MetaStealer Part 2, Google Cookie Refresher Madness and Stealer Drama

russianpanda.com/2023/12/28/MetaStealer-Part-2/

Stealer's World of Drama

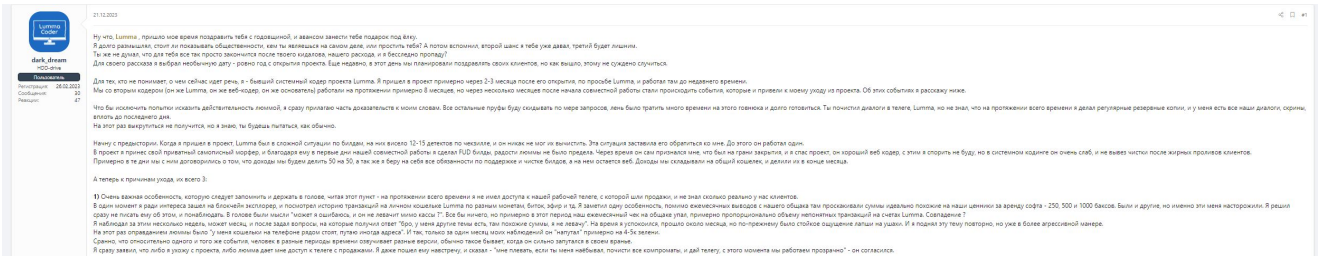
Previously, I wrote a blog going through some of MetaStealer's functionalities and did a brief comparison with Redline since they are both very similar but, at the same time, different. You might say that all stealers are the same because they have one purpose - to steal. However, each of them is somewhat different from the others, even if they borrowed the code from their predecessors.

Every stealer tries to be better than the other one despite having similar code and functionality. What is considered a good stealer? The stealer has a low detection rate and a high rate of successful infection, or what we call "отстук" in Russian. Stealers such as Redline, Metastealer, Raccoon Stealer, Lumma, RisePro, and Vidar have earned their names in the stealer market. Below is the list of top stealers' whose logs are being sold on RussianMarket.

The popularity of mentioned stealers among users, mainly those developed by native Russian speakers, could be attributed to the ease of communication and support in their native language. As you might have noticed, stealers are notably prevalent among Russian-speaking communities. The ability to interact in one's native language - whether it is to request new features, report issues, or inquire about the functionality of the stealer - significantly simplifies the process compared to the effort required for translation into English. This linguistic accessibility potentially broadens the client base, offering the stealer more opportunities to attract additional users.

