Fortify Standalone Report Generator

Developer Workbook

akka-serialization-jackson

Table of Contents

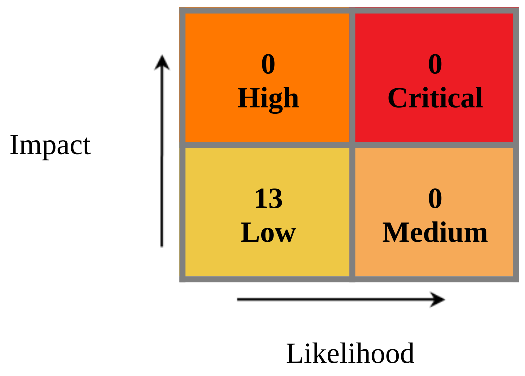
Executive Summary

This workbook is intended to provide all necessary details and information for a developer to understand and remediate the different issues discovered during the akka-serialization-jackson project audit. The information contained in this workbook is targeted at project managers and developers.

This section provides an overview of the issues uncovered during analysis.

Project Name:	akka-serialization-jackson
Project Version:	
SCA:	Results Present
WebInspect:	Results Not Present
WebInspect Agent:	Results Not Present
Other:	Results Not Present

Issues by Priority



Top Ten Critical Categories

This project does not contain any critical issues

Project Description

This section provides an overview of the Fortify scan engines used for this project, as well as the project meta-information.

SCA

Date of Last Analysis:	Jun 16, 2022, 11:45 AM	**Engine Version:**	21.1.1.0009
Host Name:	Jacks-Work-MBP.local	**Certification:**	VALID
Number of Files:	32	**Lines of Code:**	1,454

Rulepack Name	Rulepack Version
Fortify Secure Coding Rules, Extended, Java	2022.1.0.0007
Fortify Secure Coding Rules, Core, Scala	2022.1.0.0007
Fortify Secure Coding Rules, Extended, JSP	2022.1.0.0007
Fortify Secure Coding Rules, Core, Android	2022.1.0.0007
Fortify Secure Coding Rules, Extended, Content	2022.1.0.0007
Fortify Secure Coding Rules, Extended, Configuration	2022.1.0.0007
Fortify Secure Coding Rules, Core, Annotations	2022.1.0.0007
Fortify Secure Coding Rules, Community, Cloud	2022.1.0.0007
Fortify Secure Coding Rules, Core, Universal	2022.1.0.0007
Fortify Secure Coding Rules, Core, Java	2022.1.0.0007
Fortify Secure Coding Rules, Community, Universal	2022.1.0.0007

Issue Breakdown by Fortify Categories

The following table depicts a summary of all issues grouped vertically by Fortify Category. For each category, the total number of issues is shown by Fortify Priority Order, including information about the number of audited issues.

Category	Fortify Priority (audited/total)				Total Issues
	Critical	High	Medium	Low	
Code Correctness: Byte Array to String Conversion	0	0	0	0 / 1	0 / 1
Code Correctness: Constructor Invokes Overridable Function	0	0	0	0 / 1	0 / 1
Code Correctness: Erroneous String Compare	0	0	0	0 / 10	0 / 10
Code Correctness: Non-Static Inner Class Implements Serializable	0	0	0	0 / 1	0 / 1

Results Outline

Code Correctness: Byte Array to String Conversion (1 issue)

Abstract

Converting a byte array into a `String` may lead to data loss.

Explanation

When data from a byte array is converted into a `String`, it is unspecified what will happen to any data that is outside of the applicable character set. This can lead to data being lost, or a decrease in the level of security when binary data is needed to ensure proper security measures are followed. **Example 1:** The following code converts data into a String in order to create a hash.

```
  ...
  FileInputStream fis = new FileInputStream(myFile);
  byte[] byteArr = byte[BUFSIZE];
  ...
  int count = fis.read(byteArr);
  ...
  String fileString = new String(byteArr);
  String fileSHA256Hex = DigestUtils.sha256Hex(fileString);
  // use fileSHA256Hex to validate file
  ...
```

Assuming the size of the file is less than `BUFSIZE`, this works fine as long as the information in `myFile` is encoded the same as the default character set, however if it's using a different encoding, or is a binary file, it will lose information. This in turn will cause the resulting SHA hash to be less reliable, and could mean it's far easier to cause collisions, especially if any data outside of the default character set is represented by the same value, such as a question mark.

