An In-Depth Analysis of Android's Java Class Library: its Evolution and Security Impact

Timothé Riom, Alexandre Bartel

Software Engineering and Security

Umeå Universitet

this work has been supported by Kempestiftelserna

What version of Java is used in Android ?

What version of Java is used in Android?

- Java Class Library:
  - Android applications can be written in Java and compiled into Dalvik bytecode[1]
  - The Java classes (like `java.lang.string`) at the Core for interpretation and execution (Runtime or **Android RT**), are grouped in a component called *libcore*.
- OpenJDK
  - Since Android 7 (2016), switch from Apache Harmony (Google had to maintain since 2011) for OpenJDK.

[1] Now Optimized Dex and compiled AOT

What version of Java does Android uses ?

- OpenJDK change of release and lifecycle policy in 2018

LTS Versions	End of Active Support	End of support Security Updates
OpenJDK-1.7[1]	-	June 2020
OpenJDK-1.8[1]	-	November 2026
OpenJDK-11[1]	-	October 2024
OpenJDK-17	October 2027	Septembre 2029
OpenJDK-21	December 2029	-

[1] Red Hat stewardship

We asked around:

- ChatGPT

The following table shows the versions of Android OS and the corresponding version of Java supported by the Android Runtime (ART):

Android OS Version	Java Version
Android 11	OpenJDK 11
Android 10	OpenJDK 8
Android 9	OpenJDK 8
Android 8.0 - 8.1	OpenJDK 8
Android 7.0 - 7.1	OpenJDK 7
Android 6.0	OpenJDK 6

- StackOverflow

0 votes 0 answers 91 views

How to determine Java Version per Android OS version

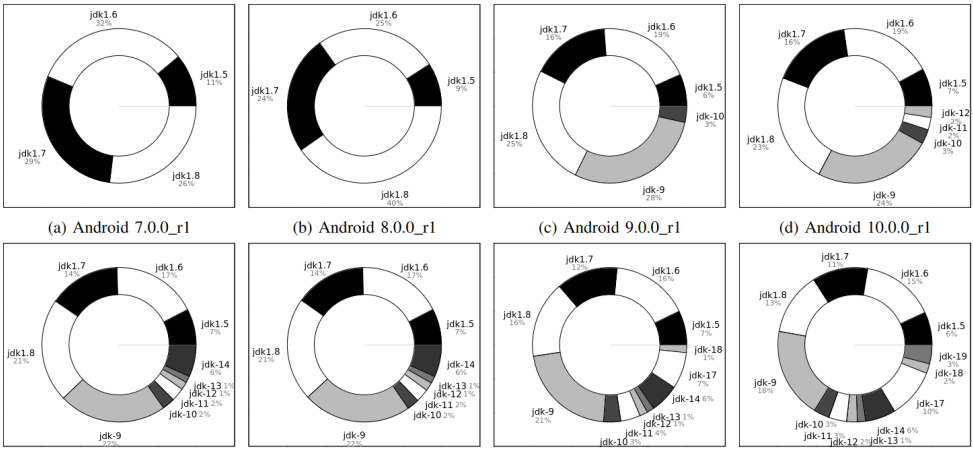
java android code-analysis runtime-environment

- How can I reach the same conclusion?

RQ1: Which OpenJDK versions are used in Android's libcore? How much do they diverge from the OpenJDK upstream? (1/2)

- # Classes: Android 7: 1200 up to Android 13 >2000
- We compute the distance (`tlsh_unittest`) between one Java Class in libcore and all OpenJDKs' versions of the same class.
  - Selection of closest version • Oldest version selected
- We observe first that overall the distance with OpenJDK increases over versions.
  - More and more Android customisation

RQ1: Which OpenJDK versions are used in Android's libcore? ... (2/2)



(a) Android 7.0.0_r1 (b) Android 8.0.0_r1 (c) Android 9.0.0_r1 (d) Android 10.0.0_r1

(e) Android 11.0.0_r1 (f) Android 12.0.0_r1 (g) Android 13.0.0_r1 (h) master branch

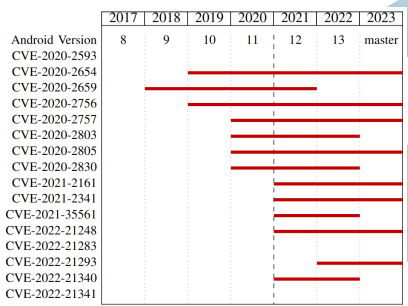
Fig. 4: Representation of the OpenJDK profiles of Android's libcore over versions

RQ2: How have OpenJDK vulnerabilities been managed in libcore? (1/2)

- Methodology:
  - Retrieve all 82 OpenJDK CVEs from NVD feeds
  - Gather the CVE patching commits
  - List files patched in OpenJDK
  - Detect when patched in OpenJDK
  - List files present in Android
  - Detect when patched in libcore
- We consider CVEs if and only if all files are present in libcore
- 78% of OpenJDK CVEs for which files in the patch are never present in libcore (63 CVEs)
- 80.5% of OpenJDK's CVEs are not fully present in libcore (66 CVEs)

RQ2: How have OpenJDK vulnerabilities been managed in libcore? (2/2)

- Over-Exposure[1]



- 13 cases over-exposures found
- 8 CVEs still unpatched in master (24th March 2023[2]).