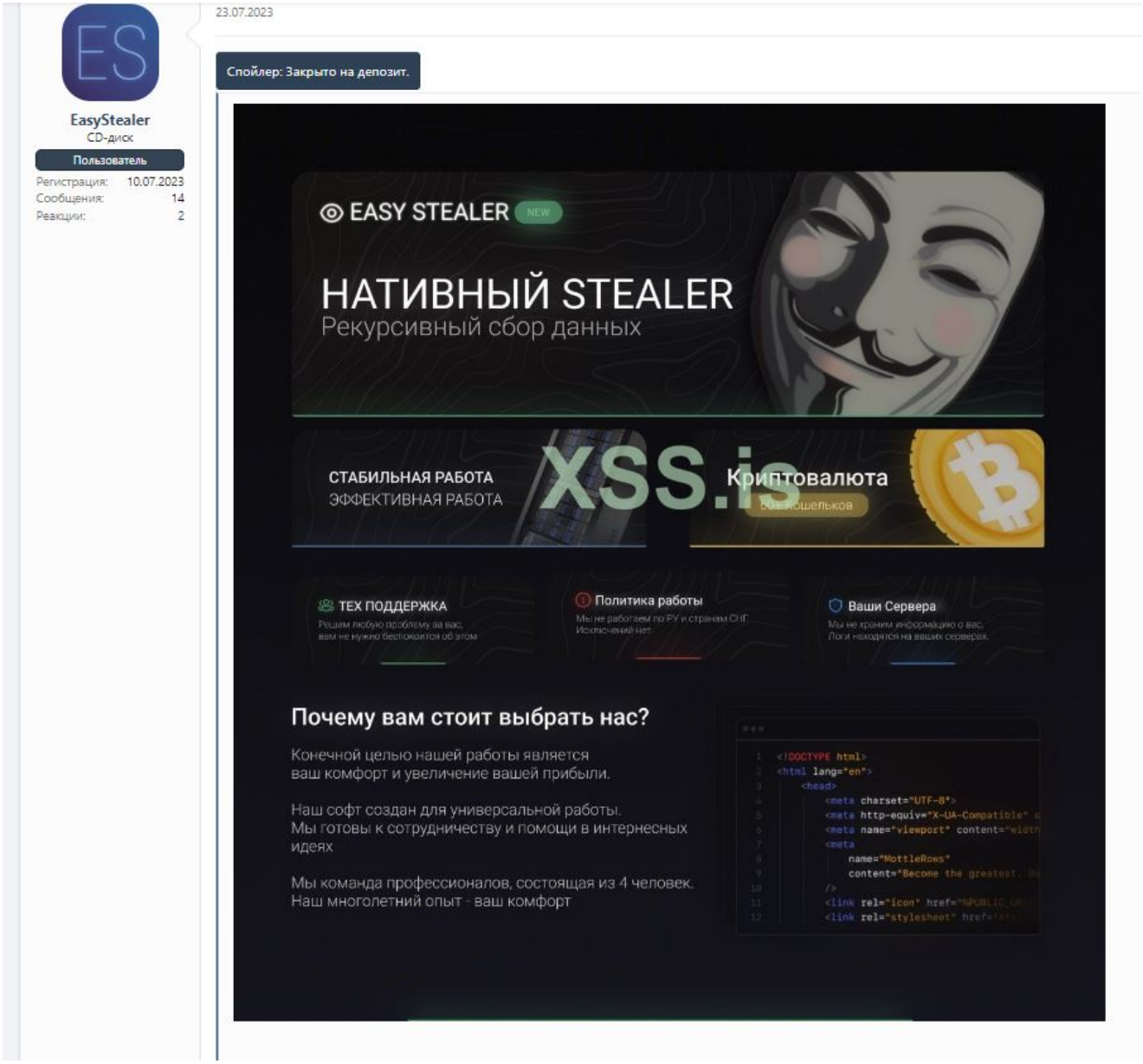
The world of stealers is rife with drama, much like any other corner of the cybercriminal ecosystem. I was recently informed about an incident related to the Santa Barbara topic on XSS forums. This topic was created by one of Lumma's former coders, coinciding with

Lumma’s one-year anniversary. To put it briefly, Lumma’s founder did not adequately recognize or compensate the coder’s contributions, leading to dissatisfaction and underpayment.



Another drama story: some of you might know how Aurora Stealer left the market before their infamous botnet release; some users deposited money for the botnet and never got it back, of course. Now, Aurora has become a meme within the stealer’s community.

In July 2023, an advertisement was posted on XSS forums for a new stealer written in Golang, known as “EasyStealer”, then the rumors started spreading among the stealer’s community that this was the work of an Aurora developer, now the stealer is nowhere to be found.



Does all of this impact the sales of stealers? Not at all. People continue to purchase stealers as long as their functionality meets their requirements.

Google Cookie Refresher “feature” or a “0day”

So, you’ve likely heard about the ongoing Google “0day” vulnerability, which allows attackers to obtain fresh cookies, granting them “indefinite” access to Google accounts. It is a rather convenient “feature,” isn’t it? However, it is also quite dangerous because an attacker would be able to get fresh cookies to Google accounts each time the old ones expire.

As @g0njxa mentioned, the feature is abused by many stealers, including RisePro, MetaStealer, Whitesnake, StealC, Lumma, Rhadamanthys, and Meduza. Additionally, as of December 29th, Vidar Stealer has implemented this feature.

The question of how long it will take Google to respond to this issue remains unanswered. However, this situation presents even more opportunities for stealers to take advantage of the vulnerability.

