This page summarizes a bug report and and a suggested fix posted on dev@maven.apache.org: https://lists.apache.org/thread/q0bn38qtxkv4orx6o9lhtonjcxkbtw5f

The reproducer project: https://github.com/vlsi/jarsplit

Claim 1

With Maven 4 today, a library author cannot safely split a monolithic artifact into several modules and release it as a backward-compatible minor version (e.g., 1.1.0) without coordinated changes in downstream applications.

Minimal test case:
* app-maven4 application uses only svg and http libraries; it's build with Maven 4
* commons-compress splits into multiple modules and releases 1.1.0
* http:1.1.0 adopts fine-grained commons-compress-tar:1.1.0
* The app app-maven4 bumps http from 1.0.0 to 1.1.0
* **Result**: NoSuchMethodError ... TarCompressor.methodAddedIn11 in the runtime

Note that the user application fails **even in the case commons-compress maintains full backward compatibility**.

This happens **even when** commons-compress:1.1.0 **is backward-compatible** for consumers; the failure is caused by resolution not aligning related commons-compress-* modules

Known workarounds (and why they're not sufficient):
a) Exclude commons-compress transitive deps from svg/http and re-add the desired ones explicitly.
b) Add explicit dependencyManagement for commons-compress to the app's root POM.

Both require **application-side** curation and ongoing maintenance. If svg/http later stop using commons-compress, the app must be updated again. In effect, the **producer cannot split** without the **consumer editing their POM**, which is exactly what we'd like to avoid for a backward-compatible minor.

However, both workarounds require changes to the application pom, and both workarounds require app developer to figure out the compress version, and maintain the workaround. For instance, if both svg and http stop using commons-compress, the app developer would need to remove commons-compress from their pom as well.
The Maven issue restricts commons-compress vendor from splitting the artifact without coordinated changes to all the applications.

Note: Gradle builds of the same scenario continue to run after the http version bump, i.e., they are not affected by this artifact-split scenario.

Claim 2

Maven users often suffer when multi-module libraries get misaligned versions.
Even though different modules of a single library might have a completely independent lifecycle, often there's a need to use consistent versions.

For instance, org.apache.jmeter releases 10+ jars for every release: https://central.sonatype.com/search?q=jmeter&namespace=org.apache.jmeter. If one dependency uses ApacheJMeter_core:5.0, and another one uses ApacheJMeter_http:5.6, then the app might crash unless Maven bumps _core to 5.6. At the end of the day _http might depend on _core 5.6 features.

The same goes with commons-compress above: if one of the libraries depend on commons-tar:1.1 and another library depends on commons-gz:1.2, then it would be better to align all commons-* to the same 1.2 version (assuming they release together).

Historically, the Maven has a workaround of importing library's bom to the end-user pom file: https://docs.junit.org/current/user-guide/#running-tests-build-maven-bom

Such an import is great if the end user uses the library directly (like with junit), however, it is bad for transitive dependencies. The app author might not even know there's commons-compress somewhere among transitives.

https://jlbp.dev/JLBP-6 explicitly refrains from splitting artifacts; however, it is there to carve around Maven's issues. Gradle has no such limitations: https://github.com/GoogleCloudPlatform/cloud-opensource-java/issues/2456

Suggested fix

Can we consider changing Maven 4 resolution so that transitive <dependencyManagement> is honored at classpath scope, enabling version alignment across a module family (e.g., commons-compress-*) when a producer ships a split minor?
This would let library authors split/merge modules without forcing each application to hand-craft excludes or duplicate management blocks.

If you see a better mechanism than transitive <dependencyManagement> to enable seamless artifact splits, I'm all ears—please suggest it.

Similar demands from others

Piotr P. Karwasz (Logging PMC):
https://lists.apache.org/thread/z5vqbh75nsl03sy26xtrxh3fwv1h2kn7

However, automatic alignment is very useful in practice. For example, it can keep a project's Log4j modules consistent even without an explicit `log4j-bom` import. This will matter even more going forward: with Log4j Core 3 the line splits: `log4j-core` and its extensions move to 3.x, while `log4j-api` and the bridges remain on 2.x, so naïve version alignment will not work.

Chris Povirk (Guava):
https://github.com/google/guava/issues/8079#issuecomment-3492464905
The situation with Maven has been a showstopper, so if it were improved, then this would become an interesting discussion for us to have, time permitting.

