
  Bug: upgrading to gradle 8.x , it removes important classes when obfuscating

+15

Hotlists (4)

Mark as Duplicate






Comments (15)DependenciesDuplicates (0)Blocking (0)Resources (5)

WAI


Bug

P3

+ Add Hotlist

 STATUS UPDATE No update yet. 

Edit

 DESCRIPTION lb...@gmail.com created issue [#1](#)

- Steps to reproduce the problem (including sample code if appropriate).

1. Have your app use ZipFile of the apache-compress, on a real file that exists on the file system, while having the MANAGE\_EXTERNAL\_STORAGE permission granted.
2. Run the app, not obfuscated. See that it works fine.
3. Do the same again, but this time obfuscated

- What happened.  
It crashes, with the exception of :  
  

```
java.lang.NoClassDefFoundError: org.apache.commons.compress.archivers.zip.ExtraFieldUtils
    at org.apache.commons.compress.archivers.zip.ZipArchiveEntry.getUnparseableOnly(ZipArchiveEntry.java:896)
    at org.apache.commons.compress.archivers.zip.ZipArchiveEntry.getAllExtraFieldsNoCopy(ZipArchiveEntry.java:563)
    at org.apache.commons.compress.archivers.zip.ZipArchiveEntry.setExtra(ZipArchiveEntry.java:1169)
    at org.apache.commons.compress.archivers.zip.ZipArchiveEntry.setExtraTimeFields(ZipArchiveEntry.java:1229)
    at org.apache.commons.compress.archivers.zip.ZipArchiveEntry.setTime(ZipArchiveEntry.java:1380)
    at org.apache.commons.compress.archivers.zip.ZipArchiveInputStream.getNextZipEntry(ZipArchiveInputStream.java:785)
```

Not only that, but using Proguard doesn't seem to help.  
I tried many combinations for this.


- What you think the correct behavior should be.  
Should work fine.

- Is this a security related issue? Yes/No  
Yes, as it's about obfuscation.


Sadly, I can't provide a sample code, but know that it does occur. And only because of upgrading to gradle 8.x.  
I also tried the newer versions, including the one that comes with canary 5 of Android Studio.  
Still same issue.

✓ Mentioned issues (1)


✓ Links (3)


 **Mentioned issues (1)**

P3 Bug: upgrading to gradle 8.x , it causes "newInstance" not to work sometimes, when obfuscating ["https://issuetracker.google.com/299833808"](https://issuetracker.google.com/299833808)


 **Links (3)**



["https://r8.googlesource.com/r8/+refs/heads/master/compatibility-faq.md..."](https://r8.googlesource.com/r8/+refs/heads/master/compatibility-faq.md...)



" ... of the apache code online and I can't see any indication of why ExtraFieldUtils would be eliminated, so it looks like there is a more complicated judgment going on. If you can share a  [compiler ir](#)



"Thanks for the log. That shows the point of failure and the reflection. Online I found the following  [code for ExtraFieldUtils.java](#) :"



COMMENTS



 lb...@gmail.com <lb...@gmail.com>


 deleted  
0 B 

 deleted  
0 B 

 deleted  
0 B 

 deleted  
0 B 

 deleted  
0 B 

 lb...@gmail.com <lb...@gmail.com> [#2](#)

This was tested on 8.0.2.  
The issue still exists on 8.2.0-alpha07.

**lb...@gmail.com** <lb...@gmail.com> [#3](#)

Still the issue exists on 8.2.0-alpha16.

Why does the gradle plugin mess with java.util.zip.ZipFile class when using "minifyEnabled true" ?

I also tried to add this to the proguard rules:

```
-keep public class java.util.zip.ZipFile.** {  
    public protected *;  
}
```

Didn't help.

**lb...@gmail.com** <lb...@gmail.com> [#4](#)

Issue occurs on v8.3.0-alpha02 , too.

**lb...@gmail.com** <lb...@gmail.com> [#5](#)

Similar case, yet this time at least for one place I've used a Proguard rule to protect againsts this:

<https://issuetracker.google.com/issues/299833808>

It shouldn't need Proguard rule though, not here and not there.

**lb...@gmail.com** <lb...@gmail.com> [#6](#)

I think I've found the reason for this:

Need to tell it not to be so confident about what to obfuscate/remove, by adding this to gradle.properties file (default became true) :  
android.enableR8.fullMode=false

More details:

<https://r8.googlesource.com/r8/+refs/heads/master/compatibility-faq.md?pli=1#r8-full-mode>

Why should it be turned on by default though on stable version of gradle, when it's so problematic?  
Please fix this issue. It shouldn't get rid of important classes/functions.

**ze...@google.com** <ze...@google.com> [#7](#)

*Assigned to ze...@google.com.*

Thanks for the report. Nothing is changing `java.util.zip.ZipFile` as that is part of the devices bootclasspath and cannot be changed or overridden by your program. It looks like the `org.apache.commons.compress.archivers.zip.ZipArchiveEntry.getUnparseableOnly (ZipArchiveEntry.java:896)` is making reflective calls? Does adding `-keep org.apache.com`

**ze...@google.com** <ze...@google.com> [#8](#)

Did a quick scan of the apache code online and I can't see any indication of why `ExtraFieldUtils` would be eliminated, so it looks like there is a more complicated judgment going on. If you  
zerny@google.com

**lb...@gmail.com** <lb...@gmail.com> [#9](#)

@7 Something is wrong with the rule you've created.

The IDE can't reach this path even though it seems correct, and it also complains about ";" .

Look at the video.