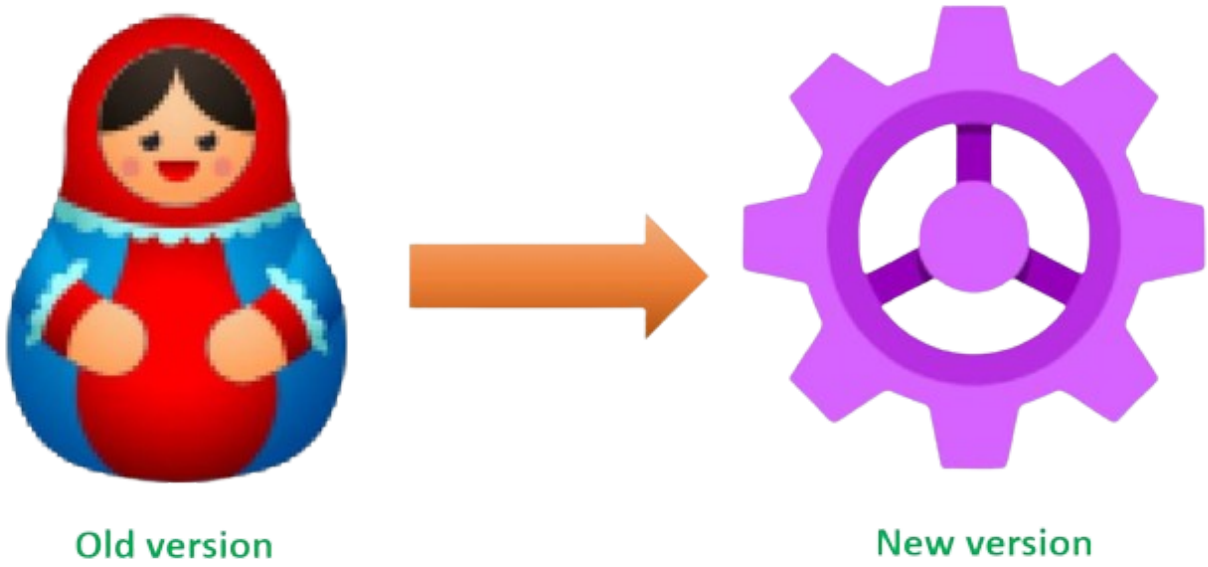
The reason why I brought this up is how easily it can be exploited with just a few lines of Python code that includes the decrypted token value, account ID, and the proper request to the server if some people are curious enough to find out. Although, certain parameters need to be slightly tweaked from the server's response to make it work. Here is my video with proof-of-concept on how it works on a high level. I have created a video demonstrating the proof-of-concept at a high level. For ethical reasons, I will not delve into the technical details of the POC.

