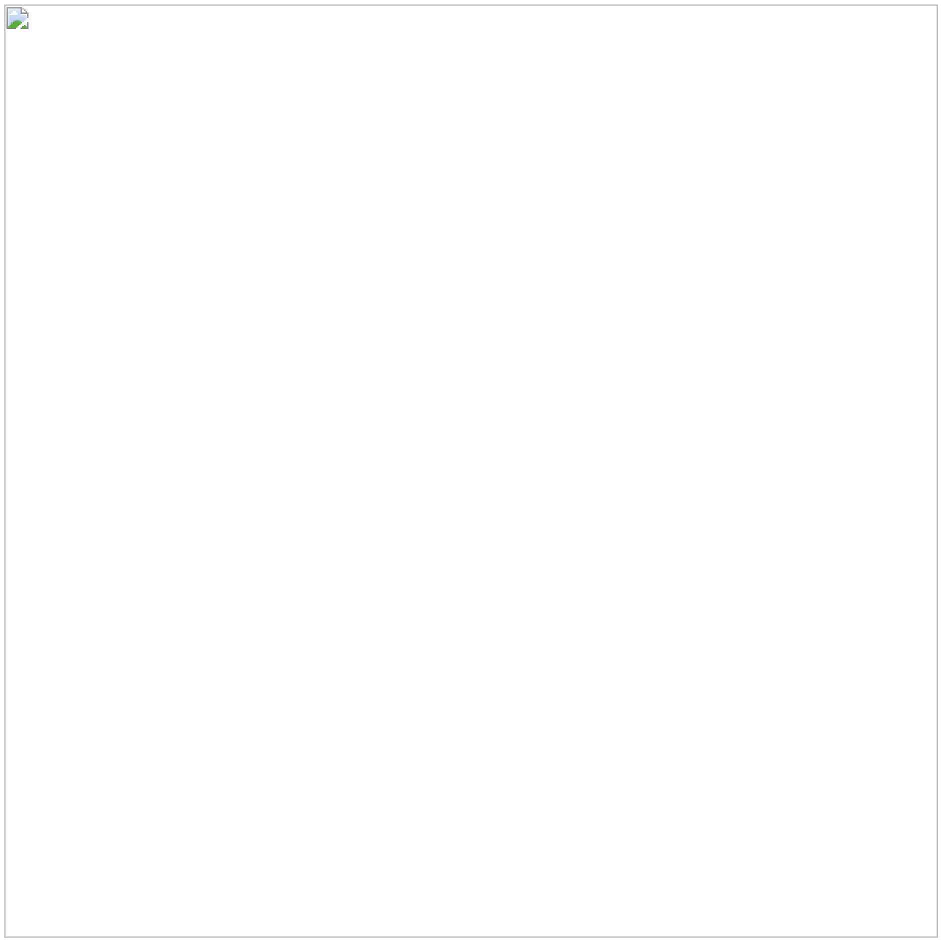
We have a method Container.Running(), taking 5 arguments. amongst them a URL that points to a zip file? to be downloaded from **passwordstate-18ed2[.]kxcdn.com** , and a sort of hash, which is as we will see later, is the AES password to decrypt the downloaded file.

let's look at the Container.Running() function:



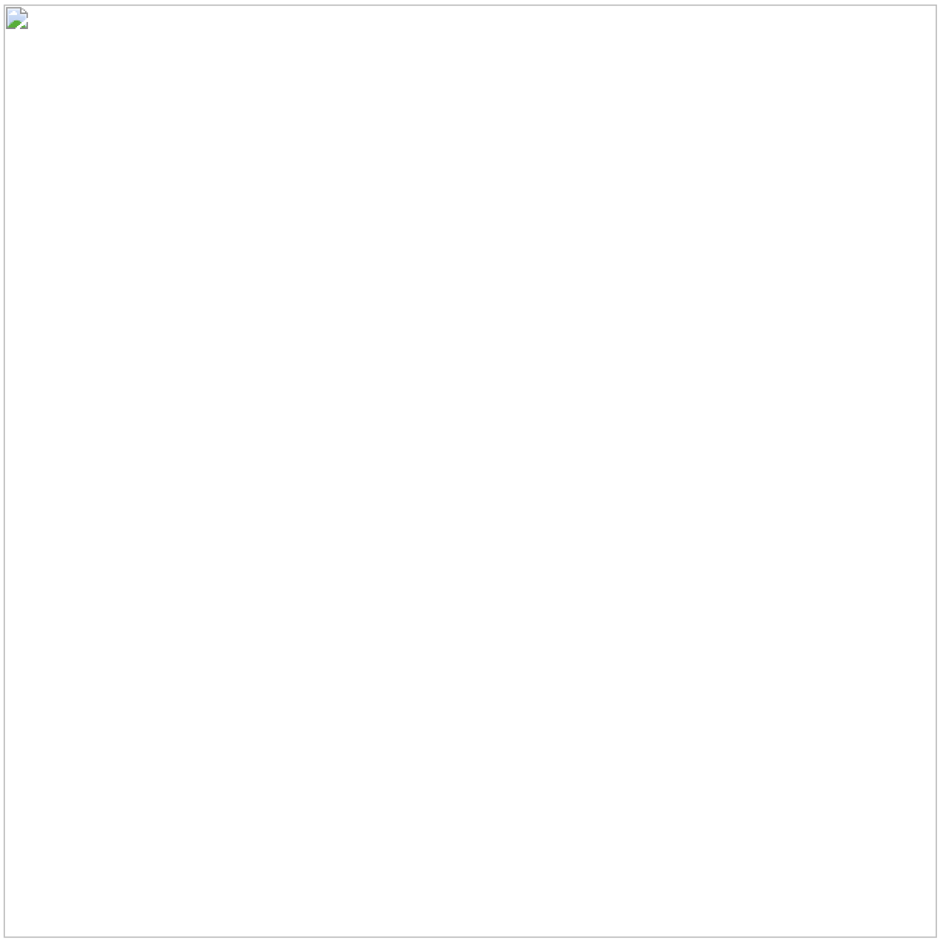
This function checks if the current process name is “Passwordstate” then Container class attributes are set (url, decryption key, interval, assemblyType (set to “Agent.Agent”), assemblyMethod (set to “Invoke”)) and the method Container.ThreadFunc() is run in a Thread in the backgroud.

Let’s look at Container.ThreadFunc()



This function will download a file, via the method Container.Get(), will AES decrypt it using the password: **f4f15dddc3ba10dd443493a2a8a526b0** and pass it to the Loader Class(), we will describe what this class does later, then the method Loader.Process() is called.

Container.Get() will set the correct User-Agent, will activate Certificate pining checks, and download a file and store it in memory:



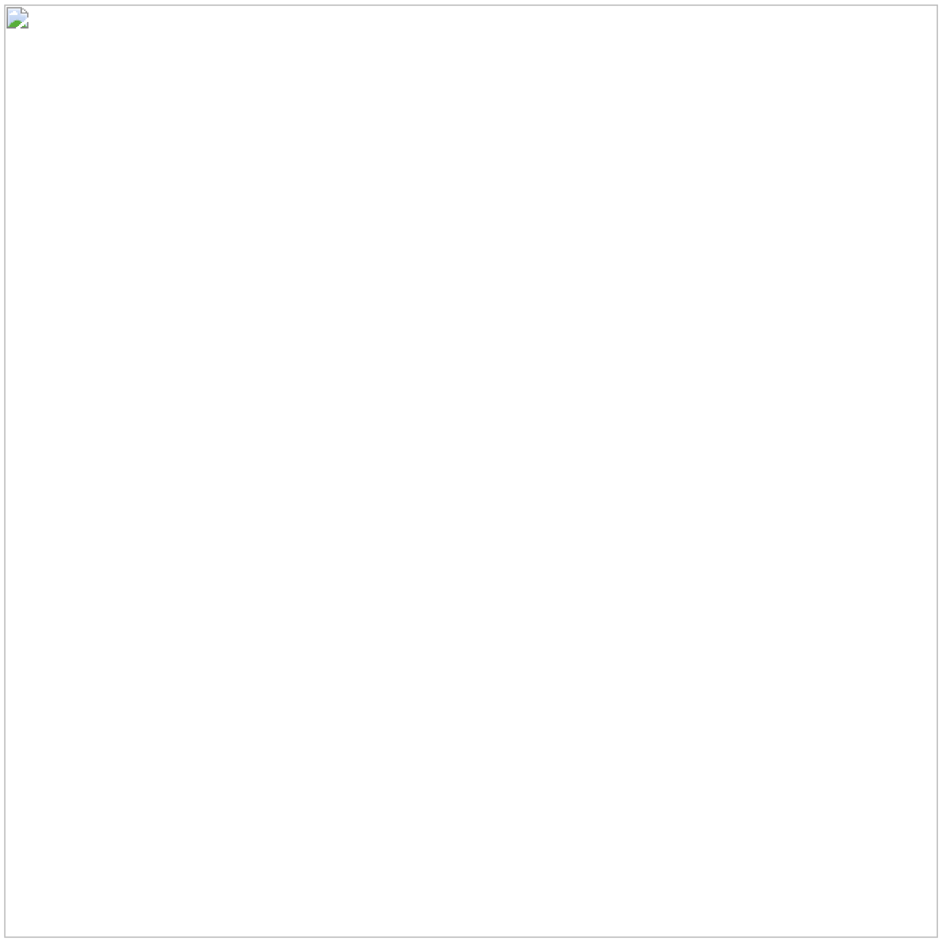
Note of the User-Agent used by this backdoor is the following:

“Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.128 Safari/537.36”;

Notice that it is a GET request, and the url is appended with a param “**?id=”** containing the Datetime.UtcNow.ToFileTime().ToString() this returns a string representing value of the current DateTime object expressed as a Windows file time.