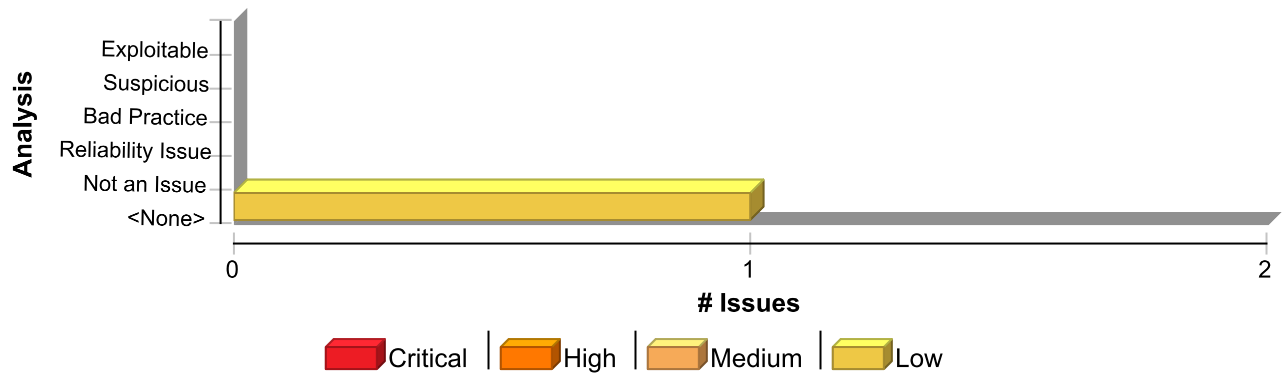
Recommendation

Generally speaking, a byte array potentially containing noncharacter data should never be converted into a `String` object as it may break functionality, but in some cases this can cause much larger security concerns. In a lot of cases there is no need to actually convert a byte array into a String, but if there is a specific reason to be able to create a `String` object from binary data, it must first be encoded in a way such that it will fit into the default character set. **Example 2:** The following uses a different variant of the API in `Example 1` to prevent any validation problems.

```
  ...
  FileInputStream fis = new FileInputStream(myFile);
  byte[] byteArr = byte[BUFSIZE];
  ...
  int count = fis.read(byteArr);
  ...
  byte[] fileSHA256 = DigestUtils.sha256(byteArr);
  // use fileSHA256 to validate file, comparing hash byte-by-byte.
  ...
```

In this case, it is straightforward to rectify, since this API has overloaded variants including one that accepts a byte array, and this could be simplified even further by using another overloaded variant of `DigestUtils.sha256()` that accepts a `FileInputStream` object as its argument. Other scenarios may need careful consideration as to whether it's possible that the byte array could contain data outside of the character set, and further refactoring may be required.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Code Correctness: Byte Array to String Conversion	1	0	0	1
Total	**1**	**0**	**0**	**1**

Code Correctness: Byte Array to String Conversion — Low

Package: akka.serialization.jackson

test/scala/akka/serialization/jackson/JacksonSerializerSpec.scala, line 150 (Code Correctness: Byte Array to String Conversion) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Semantic)

Sink Details

Sink: String()
Enclosing Method: fromBinary()
File: test/scala/akka/serialization/jackson/JacksonSerializerSpec.scala:150
Taint Flags:

```
147  override def identifier: Int = 123451
148  override def manifest(o: AnyRef): String = "M"
149  override def toBinary(o: AnyRef): Array[Byte] = o.asInstanceOf[HasAkkaSerializer].description.getBytes()
150  override def fromBinary(bytes: Array[Byte], manifest: String): AnyRef = HasAkkaSerializer(new String(bytes))
151  }
152
153  final case class WithAkkaSerializer(
```

Code Correctness: Constructor Invokes Overridable Function (1 issue)

Abstract

A constructor of the class calls a function that can be overridden.