MetaStealer Part 2: Technical Analysis

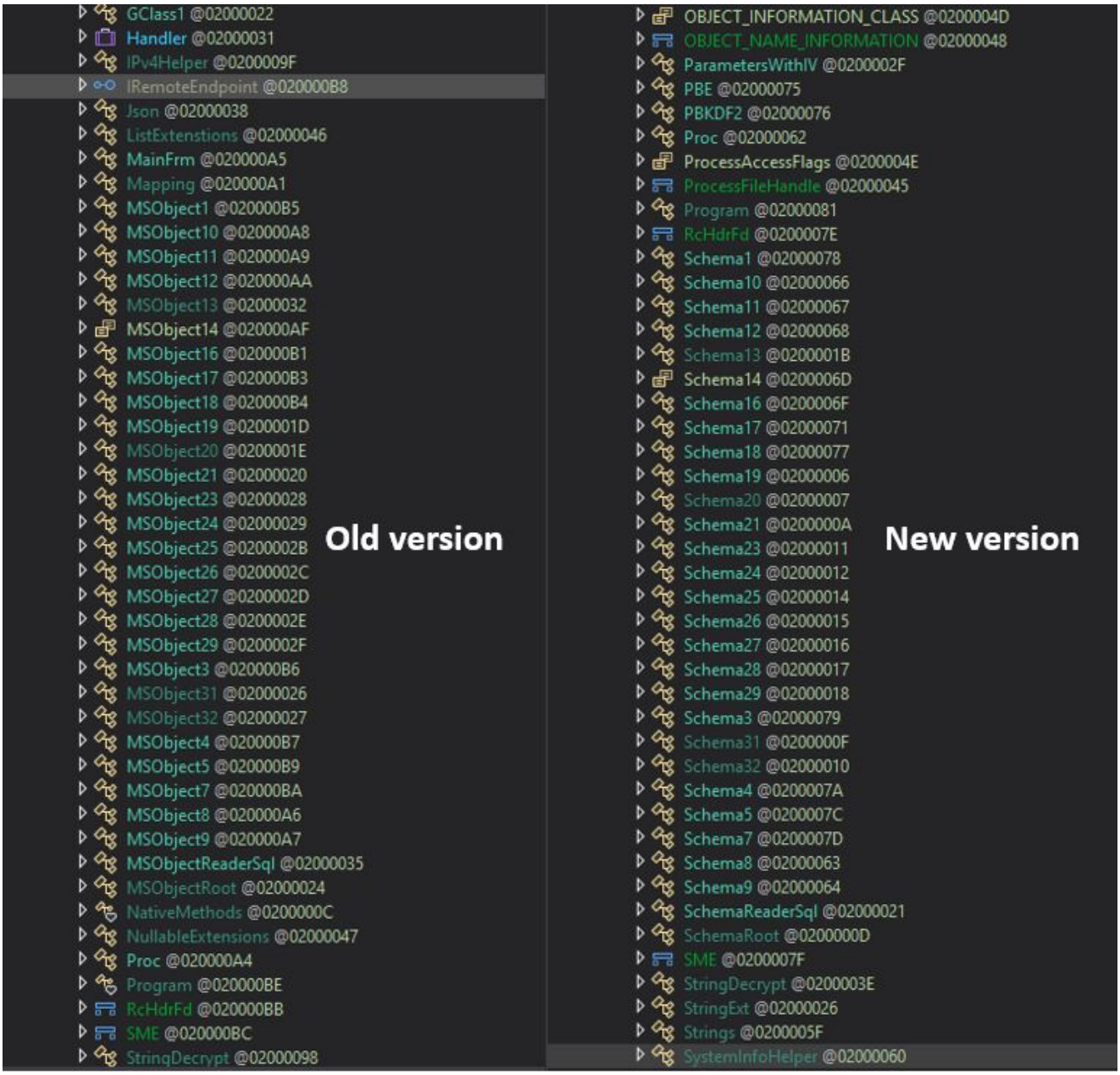
In November 2023, I released the writeup on MetaStealer. However, soon after its release, the malware developer made another update that changed the class names, string encryption algorithm, binary description, and file icon.

MetaStealer new version is approximately 368KB in size with the binary description **Cavils Corp. 2010** (the previous one was **METRO 2022 Dev**).

The logo change:



If previously, MetaStealer used “Entity” for class names; now it’s using “Schema” and “TreeObject” to store data and configurations instead of **MSValue**.



Instead of string replacement operations, it now accesses a decrypted string from an array based on the given index. For example, below, where it uses **ManagementObjectSearcher** class to query system management information. The constructor of **ManagementObjectSearcher** takes two parameters: a WMI query path and a query string, for example **"ROOT\SecurityCenter: SELECT * FROM AntivirusProduct"**.

```
237     // Token: 0x060001A0 RID: 416 RVA: 0x000132E4 File Offset: 0x000114E4
238     public static List<string> GetVs()
239     {
240         List<string> list = new List<string>();
241         try
242         {
243             foreach (string text in Strings.Get(42).Split(new char[] { '|' }))
244             {
245                 try
246                 {
247                     using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher(Strings.Get(43), Strings.Get(44) + text))
248                     {
249                         using (ManagementObjectCollection managementObjectCollection = managementObjectSearcher.Get())
250                         {
251                             foreach (ManagementBaseObject managementBaseObject in managementObjectCollection)
252                             {
253                                 try
254                                 {
255                                     string text2 = managementBaseObject[Strings.Get(45)] as string;
256                                     if (!list.Contains(text2))
257                                     {
258                                         list.Add(text2);
259                                     }
260                                 }
261                                 catch
262                                 {
263                                 }
264                             }
265                         }
266                     }
267                 }
268                 catch
```