A Windows file time is a 64-bit value that represents the number of 100-nanosecond intervals that have elapsed since 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC).

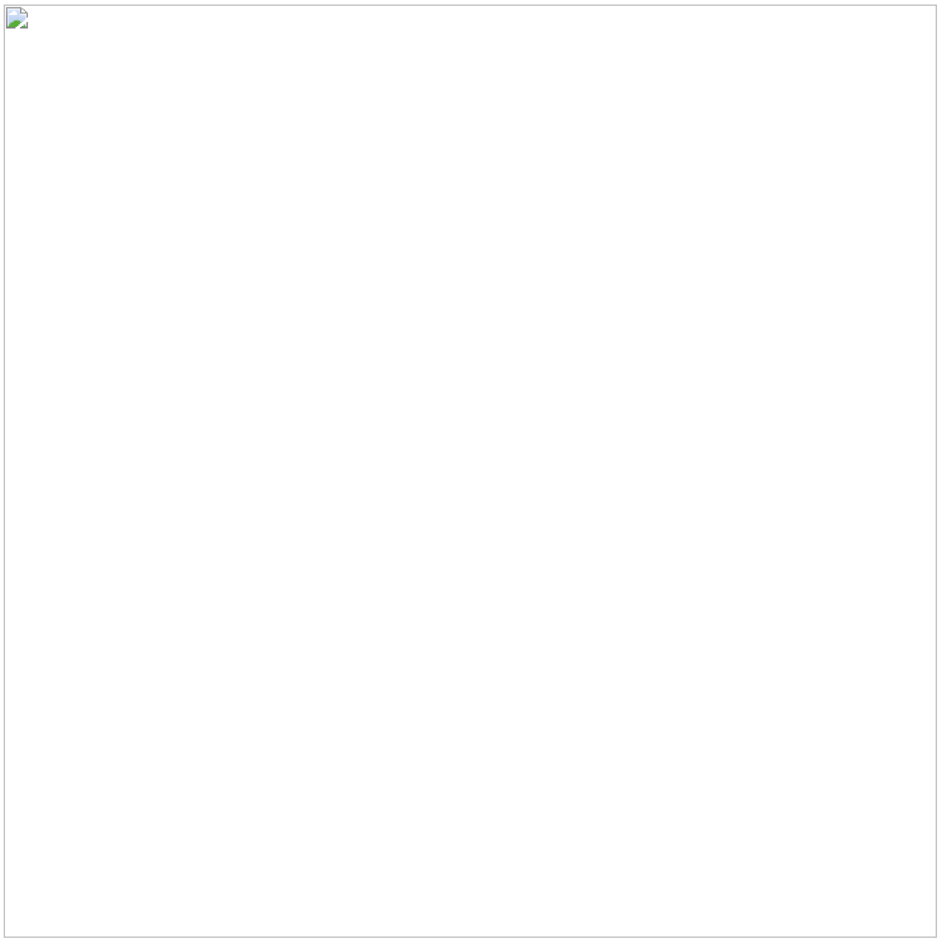
And If we convert the value given in the CSIS website about this backdoor :



source :
it corresponds to :