Jackson (since 2.12.0), **JUnit** use version alignment for 5+ years already:
https://blog.gradle.org/alignment-with-gradle-module-metadata

Commons Compress: there's a desire to split the current commons-compress.jar into modules; however, it can only be done with package rename and artifact rename.
https://lists.apache.org/thread/bwhsonqdq1f57hrfz3l6wy1gxrtssbsc

FAQ

- **Q**: Why did nobody file such an issue earlier?

  A: Does it really matter? People might switch to Gradle, or they might lose hope that Maven could be improved. The bug is present, so let's focus on how it could be fixed.
- **Q**: Dependency management is already complicated. Why make it even harder?

  A: The proposal makes dependency management **easier** for consumers at a slight cost for the producers.
- **Q**: Which Maven core principles does the proposal break?

  A: None
- **Q**: Why don't you apply the regular Maven pattern of renaming packages and renaming artifacts?

  A: As an artifact publisher, I would like to have a backward-compatible release so my consumers won't have to modify their code when upgrading
- **Q**: The exec maven plugin could produce a misleading classpath. Why do you use it?

  A: The exec plugin is very popular, and it uses the direct MavenProject#getRuntimeArtifacts API, so there's not much room for failure. Both exec-maven-plugin and maven-dependency-plugin resolve the same classpath, so the bug is likely with Maven rather than the exec plugin.
- **Q**: Why does the application fail at runtime?

  A: The application uses two libraries:
  svg:1.0.0, which uses commons-compress:1.0.0
  and
  http:1.1.0, which depends on compress-tar:1.1.0, which depends on

compress-core:1.1.0.

The classpath in Java is order sensitive, so Maven builds the following classpath:
svg-1.0.0.jar
commons-compress-1.0.0.jar // still includes TarCompressor v1.0.0
http-1.1.0.jar // it uses TarCompressor v1.1.0
commons-tar-1.1.0.jar // it brings TarCompressor v1.1.0
commons-base-1.1.0.jar

This classpath causes http library to use old TarCompressor 1.0.0, thus the app fails.

The problem is that Maven currently can't recognize that it should bump commons-compress in the presence of commons-tar/commons-base.

- **Q**: How does the proposal fix the issue?

  A: The proposal would enable commons-tar 1.1.0 author to publish metatada so maven could understand tar 1.1.0 needs to bump commons-compress to 1.1.0. Note: it differs from a regular dependency since tar 1.1.0 alone must not bring regular commons-compress dependency.
- **Q**: Has this been tested already?

  A: Gradle has implemented platforms, and multiple projects adopt the technology since 2019: https://blog.gradle.org/alignment-with-gradle-module-metadata
- **Q**: Why Google Java Best Practices refrain authors from splitting the artifacts like in https://jlbp.dev/JLBP-6?

  A: They carve around Maven tool limitations. Gradle allows library authors to split their jars and keep backward compatibility at the same time. There's an issue to improve JLBP-6 wording: https://github.com/GoogleCloudPlatform/cloud-opensource-java/issues/2456

Actions on dev@maven

- **Guillaume Nodet**: opened issue https://github.com/apache/maven/issues/11391: Configurable Dependency Version Resolution Strategy

  In fact, Guillaume replied to a different thread https://lists.apache.org/thread/dnsgqc2hdw4dw4dvh67ggdybkvcwpjyj, so the created issue had nothing to do with the current bug/fix.

Suggestions on dev@maven

Unfortunately, none of the suggestions resolves the issue.

- **Sergey Chernov**: Don't split jars. Many libraries like apache httpclient suffer from it as they are quite senstive to matching httpcore, httpclient and httpasyncclient

  Response: It is exactly the issue I mention: automatic alignment is needed. If the fix is implemented, Apache httpclient authors would have a way to configure poms in such a way so httpcore, httpclient align to the same (or whatever workable) versions.
- **Sergey Chernov**: use shade plugin

  Response: Shading does not solve the version alignment issue. Shading is hard to get right: overliapping license files, merging manifests, merging custom resources, signed jars make shaing non-trivial. At the same time, the example shows http and svg third-party libraries, and forcing their authors to shade is not possible.
- **Sergey Chernov**: Use maven-enforcer-plugin in your projects (not libraries). Use plugins (don't remember exact name) that verify there are no classpath duplications on your project (not libraries)