100 %

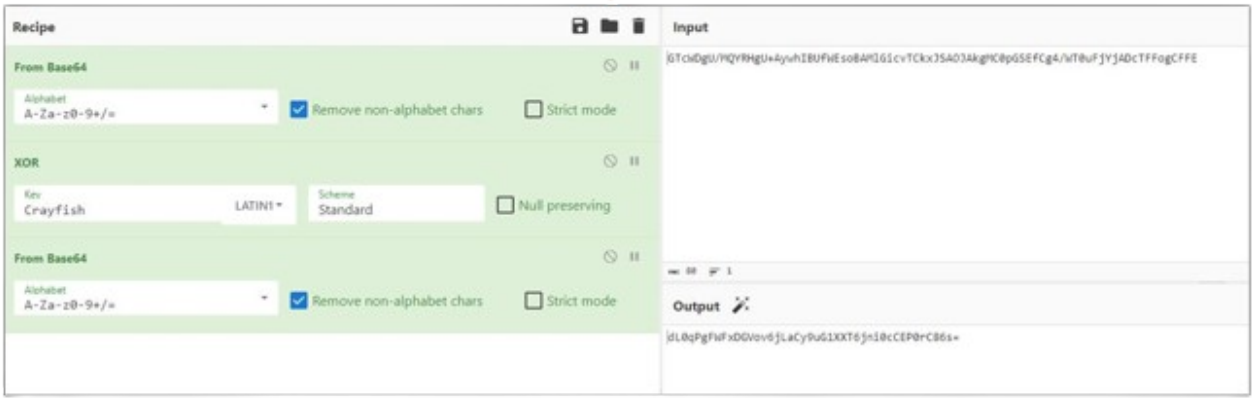
Locals

Name	Value	Type
Strings.Get returned	@"ROOT\SecurityCenter"	string
Strings.Get returned	"SELECT * FROM "	string
string.Concat returned	"SELECT * FROM AntivirusProduct"	string
list	Count = 0x00000000	System.Collections.Generic.List<st...
array	{string[0x00000003]}	string[]
i	0x00000000	int
text	"AntivirusProduct"	string
managementObjectSearcher	{System.Management.ManagementObjectSearcher}	System.Management.Managemen...
managementObjectCollection	null	System.Management.Managemen...
managementObjectEnumerator	null	System.Management.Managemen...
managementBaseObject	null	System.Management.Managemen...

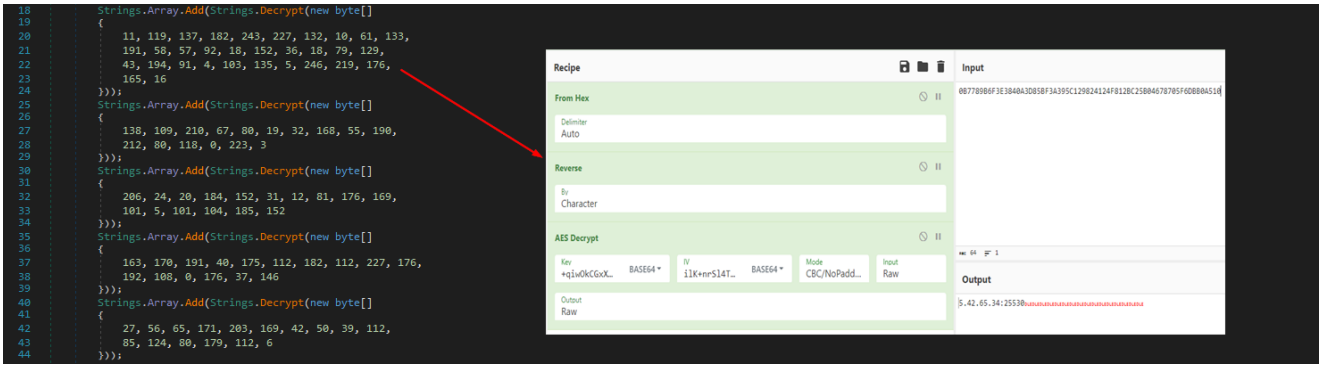
The new string decryption algorithm works the following way:

First, the base64-encoded string gets base64-decoded and XOR'ed with the hardcoded key (in our example, it is **Crayfish**); the XOR'ed string then gets base64-decoded again.

```
8   // Token: 0x0200005F RID: 95
9   public static class Strings
10  {
11      // Token: 0x06000195 RID: 405 RVA: 0x00011868 File Offset: 0x0000FA68
12      static Strings()
13      {
14          Strings.Keys[0] = "Crayfish";
15          Strings.Keys[1] = StringDecrypt.Read("GTcWDgU/HQYRHgU+AywhIBUfWEsoBAM1G1cvTCkxJSAOJAkgMC0pGSEfCg4/WT0uFjYjADcTFFogCFFE", Strings.Keys[0]);
16          Strings.Keys[2] = StringDecrypt.Read("GSY7SDAuRjAmOg0fWQIDBBdADkg0LgRZDCorFjwTQ1E=", Strings.Keys[0]);
17          Strings.Array = new List<string>();
```