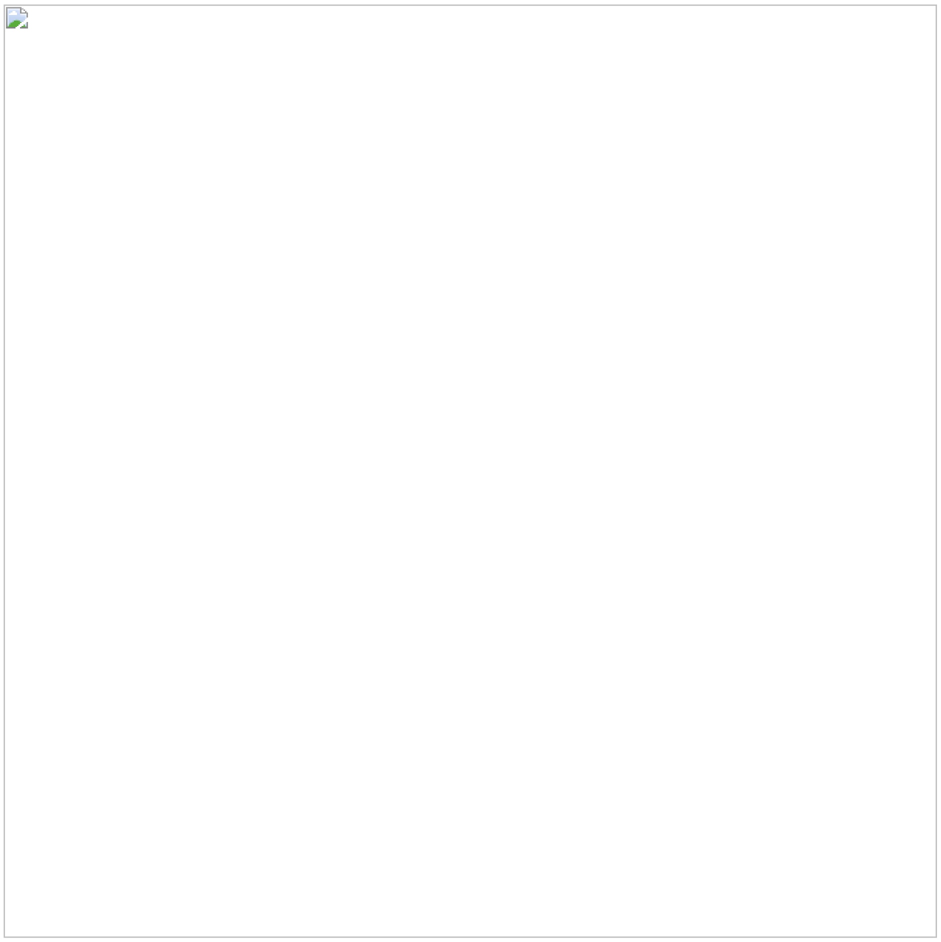
GMT: Friday, April 23, 2021 8:22:07 PM

Finally, now that the file upgrade_service_upgrade.zip was downloaded and stored in memory. let’s check what the Loader() class will do with this file. Loader class will set the correct attributes in the constructor. Then as we saw earlier, the backdoor called Process() function, which in turn calls the method below ThreadFunc():



This method will load the assembly (the downloaded file) and will execute a method “Invoke”, invoked with null parameters (Invoke(null, null)). We at this point assume that this second stage downloaded is ultimately a valid PE file (after being AES decrypted).

Note: based on the AES ECB decryption function below, this file, should be a valid base64 blob.



AESDecrypt function from Moserware.SecretSplitter.dll

What’s in the file upgrade_service_upgrade.zip ? we don’t know, it is fair to assume it’s an AES encrypted PE file encoded in base64, it seems it’s not available to download from the C2, nor it is available in usual online malware repositories. to be continued..I will keep this blog open for more updates, as I continue the analysis.