 **studio64\_2023-09-26\_10-02-51.mp4**  
2.7 MB Download 

**lb...@gmail.com** <lb...@gmail.com> [#10](#)

@7 Is there any way to disable this destructive behavior for specific and/or all dependencies ?  
Some dependencies aren't related to Android, so it's impractical to handle them...



**lb...@gmail.com** <lb...@gmail.com> [#11](#)

@7 I tried to use this rule instead:

```
-keep class org.apache.commons.compress.archivers.zip.ExtraFieldUtils { *; }
```

Still seems same exception.

See attached.  
Also please restore my files that I've attached. I don't understand why you keep deleting my files here.

 **logs.txt**  
20 KB View Download 

**ze...@google.com** <ze...@google.com> [#12](#)

Yes, sorry, forgot the `class` in the proposed rule.

Thanks for the log. That shows the point of failure and the reflection. Online I found the following [code for ExtraFieldUtils.java](#):

```
class ExtraFieldUtils {
<snip>
static {
    IMPLEMENTATIONS = new ConcurrentHashMap<>();
    register(AsiExtraField.class);
    register(X5455_ExtendedTimestamp.class);
    register(X7875_NewUnix.class);
    register(JarMarker.class);
    register(UnicodePathExtraField.class);
    register(UnicodeCommentExtraField.class);
    register(Zip64ExtendedInformationExtraField.class);
    register(X000A_NTFS.class);
    register(X0014_X509Certificates.class);
    register(X0015_CertificateIdForFile.class);
    register(X0016_CertificateIdForCentralDirectory.class);
    register(X0017_StrongEncryptionHeader.class);
    register(X0019_EncryptionRecipientCertificateList.class);
    register(ResourceAlignmentExtraField.class);
}
<snip>
/**
 * Register a ZipExtraField implementation.
 *
 * <p>The given class must have a no-arg constructor and implement
 * the {@link ZipExtraField ZipExtraField interface}.</p>
 * @param c the class to register
 */
public static void register(final Class<?> c) {
    try {
        final ZipExtraField ze = (ZipExtraField) c.newInstance();
        IMPLEMENTATIONS.put(ze.getHeaderId(), c);
    } catch (final ClassCastException cc) { // NOSONAR
        throw new IllegalArgumentException(c + " doesn't implement ZipExtraField"); //NOSONAR
    } catch (final InstantiationException ie) { // NOSONAR
        throw new IllegalArgumentException(c + " is not a concrete class"); //NOSONAR
    } catch (final IllegalAccessException ie) { // NOSONAR
        throw new IllegalArgumentException(c + "'s no-arg constructor is not public"); //NOSONAR
    }
}
}
```

The register function is reflectively calling `newInstance()` on the class constants and assuming the are of type `ZipExtraField`. I'd add the following rule to capture this:

```
-keep class * extends org.apache.commons.compress.archivers.zip.ZipExtraField { <init>(); }
```

Or, you could make it conditional:

```
-if class * extends org.apache.commons.compress.archivers.zip.ZipExtraField
-keep class <1> { <init>(); }
```

With the latter rule, it should be such that if your project does not actually use the `ExtraFieldUtils` then it won't end up keeping those classes.

**lb...@gmail.com** <lb...@gmail.com> [#13](#)

@12 Guys how could I have found such a thing without your help?  
And what would you have done if it wasn't open sourced? Let alone very obfuscated on its own (or uses JNI)...  
This new R8 mechanism is very destructive and doesn't provide enough details of how to handle such cases.  
Not before creating the APK and now while running it.  
You've seen it by yourself, that you wanted to use what's shown on the logs, but it failed.

Please improve it. Have a way to at least exclude it from reaching dependencies that we have no control over.

Would it be possible to contact the developers of this library as you suggested me somewhere? This dependency isn't even related to Android, and should be safe to use everywhere...

I've added the first rule and it works now:

```
-keep class * extends org.apache.commons.compress.archivers.zip.ZipExtraField { <init>(); }
```

Second rule also works fine:

```
-if class * extends org.apache.commons.compress.archivers.zip.ZipExtraField  
-keep class <1> { <init>(); }
```

You can see it's impossible to find all of these issues until it's too late.

Please at least make the IDE find all of the reflection cases for us and offer suggestions of what to add to Proguard for the various dependencies so we could have a safer migration.

I've compared now the APK before and after the migration:

Before: 8,131,688 bytes

After: 7,847,566 bytes

A difference of less than 300,000 bytes, which is less than 4% of the original size.

For an app that is already very small.

How could I know if that's the last issue it has caused?

Message last modified on Sep 26, 2023 06:19PM

**ze...@google.com** <ze...@google.com> [#14](#)

*Status: Won't Fix (Intended Behavior)*

I'm happy to hear the rule works for you.

You can continue to use compatibility mode which can hide some cases such as these, but if it will work or not is a bit arbitrary (for example the reflective identification is brittle). Some users providing full mode.

Regarding the "last issue", the use of reflection can cause issues in both compatibility mode and full mode. There is no way to avoid that when using a shrinker. We know that it is hard to debug looking at the code.

**lb...@gmail.com** <lb...@gmail.com> [#15](#)

@14

Thing is that now it's turned on by default, as if it's ready for all apps, but it's not, as you can see that it causes issues that didn't occur in the past, and there is no way to fix them, and not even

Suppose you have an app that has plenty of dependencies, and then using it would shrink a lot, how could you know what should be fixed ?

Accessing code of dependencies isn't even always possible, so the solution of "looking at the code" isn't something you can do.

Would you search globally for reflection-related functions, such as "newInstance"?

Why can't the IDE help at all here? There is the new "AGP upgrade assistant" tool, which should have helped here...

Is there a way to avoid using this for specific/all dependencies?