- Each XOR'ed and base64-decoded string is assigned as an AES key and IV (Keys[1] and Keys[2]).
- The encrypted byte arrays are then reversed and decrypted using the keys and IV mentioned above



To save us some time, we can use the dynamic approach to decrypt the strings using **dnlib**. The wonderful approach was detailed by @n1ghtw0lf in this blog. Also, I want to thank @cod3nym for amazing tips when it comes to dealing with .NET shenanigans!

Here are the steps to decrypt the strings:

> We will use **dnlib**, a library for reading and writing .NET assemblies to load a .NET module and assembly from a given file path.

```
def load_net_module(file_path):
    return ModuleDefMD.Load(file_path)

def load_net_assembly(file_path):
    return Assembly.LoadFile(file_path)
# Main script
module = load_net_module(file_path)
assembly = load_net_assembly(file_path)
```

> We will define the decryption signature (**decryption_signature**) to identify methods that are likely used for decryption. This signature includes the expected parameters and return type of the decryption methods.

```
decryption_signature = [
    {"Parameters": ["System.Int32"], "ReturnType": "System.String"}
]
```

> We will search the loaded assembly for methods that match the defined decryption signature.

```
def find_decryption_methods(assembly):
    suspected_methods = []
    flags = BindingFlags.Static | BindingFlags.Public | BindingFlags.NonPublic
    for module_type in assembly.GetTypes():
        for method in module_type.GetMethods(flags):
            for sig in decryption_signature:
                if method_matches_signature(method, sig):
                    suspected_methods.append(method)
    return suspected_methods
```

Finally, we will invoke the suspected decryption methods by scanning the assembly's methods for calls to the suspected decryption methods, extracting the parameters passed to these methods, and invoking the decryption methods with the extracted parameters.

```
def invoke_methods(module, suspected_methods):
    results = {}
    for method in suspected_methods:
        for module_type in module.Types:
            if not module_type.HasMethods:
                continue
            for m in module_type.Methods:
                if m.HasBody:
                    for insnIdx, insn in enumerate(m.Body.Instructions):
                        if insn.OpCode == OpCodes.Call:
                            called_method_name = str(insn.Operand)
                            if method.Name in called_method_name:
                                params = extract_parameters(m.Body.Instructions,
insnIdx, method)
                                if len(params) == len(method.GetParameters()):
                                    try:
                                        result = invoke_method_safely(method, params)
                                        if result is not None:
                                            location = f"{module_type.FullName}.
{m.Name}"
                                            results[location] = result
                                    except Exception as e:
                                        None
    return results
```