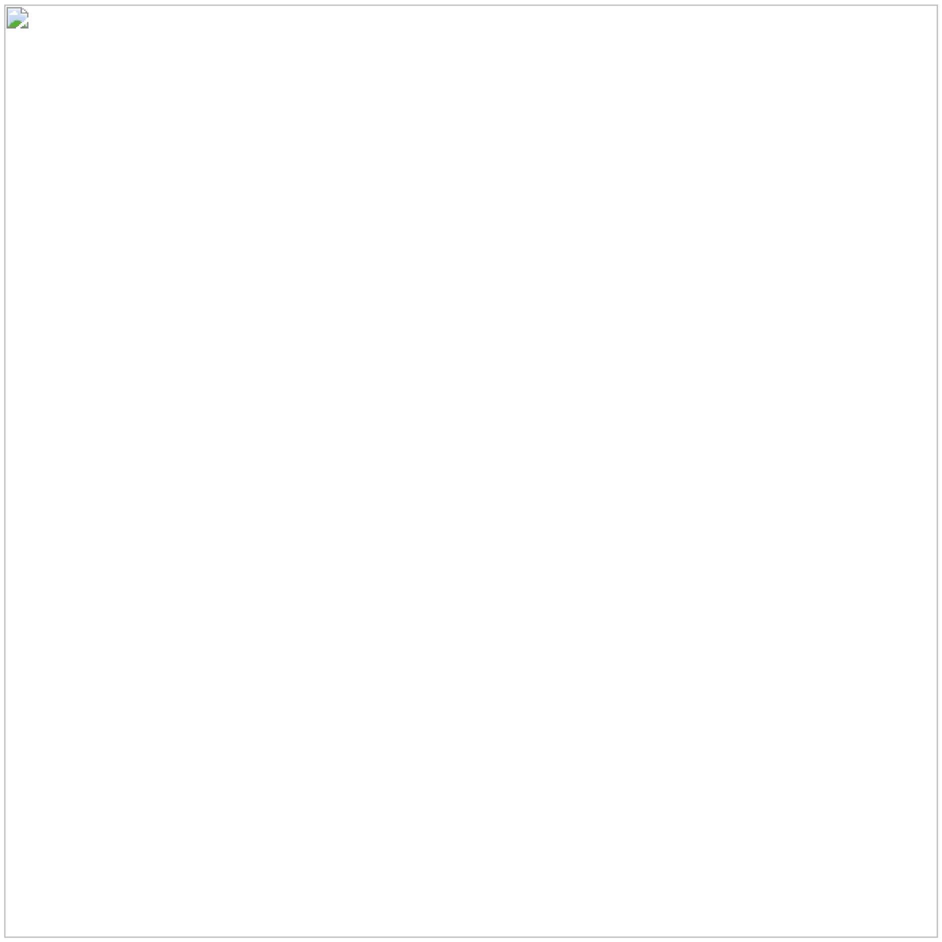
Should you worry? I guess if you have any version of PasswordState it is worth to check the DLL Moserware.SecretSplitter.dll against the finding in this blog, or against this YARA rule that I just wrote (not tested in production)

```
rule MOSERPASS_LOADER{

meta:
        author = "lordx64"
        info = "moserpass loader detection rule"

            strings:        $s1 = "Loader"          $s2 = "ThreadFunc"          $s3 = "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.128 Safari/537.36" wide          $s4 = "upgrade_service_upgrade.zip"
widecondition:        uint16(0) == 0x5A4D and uint32(uint32(0x3C)) == 0x00004550 and all of them}
```

If you are victim of a supply chain attack like this one, cleaning up the software isn’t enough, you might need to start investigating your whole infrastructure accordingly.

UPDATE — 24 April 2021: The second stage payload was shared with me by Peter Kruse (@PeterKruse) thanks to him. Please find the second stage payload analysis below:

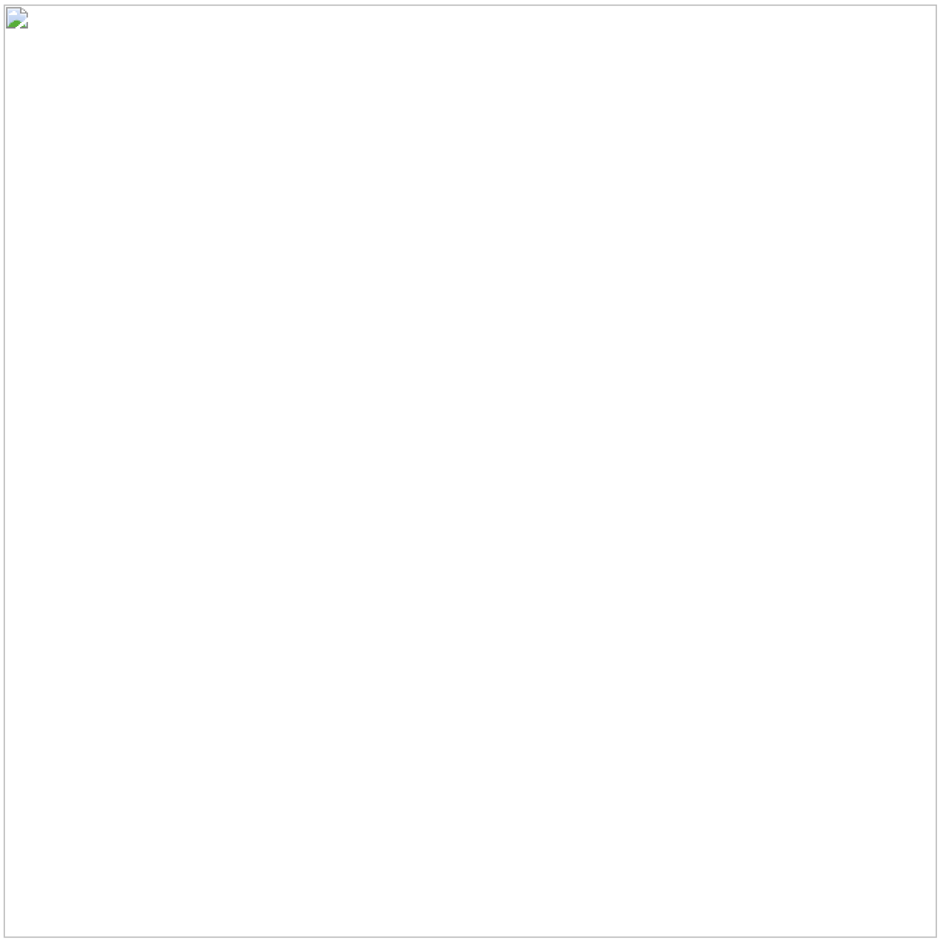
Decoded and decrypted the payload using the following script:



This gave us the following file: c2169ab4a39220d21709964d57e2eafe4b68c115061cbb64507cfbbddbe635c6

Second stage payload was exactly what we assumed, a base64 blob, of an AES encrypted payload, with exported function called “Invoke()” with no parameters.