  Response: Enforcer can detect issues; however, it can't align versions. Enforcer can't help library authors to safely split their libraries.
- **Sergey Chernov**: consider maven "<distributionManagement><relocation>" pom

  Response: relocation pom it can't align versions. Enforcer can't help library authors to safely split their libraries

- **Tamás Cservenák**: You need to "tweek" the project to add commons-compress-core:1.1.0 depends on commons-compress:1.1.0 (*note: version numbers are updated to correspond to the current bug/fix text above*)
  https://lists.apache.org/thread/93zs653trcnc57bc4mrg1qy7rxr73vkw

  Response: Such a change defeats the purpose of the modularization. Users won't be able to depend on "core alone":
  https://lists.apache.org/thread/1go00cd938mhwvlk1qcmfhvp13noq3o6
  Similar suggestions do not solve the issue, but they create a circular dependency:
  https://lists.apache.org/thread/dbk0vldv08x92pcxj4t5f2g3rx36zkm1

Comments on dev@maven

There are many comments by Romain Manni-Bucau, unfortunately, none of them have a technical justification

- **Romain Manni-Bucau**: the proposal requires/assumes semantic versions (semver). https://lists.apache.org/list?dev@maven.apache.org:lte=1M:semver%20Honor%20dependencyManagement

  Response: The proposal does not require semver, and it does not assume semver. Neither “claim” nor “suggested fix” mention semver. The fix would work with any Maven-compatible version.

- **Romain Manni-Bucau**: The proposal will break users, it will mess up projects. https://lists.apache.org/list?dev@maven.apache.org:lte=1M:mess%20Honor%20dependencyManagement

  Response: So far, there has been absolutely no concrete example that would show how the proposal would break anything.

- **Romain Manni-Bucau**: today it works for every maven users - they do deliver, even you in your other projects
  https://lists.apache.org/thread/qks4krjkv5mrdcfkmt5zrcwg278x7l5o

  Response: reproducer shows that Maven fails at runtime. See “Similar demands from others” which highlight the feature is needed by many projects.

- **Romain Manni-Bucau**: You do understand what you mean when you say "today it is broken", if you reverse it then you just break it the exact same way where it works today
  https://lists.apache.org/thread/7glb632ob043z8gkzxosw29snmh9zgbb

  Response: The logic is flawed: if one goes by that logic, it would be forbidden to fix any bug in Maven. Romain did not clarify what they mean by “reverse”.

- **Romain Manni-Bucau**: exec resolution can differ from dependency plugin (and other plugins: https://lists.apache.org/thread/khs6l9zl2wfd99g8t4p8mgd7o3wxqgnd
  but exec print 1.0.0 in your main so likely a bug in exec which doesn't use the strict resolution of resolver:
  https://lists.apache.org/thread/19bg02q09njgbzymkq8w6463b5c64jrx

  Response: This is a false claim.
  Running `./mvnw dependency:tree`; and `./mvnw -X verify` at b7dae3f1609b978e1a33ffa63d717bc66e45daa5 confirm exec plugin and dependency plugin resolve the same classpath:

  ```
  [DEBUG] Collected project artifacts
  [org.example:svg:jar:1.0.0:compile,
  org.example:commons-compress:jar:1.0.0:runtime,
  org.example:http:jar:1.1.0:compile,
  org.example:commons-compress-tar:jar:1.1.0:runtime,
  org.example:commons-compress-core:jar:1.1.0:runtime]
  ```

```
[INFO] org.example:app:jar:1.0.0
[INFO] +- org.example:svg:jar:1.0.0:compile
[INFO] |  \- org.example:commons-compress:jar:1.0.0:runtime
[INFO] \- org.example:http:jar:1.1.0:compile
[INFO]    \- org.example:commons-compress-tar:jar:1.1.0:runtime
[INFO]       \- org.example:commons-compress-core:jar:1.1.0:runtime
```

As you can see, both result in svg:1.0.0; commons-compress:1.0.0; http:1.1.0; commons-compress-tar:1.1.0; commons-compress-core:1.1.0

Interestingly, even after this obvious proof, Romain still does not agree the classpaths are the same between exec and dependency plugins (see “Obviously not since I know it is wrong and technically it can't be”): https://lists.apache.org/thread/27r4fwxkdgzgx47vc6dt58d27o221xcg