Explanation

When a constructor calls an overridable function, it may allow an attacker to access the `this` reference prior to the object being fully initialized, which can in turn lead to a vulnerability. **Example 1:** The following calls a method that can be overridden.

```
  ...
  class User {
    private String username;
    private boolean valid;
    public User(String username, String password){
      this.username = username;
      this.valid = validateUser(username, password);
    }
    public boolean validateUser(String username, String password){
      //validate user is real and can authenticate
      ...
    }
    public final boolean isValid(){
      return valid;
    }
  }
```

Since the function `validateUser` and the class are not `final`, it means that they can be overridden, and then initializing a variable to the subclass that overrides this function would allow bypassing of the `validateUser` functionality. For example:

```
  ...
  class Attacker extends User{
    public Attacker(String username, String password){
      super(username, password);
    }
    public boolean validateUser(String username, String password){
      return true;
    }
  }
  ...
  class MainClass{
    public static void main(String[] args){
      User hacker = new Attacker("Evil", "Hacker");
      if (hacker.isValid()){
        System.out.println("Attack successful!");
      }else{
        System.out.println("Attack failed");
      }
    }
  }
```

The code in `Example 1` prints "Attack successful!", since the `Attacker` class overrides the `validateUser()` function that is called from the constructor of the superclass `User`, and Java will first look in the subclass for functions called from the constructor.

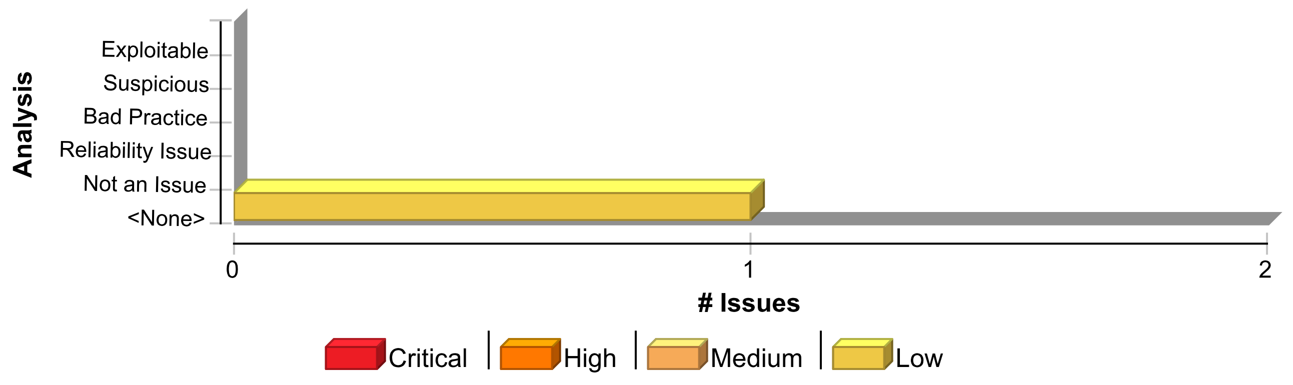
Recommendation

Constructors should not call functions that can be overridden, either by specifying them as `final`, or specifying the class as `final`. Alternatively if this code is only ever needed in the constructor, the `private` access specifier can be used, or the logic could be placed directly into the constructor of the superclass. **Example 2:** The following makes the class `final` to prevent the function from being overridden elsewhere.

```
  ...
  final class User {
    private String username;
    private boolean valid;
    public User(String username, String password){
      this.username = username;
      this.valid = validateUser(username, password);
    }
    private boolean validateUser(String username, String password){
      //validate user is real and can authenticate
      ...
    }
    public final boolean isValid(){
      return valid;
    }
  }
```

This example specifies the class as `final`, so that it cannot be subclassed, and changes the `validateUser()` function to `private`, since it is not needed elsewhere in this application. This is programming defensively, since at a later date it may be decided that the `User` class needs to be subclassed, which would result in this vulnerability reappearing if the `validateUser()` function was not set to `private`.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Code Correctness: Constructor Invokes Overridable Function	1	0	0	1
Total	**1**	**0**	**0**	**1**

Code Correctness: Constructor Invokes Overridable Function	**Low**
Package: akka.serialization.jackson	
test/scala/akka/serialization/jackson/JacksonSerializerSpec.scala, line 725 (Code Correctness: Constructor Invokes Overridable Function)	**Low**

Issue Details

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.serialization.jackson

test/scala/akka/serialization/jackson/JacksonSerializerSpec.scala, line 725 (Code Correctness: Constructor Invokes Overridable Function) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: baseConfig
Enclosing Method: JacksonSerializerSpec()
File: test/scala/akka/serialization/jackson/JacksonSerializerSpec.scala:725
Taint Flags:

```
722 }
723
724 abstract class JacksonSerializerSpec(serializerName: String)
725  extends TestKit(
726  ActorSystem(
727  "JacksonJsonSerializerSpec",
728  ConfigFactory.parseString(JacksonSerializerSpec.baseConfig(serializerName))))
```

Code Correctness: Erroneous String Compare (10 issues)

Abstract

Strings should be compared with the `equals()` method, not `==` or `!=`.