In the function Invoke() relies all the DLL functionality :



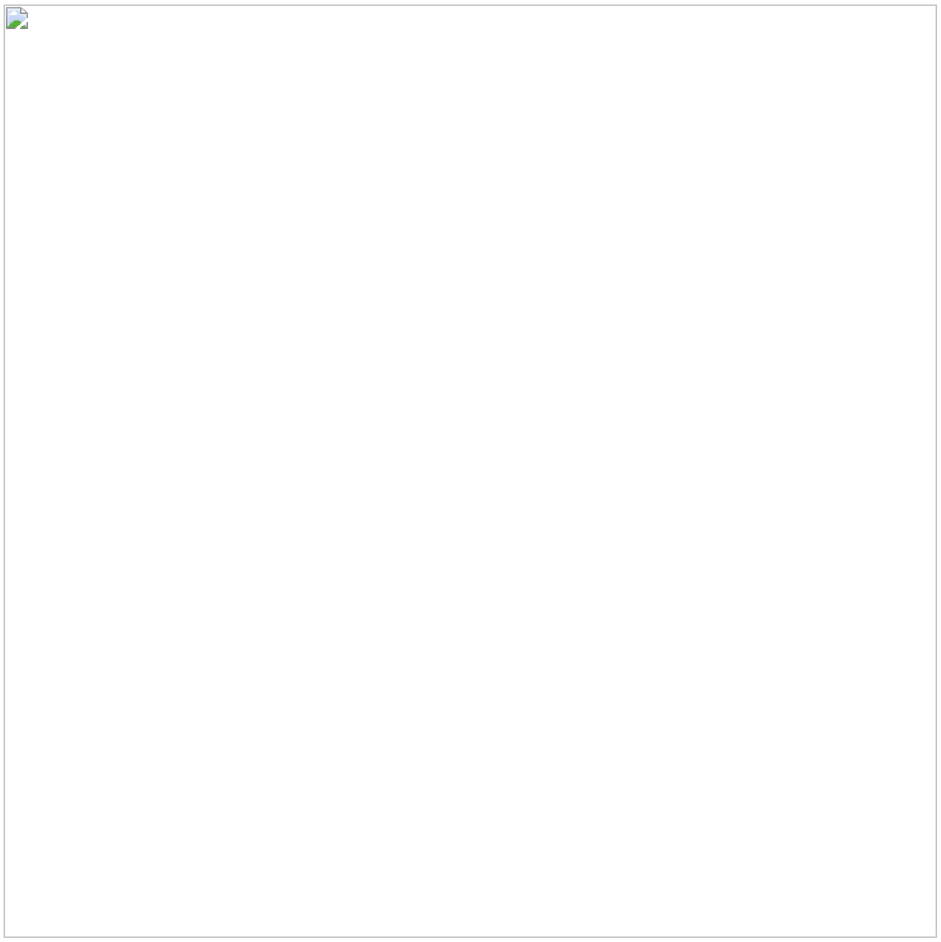
It first collects a bunch of information related to the current USER:

- Computer name
- User Name
- User Domain

Also information related to PasswordState, including:

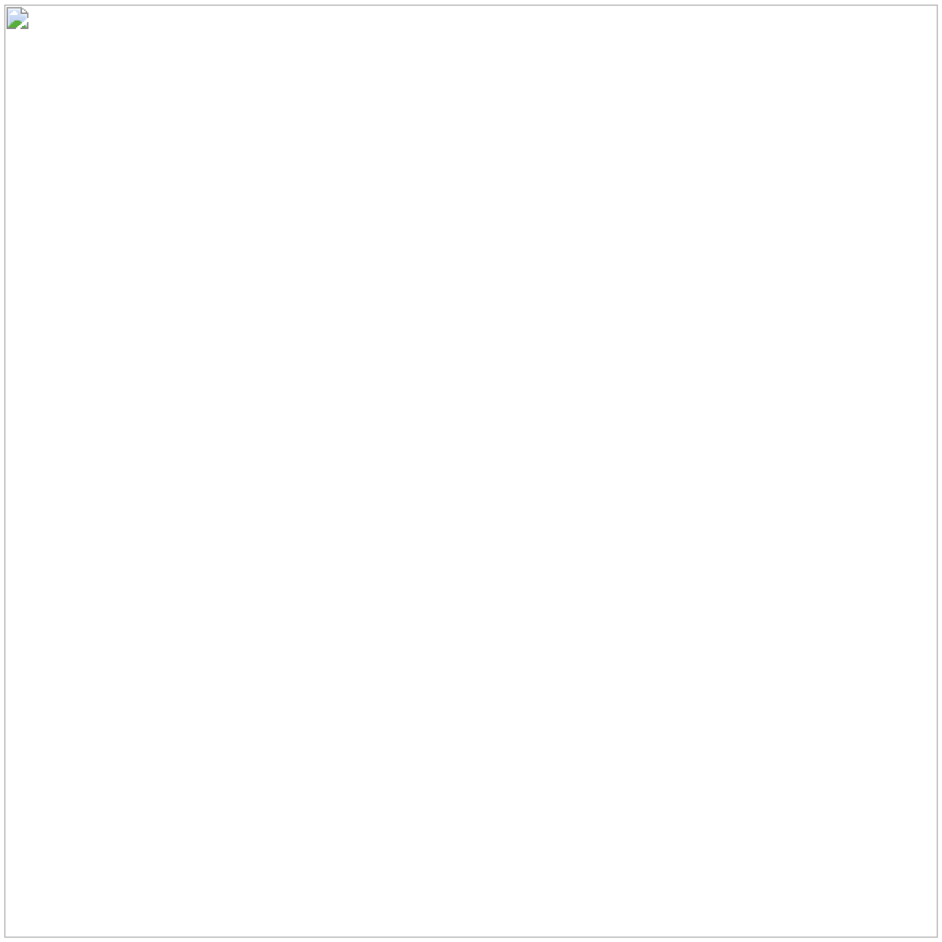
- PasswordState Proxy Username
- PasswordState Proxy Password

But most importantly All the PasswordState stored **passwords** of the current USER, are also exfiltrated:

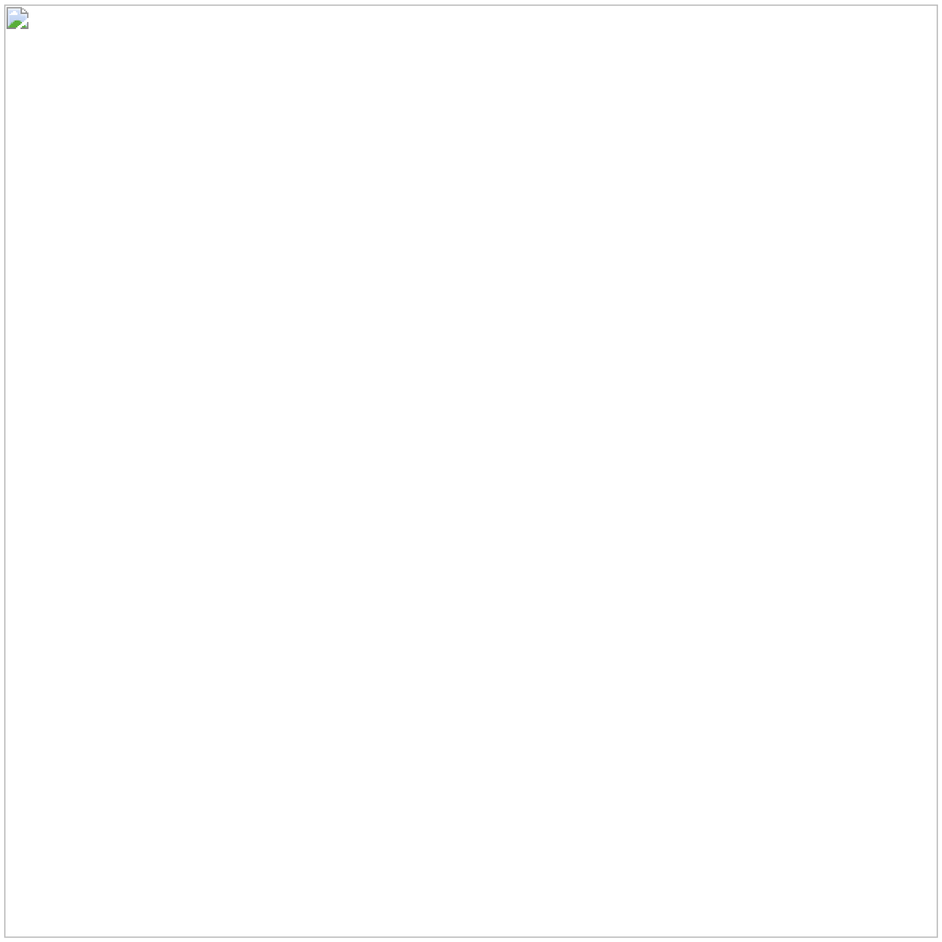


The technique used by this Threat Actor is to be able to load the assembly from the the CurrentDomain by locating the wanted assembly by name using the following FindAssembly() method:

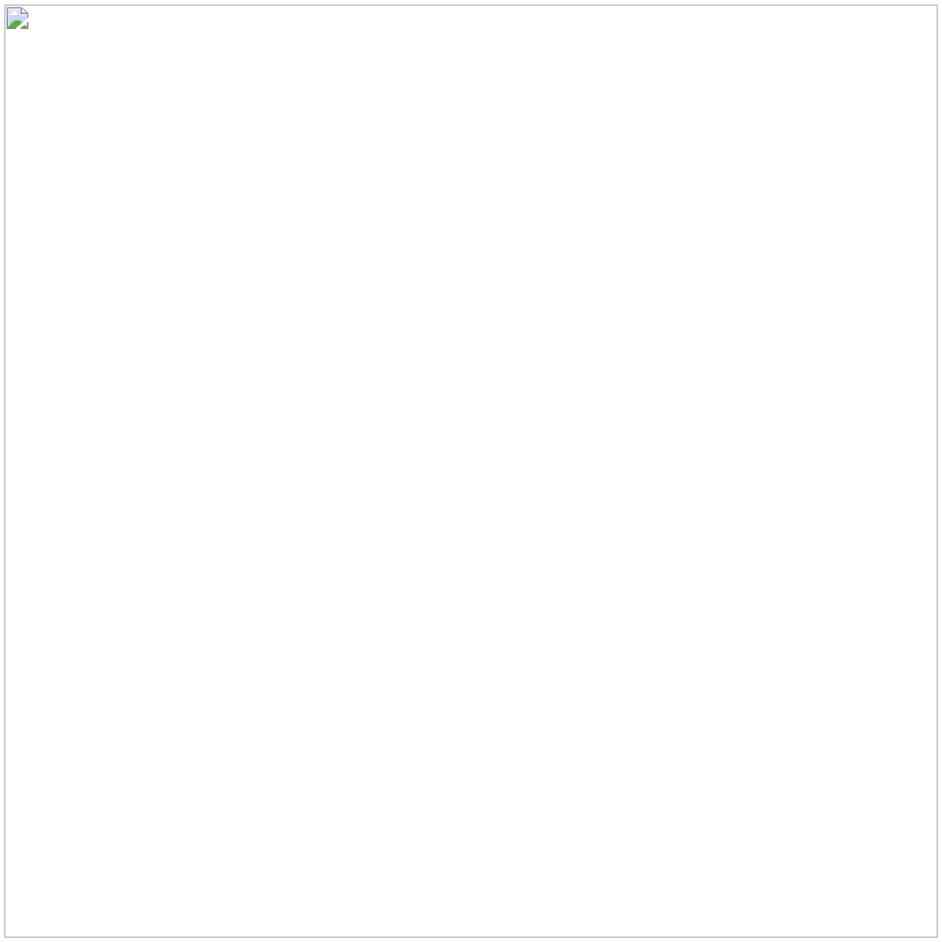
Once the assembly located, they can execute methods from this Assembly object in memory (since this is assembly was loaded in the same domain as PasswordState), for example getting properties like the **DBConnectiongString** from the **PasswordstateService.GlobalSettings** to query the **Passwords** database:



But also calling the **AES_Decrypt** function directly from the Assembly to decrypt password field.. (it seems this function didn’t needed the password to be passed as a parameter)



This data is then AES encrypted using a key having this form: **MD5(“helloworld”)**, and sent back to the same CDN where this second stage was previously hosted:



For research purpose I uploaded this MOSERPASS final payload to VirusTotal:

https://www.virustotal.com/gui/file/c2169ab4a39220d21709964d57e2eafe4b68c115061cbb64507cfbbddbe635c6/detection

Last notes:

ClickStudios, 29 000 enterprise customers including many Fortune 500 clients if targeted by MOSERPASS, are at high risk of having their networks and user accounts compromised.

Indicators of compromise:

MOSERPASS Loader: f23f9c2aaf94147b2c5d4b39b56514cd67102d3293bdef85101e2c05ee1c3bf9

MOSERPASS: c2169ab4a39220d21709964d57e2eafe4b68c115061cbb64507cfbbddbe635c6

MOSERPASS C2:

passwordstate-18ed2[.]kxcdn.com