

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

By Ayush Anand

Published: 2020-06-28 · Archived: 2026-04-05 12:58:12 UTC

June 28, 2020

[Adwind](#), [binwalk](#), [Bytecode Viewer](#), [CVE-2020-1464](#), [file](#), [Glueball](#), [IoCs](#), [JAR](#), [Java](#), [MSI](#), [RAT](#), [Ratty](#), [xxd](#), [Yara](#), [ZIP](#)

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.

CONTENTS

1. [Overview of ZIP, JAR & MSI file format](#)
2. [How does it work?](#)
3. [Analysis of JAR appended to signed MSI files using Ratty RAT](#)
4. [Timeline](#)
5. [Conclusion](#)
6. [Yara Signature](#)
7. [Indicator of Compromise](#)
8. [References](#)

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***.

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 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.

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.

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.

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
```

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.

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.

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



rule jar_in_msi
{
meta:
description = "Detect jar appended to MSI"
author = "Securityinbits"
date = "2020-06-14"
reference = "https://twitter.com/Securityinbits/status/1271406138588708866"
hash_1 = "13a4072d8d0eba59712bb4ec251e0593"
hash_2 = "63bed40e369b76379b47818ba912ee43"
hash_3 = "85eb931d0d27179ae7c13085fb050b11"
strings:
\$msi_magic = { D0 CF 11 E0 A1 B1 1A E1 }
//To detect zip Local file header(lfh) & End of central directory record(eocd)
\$s_zip_magic_lfh = { 50 4B 03 04 }
\$s_zip_magic_eocd = { 50 4B 05 06 }
\$s_jar = "META-INF/MANIFEST.MF"
\$s_java_class = ".class"
condition:

	\$msi_magic at 0 and filesize > 60KB and all of (\$s_*)
	}

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  
80908e5e21c3aff7e8bcacddb99e02e 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
```

8. References

Thanks for reading. Feel free to connect with me on or [LinkedIn](#) for any suggestions or comments.

For more updates and exclusive content, subscribe to our newsletter. Stay sharp. Keep defending. 😊

Join 150+ subscribers who get 0x1 actionable security bit every week.

Source: <https://www.securityinbits.com/malware-analysis/interesting-tactic-by-ratty-adwind-distribution-of-jar-appended-to-signed-msi/>