Explanation

This program uses `==` or `!=` to compare two strings for equality, which compares two objects for equality, not their values. Chances are good that the two references will never be equal. **Example 1:** The following branch will never be taken.

```
  if (args[0] == STRING_CONSTANT) {
      logger.info("miracle");
  }
```

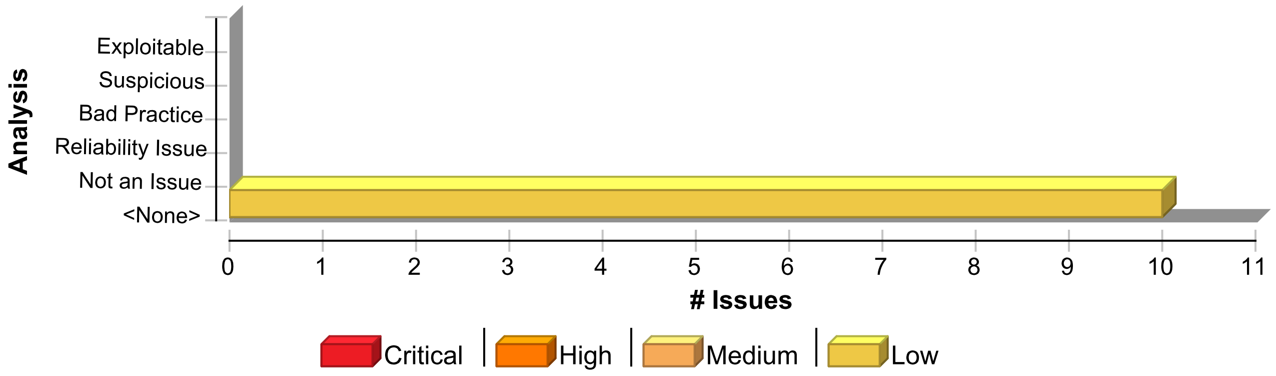
The `==` and `!=` operators will only behave as expected when they are used to compare strings contained in objects that are equal. The most common way for this to occur is for the strings to be interned, whereby the strings are added to a pool of objects maintained by the `String` class. Once a string is interned, all uses of that string will use the same object and equality operators will behave as expected. All string literals and string-valued constants are interned automatically. Other strings can be interned manually be calling `String.intern()`, which will return a canonical instance of the current string, creating one if necessary.

Recommendation

Use `equals()` to compare strings. **Example 2:** The code in `Example 1` could be rewritten in the following way:

```
  if (STRING_CONSTANT.equals(args[0])) {
      logger.info("could happen");
  }
```

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Code Correctness: Erroneous String Compare	10	0	0	10
Total	**10**	**0**	**0**	**10**

Code Correctness: Erroneous String Compare — Low

Package: akka.serialization.jackson

main/scala/akka/serialization/jackson/JacksonSerializer.scala, line 185 (Code Correctness: Erroneous String Compare) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: JacksonSerializer()
File: main/scala/akka/serialization/jackson/JacksonSerializer.scala:185
Taint Flags:

```
182 private val isDebugEnabled = conf.getBoolean("verbose-debug-logging") && log.isDebugEnabled
183 private final val BufferSize = 1024 * 4
184 private val compressionAlgorithm: Compression.Algoritm = {
185 toRootLowerCase(conf.getString("compression.algorithm")) match {
186 case "off" => Compression.Off
187 case "gzip" =>
188 val compressLargerThan = conf.getBytes("compression.compress-larger-than")
```

main/scala/akka/serialization/jackson/JacksonObjectMapperProvider.scala, line 272 (Code Correctness: Erroneous String Compare) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: isModuleEnabled()
File: main/scala/akka/serialization/jackson/JacksonObjectMapperProvider.scala:272
Taint Flags:

```
269 }
270
271 private def isModuleEnabled(fqcn: String, dynamicAccess: DynamicAccess): Boolean =
272 fqcn match {
273 case "akka.serialization.jackson.AkkaTypedJacksonModule" =>
274 // akka-actor-typed dependency is "provided" and may not be included
275 dynamicAccess.classIsOnClasspath("akka.actor.typed.ActorRef")
```

main/scala/akka/serialization/jackson/JacksonObjectMapperProvider.scala, line 272 (Code Correctness: Erroneous String Compare) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Code Correctness: Erroneous String Compare — Low