[1] Patched in OpenJDK and not in Android

[2] Google notified in April 2023

What is Google doing: `Expected_Upstream` (1/2)

- Android is currently working on the OpenJDK update issue
  - In Sept. 2021: Google automates updates through the `Expected_Upstream` file.
  - Slow `rebase` of libcore classes on latest OpenJDK versions

    → First class updated to OpenJDK-17 in February 2022 (GA Sept.21).

    June 2024: 1608 over around 2600 classes.

    → Oct 23: 48 classes updated to JDK-21 —— Oct 24: 1239

    → Still 128 classes based on JDK-7, 378 on JDK-11

What is Google doing: `Expected_Upstream` (2/2)

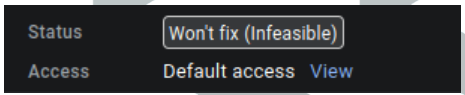
```
15 # This table has 3 columns, i.e.
16 # <destination path in ojluni>,<upstream release version / git-tag>,<source path in the upstream repository>
17
18 ojluni/src/main/java/com/sun/net/ssl/internal/ssl/X509ExtendedTrustManager.java,jdk8u/jdk8u121-b13,jdk/src/share/classes/com/sun/net/ssl/internal/ssl/X509ExtendedTrustManager.java
19 ojluni/src/main/java/com/sun/nio/file/ExtendedCopyOption.java,jdk8u/jdk8u121-b13,jdk/src/share/classes/com/sun/nio/file/ExtendedCopyOption.java
20 ojluni/src/main/java/com/sun/nio/file/ExtendedOpenOption.java,jdk8u/jdk8u121-b13,jdk/src/share/classes/com/sun/nio/file/ExtendedOpenOption.java
21 ojluni/src/main/java/com/sun/nio/file/ExtendedWatchEventModifier.java,jdk8u/jdk8u121-b13,jdk/src/share/classes/com/sun/nio/file/ExtendedWatchEventModifier.java
```

<destination path in ojluni> <upstream release version/git-tag> <source path in the upstream repo>

On a few cases we have a difference with `Expected_Upstream`
BUT we manually confirmed our results as the class in `ojluni` *regresses* to look like an the tlsh version.

RQ3: What is the security impact of OpenJDK CVEs affecting Android?

- Over the 16 vulnerabilities for which the files are ever present in libcore:
  - 3 CVEs depend on the JVM → unexploitable on Android
  - 10/13 CVEs affect mostly Availability, 3 affect Integrity and 1 Confidentiality
  - We informed Google and provided all our code and data[1].

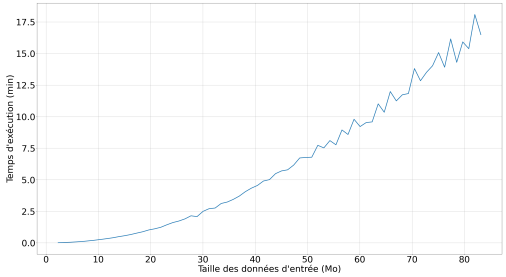


- They can fairly hope that their `Expected_Upstream` project will naturally update vulnerable files early enough.

[1] as of May 2023

CVE-2022-21340

- Exploit for CVE-2022-21340 (CVSS 5.3)
  - We wrote a PoC causing the Denial of Service in OpenJDK-11
  - Improper handling of attributes' length in compressed `.jar`.
  - The application using th same code causes a hang on Android 13




```
371 void read(Manifest.FastInputStream is, byte[] lbuf) throws
↪  IOE {
372   String name = null, value;
373   byte[] lastline = null;
375   int len;
376   while ((len = is.readLine(lbuf)) != -1) {
...       ...
388       if (lbuf[0] == ' ') {
389       // continuation of previous line
...       ...
398         lineContinued = true;
399         byte[] buf = new byte[lastline.length + len - 1];
400         System.arraycopy(lastline, 0, buf, 0,
↪  lastline.length);
401         System.arraycopy(lbuf, 1, buf, lastline.length, len
↪  - 1);
402         if (is.peek() == ' ') {
403          lastline = buf;
404          continue;
405         }
406         value = new String(buf, 0, buf.length,
↪  UTF_8.INSTANCE);
```

Conclusion

- We found Android's libcores to have a fragmented OpenJDK profile and that even for Google, it is not necessarily easy to turn on automatical updates with the upstream.
- We did not see a specific tracking of OpenJDKs CVEs. Only the `Expected_Upstream` process might catch-up with latest version and CVEs patching.
- The exploitation of CVE-2022-21340 on Android 13 proves that OpenJDK vulnerabilities have reached Android releases.
- All data, code and results are available on github[1]

thank you for listening, eager to answer your questions :-)

[1] https://github.com/software-engineering-and-security/AndroidsJCL-SecDev23

Appendix 1: TLSH

- Trendmicro's **L**ocally **S**ensitive **H**ashing
- Sliding window of 5 bytes - repartition in quartile buckets -> digest
- Distance is provided throug Hamming distance derivative between 2 digests.
- Adopted by VirusTotal, Malware Bazaar, Threat Information eXpression (STIX) 2.1

Systematically Changing a File

We started with the first 500 lines of Pride and Prejudice (pg1342.txt from [9]). We created 500 versions of this text, each one more `different` from the original text than the previous.