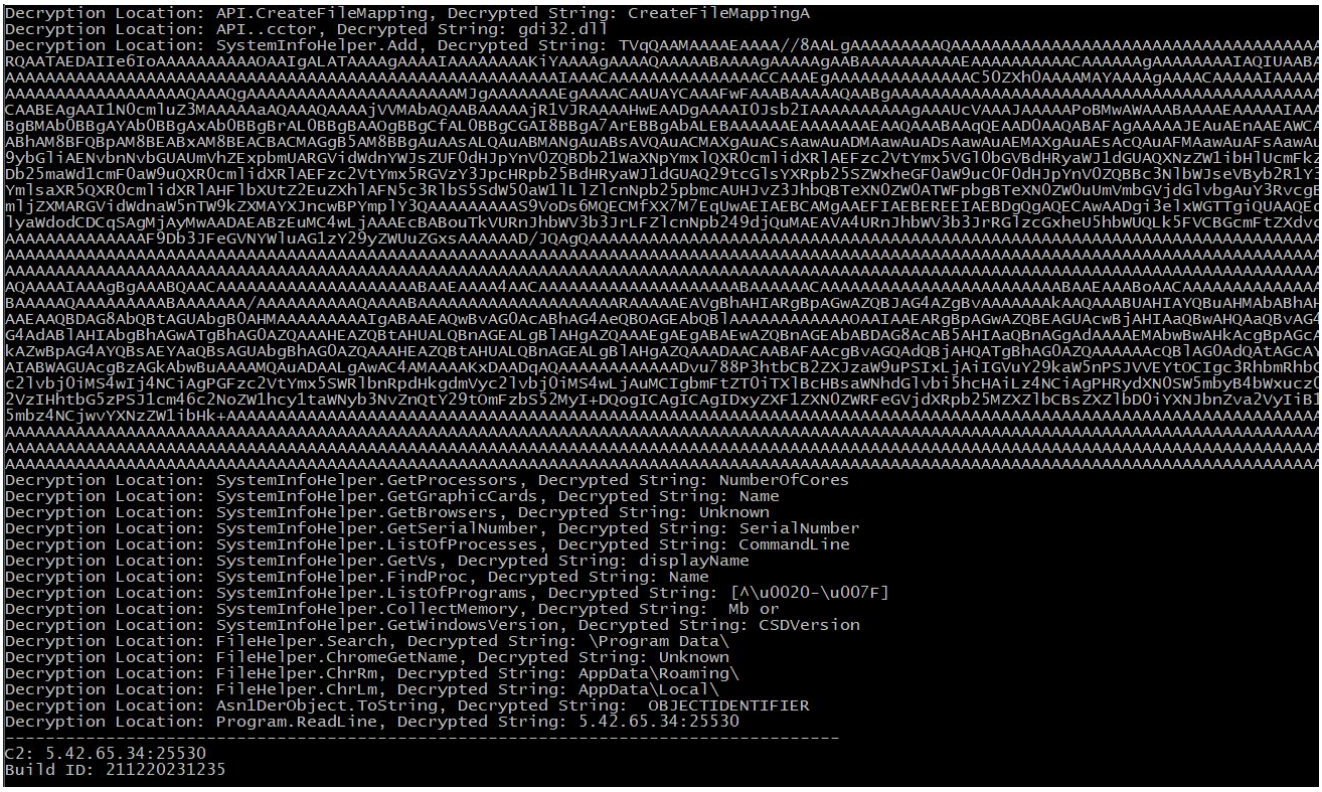
We will also include the logic to handle different types of parameters, such as integers and strings. It uses **get_operand_value** to extract values from method instructions based on their type.

```
def get_operand_value(insn, param_type):
    if "Int32" in param_type and insn.IsLdcI4():
        return Int32(insn.GetLdcI4Value())
    elif "String" in param_type and insn.OpCode == OpCodes.Ldstr:
        return insn.Operand
    return None
```

You can access the full script here.

Note: Please run the script strictly in a sandbox environment.

The output of the script (tested on the deobfuscated sample *MD5: e6db93b513085fe253753cff76054a2a*):



You might have noticed an interesting base64-encoded string in the output above.

Upon decoding, we receive a .NET executable **qemu-ga.exe** (MD5: e6db93b513085fe253753cff76054a2a).

Now, an interesting moment: MetaStealer writes that executable to the Startup after successfully receiving the configuration from the C2 server and collecting user information. The executable does not do anything but enters the indefinite loop that alternates between sleeping for 100 seconds and waiting for user input without doing anything with that input.

Another addition to the new version of MetaStealer is the username and computer name check to avoid sandbox environments; if any of the usernames/computer names are found in the list, the stealer process will exit.

List of computer names:

```
        {
                "bee7370c-8c0c-4", "desktop-nakffmt", "win-5e07cos9alr", "b30f0242-
1c6a-4", "desktop-vrsqlag", "q9iatrkprh", "xc64zb", "desktop-d019gdm", "desktop-
wi8clet", "server1",
                "lisa-pc", "john-pc", "desktop-b0t93d6", "desktop-1pykp29", "desktop-
1y2433r", "wileypc", "work", "6c4e733f-c2d9-4", "ralphs-pc", "desktop-wg3myjs",
                "desktop-7xc6gez", "desktop-5ov9s0o", "qarzhrdbpj", "oreleepc",
"archibaldpc", "julia-pc", "d1bnjkfvlh", "compname_5076", "desktop-vkeons4", "NTT-
EFF-2W11WSS"
        };
```