Package: akka.serialization.jackson

main/scala/akka/serialization/jackson/JacksonObjectMapperProvider.scala, line 272 (Code Correctness: Erroneous String Compare) — Low

Sink Details

Sink: Operation
Enclosing Method: isModuleEnabled()
File: main/scala/akka/serialization/jackson/JacksonObjectMapperProvider.scala:272
Taint Flags:

```
269 }
270
271 private def isModuleEnabled(fqcn: String, dynamicAccess: DynamicAccess): Boolean =
272 fqcn match {
273 case "akka.serialization.jackson.AkkaTypedJacksonModule" =>
274 // akka-actor-typed dependency is "provided" and may not be included
275 dynamicAccess.classIsOnClasspath("akka.actor.typed.ActorRef")
```

main/scala/akka/serialization/jackson/JacksonSerializer.scala, line 214 (Code Correctness: Erroneous String Compare) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: JacksonSerializer()
File: main/scala/akka/serialization/jackson/JacksonSerializer.scala:214
Taint Flags:

```
211 }
212 private val typeInManifest: Boolean = conf.getBoolean("type-in-manifest")
213 // Calculated eagerly so as to fail fast
214 private val configuredDeserializationType: Option[Class[_ <: AnyRef]] = conf.getString("deserialization-type") match {
215 case "" => None
216 case className =>
217 system.dynamicAccess.getClassFor[AnyRef](className) match {
```

main/scala/akka/serialization/jackson/JacksonSerializer.scala, line 185 (Code Correctness: Erroneous String Compare) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: JacksonSerializer()

Code Correctness: Erroneous String Compare — Low

Package: akka.serialization.jackson

main/scala/akka/serialization/jackson/JacksonSerializer.scala, line 185 (Code Correctness: Erroneous String Compare) — Low

File: main/scala/akka/serialization/jackson/JacksonSerializer.scala:185
Taint Flags:

```
182 private val isDebugEnabled = conf.getBoolean("verbose-debug-logging") && log.isDebugEnabled
183 private final val BufferSize = 1024 * 4
184 private val compressionAlgorithm: Compression.Algoritm = {
185 toRootLowerCase(conf.getString("compression.algorithm")) match {
186 case "off" => Compression.Off
187 case "gzip" =>
188 val compressLargerThan = conf.getBytes("compression.compress-larger-than")
```

main/scala/akka/serialization/jackson/JacksonSerializer.scala, line 185 (Code Correctness: Erroneous String Compare) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: JacksonSerializer()
File: main/scala/akka/serialization/jackson/JacksonSerializer.scala:185
Taint Flags:

```
182 private val isDebugEnabled = conf.getBoolean("verbose-debug-logging") && log.isDebugEnabled
183 private final val BufferSize = 1024 * 4
184 private val compressionAlgorithm: Compression.Algoritm = {
185 toRootLowerCase(conf.getString("compression.algorithm")) match {
186 case "off" => Compression.Off
187 case "gzip" =>
188 val compressLargerThan = conf.getBytes("compression.compress-larger-than")
```

Package: doc.akka.serialization.jackson

test/scala/doc/akka/serialization/jackson/CustomAdtSerializer.scala, line 48 (Code Correctness: Erroneous String Compare) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: deserialize()
File: test/scala/doc/akka/serialization/jackson/CustomAdtSerializer.scala:48
Taint Flags:

Code Correctness: Erroneous String Compare — Low

Package: doc.akka.serialization.jackson

test/scala/doc/akka/serialization/jackson/CustomAdtSerializer.scala, line 48 (Code Correctness: Erroneous String Compare) — Low

```
45 import Direction._
46
47 override def deserialize(p: JsonParser, ctxt: DeserializationContext): Direction = {
48 p.getText match {
49 case "N" => North
50 case "E" => East
51 case "S" => South
```

test/scala/doc/akka/serialization/jackson/CustomAdtSerializer.scala, line 48 (Code Correctness: Erroneous String Compare) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: deserialize()
File: test/scala/doc/akka/serialization/jackson/CustomAdtSerializer.scala:48
Taint Flags:

```
45 import Direction._
46
47 override def deserialize(p: JsonParser, ctxt: DeserializationContext): Direction = {
48 p.getText match {
49 case "N" => North
50 case "E" => East
51 case "S" => South
```

test/scala/doc/akka/serialization/jackson/CustomAdtSerializer.scala, line 48 (Code Correctness: Erroneous String Compare) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: deserialize()
File: test/scala/doc/akka/serialization/jackson/CustomAdtSerializer.scala:48
Taint Flags:

```
45 import Direction._
46
47 override def deserialize(p: JsonParser, ctxt: DeserializationContext): Direction = {
```