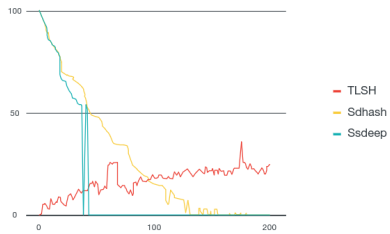
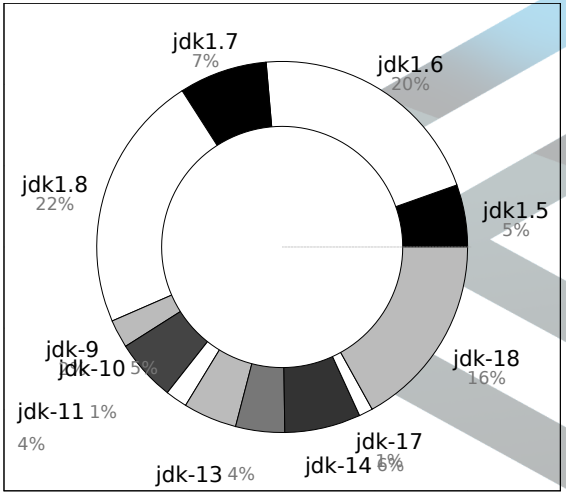
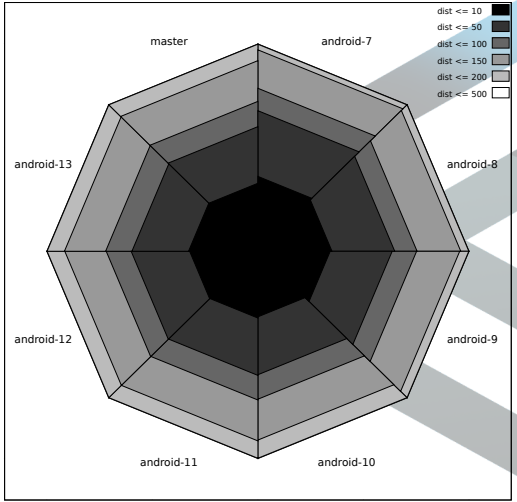


Figure 2. The scores on mutations of the first 500 lines of Pride and Prejudice.

Appendix 2: Selecting the latest version for Android 13



Appendix 3: Distance from Libcore to OpenJDK



Appendix 4: `Expected_Upstream` does not invalidate `tlsh` results

- `tlsh` provides results for *before* the existence of the Expected_Upstream file.
- On Android-13, manual investigation show that the customization of OpenJDK classes in Libcore usually make these classes *closer* to `tlsh` version, than the one pointed by the `Expected_Upstream` file.

Appendix 5: Google policy

Won't Fix (Infeasible)	The changes that are needed to address the issue are not reasonably possible.

Appendix 6: License Background - Harmony, Java and OpenJDK

- In 2011, Apache dropped effort on Harmony as IBM stopped depending on it.
- Oracle sued Google for copyright infrigement over "similar method headers" over 37 packages
- The Supreme Court stated, eventually in 2021, that it was 'fair use' (i.e. you cannot patent the header of a method that describes what the method intents to do). ▶ no issue over 2011-2016.
- The 2016 switch protects both Google Java code and applications developpers to own their code through the Classpath exception (no royalties to Oracle) and escape GPLv2's Copyleft.
- Android escapes the *Java trap* by never executing Java bytecode but Dalvik bytecode in their own VM/Runtime.

Exploit CVE-2022-21340

```java
// CVE-2022-21340
// Fixed in Oracle Java SE 17.0.2, 11.0.14, 8u321, and 7u331.
// The JVM (tested with 11.0.1) hangs on a jar file generated with the following
// 'generate.py' Python script
//
// print("a: " + 230*"b ")
// for i in range(0,300000):
//     print(" " + 230*"b ")
//
// $ mkdir META-INF
// $ python3 generate.py > META-INF/MANIFEST.INF
// $ zip -rv test.jar META-INF
//
// Tested on 11.0.14 and the JVM does not hang anymore.

public class Main {

        public static void main(String[] args) throws Exception {

                // a line in a jar manifest is max 512 bytes
                // all lines with space and single letters

                // target jar file
                String TARGET_JAR = args[0];

                // printHeapInfo();
                System.out.println("start reading jar..."+TARGET_JAR);
                java.util.jar.JarFile jf = new JarFile(TARGET_JAR);
                Map<String, Attributes> e2a = jf.getManifest().getEntries();
                Attributes ma = jf.getManifest().getMainAttributes();
                String v = ma.getValue("a1000");
                System.out.println("value for a0: " + v);
                System.out.println("entries: "+ e2a.keySet().size());
                for (String s: e2a.keySet()) {
                        System.out.println("key: "+ s);
```