List of usernames:

```
        {
                "wdagutilityaccount", "abby", "peter wilson", "hmarc", "patex",
"john-pc", "rdhj0cnfevzx", "keecfmwgj", "frank", "8nl0colnq5bq",
                "lisa", "john", "george", "pxmduopvyx", "8vizsm", "w0fjuovmccp5a",
"lmvwjj9b", "pqonjhvwexss", "3u2v9m8", "julia",
                "heuerzl", "harry johnson", "j.seance", "a.monaldo", "tvm"
        };
```

Detection Rules

You can access Yara rules here.

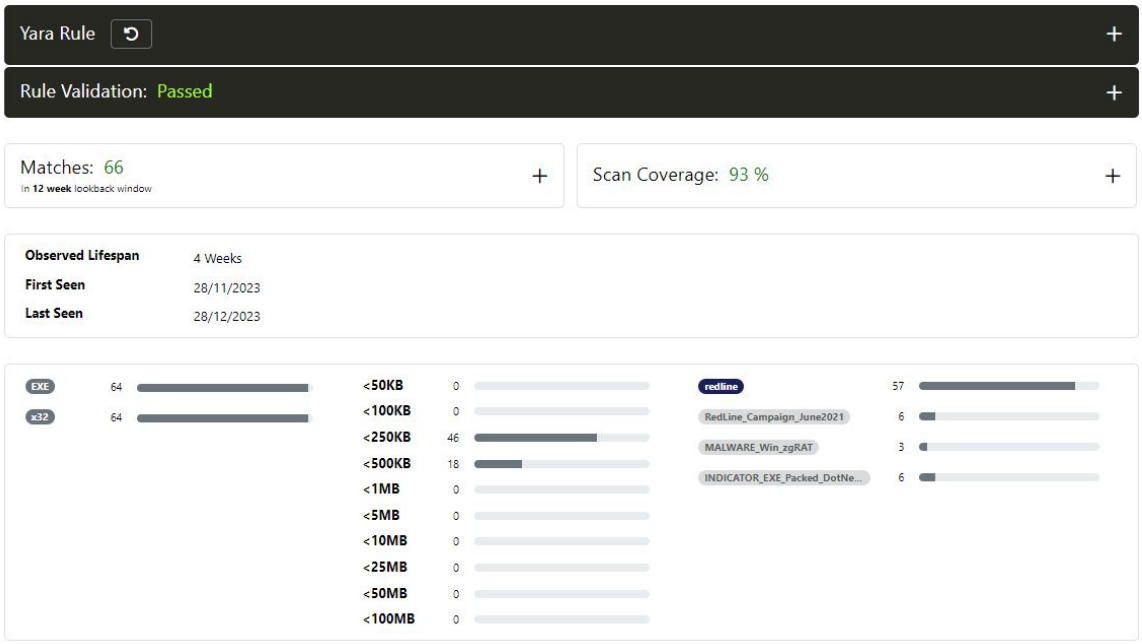
You can access Sigma rules here.

Indicators of Compromise

Name	Indicator

Name	Indicator
MetaStealer	e6db93b513085fe253753cff76054a2a
MetaStealer	a8d6e729b4911e1a0e3e9053eab2392b
MetaStealer	b3cca536bf466f360b7d38bb3c9fc9bc
C2	5.42.65[.]34:25530

For more samples, please refer to the result of my Yara scan on UnpacMe.



Reference

https://x.com/g0njxa/status/1739689195403100336?s=20

https://russianpanda.com/2023/11/20/MetaStealer-Redline's-Doppelganger

https://n1ght-w0lf.github.io/tutorials/dotnet-string-decryptor

https://twitter.com/cod3nym

https://www.unpac.me/yara/results/f87b8452-ba6d-4c8b-8adb-1ba3986eb4d9#

https://github.com/RussianPanda95/Configuration_extractors/blob/main/metastealer_string_decryptor.py

https://github.com/RussianPanda95/Yara-Rules/tree/main/MetaStealer

https://github.com/RussianPanda95/Sigma-Rules/blob/main/MetaStealer/suspicious_qemu_file_creation.yaml

Previous Post

Pure Logs Stealer Fails to Impress

Next Post