Code Correctness: Erroneous String Compare — Low

Package: doc.akka.serialization.jackson

test/scala/doc/akka/serialization/jackson/CustomAdtSerializer.scala, line 48 (Code Correctness: Erroneous String Compare) — **Low**

```
48 p.getText match {
49 case "N" => North
50 case "E" => East
51 case "S" => South
```

test/scala/doc/akka/serialization/jackson/CustomAdtSerializer.scala, line 48 (Code Correctness: Erroneous String Compare) — **Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: deserialize()
File: test/scala/doc/akka/serialization/jackson/CustomAdtSerializer.scala:48
Taint Flags:

```
45 import Direction._
46
47 override def deserialize(p: JsonParser, ctxt: DeserializationContext): Direction = {
48 p.getText match {
49 case "N" => North
50 case "E" => East
51 case "S" => South
```

Code Correctness: Non-Static Inner Class Implements Serializable (1 issue)

Abstract

Inner classes implementing `java.io.Serializable` may cause problems and leak information from the outer class.

Explanation

Serialization of inner classes lead to serialization of the outer class, therefore possibly leaking information or leading to a runtime error if the outer class is not serializable. As well as this, serializing inner classes may cause platform dependencies since the Java compiler creates synthetic fields in order to implement inner classes, but these are implementation dependent, and may vary from compiler to compiler. **Example 1:** The following code allows serialization of an inner class.

```
...
class User implements Serializable {
  private int accessLevel;
  class Registrator implements Serializable {
    ...
  }
}
```

In `Example 1`, when the inner class `Registrator` is serialized, it will also serialize the field `accessLevel` from the outer class `User`.

Recommendation

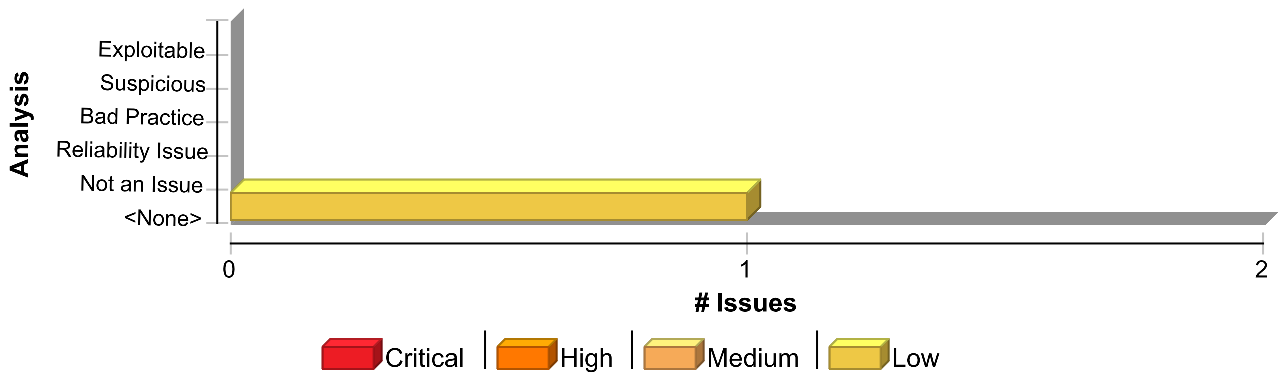
When using inner classes, they should not be serialized, or they should be changed to static-nested classes, since these do not have the drawbacks that non-static inner classes have when serialized. When a nested class is static it inherently has no association with instance variables (including those of the outer class), and would not cause serialization of the outer class. **Example 2:** The following code changes the example in `Example 1`, by stopping the inner class from implementing `java.io.Serializable`.

```
...
class User implements Serializable {
  private int accessLevel;
  class Registrator {
    ...
  }
}
```

Example 2: The following code changes the example in `Example 1`, by making the inner class into a static-nested class.

```
...
class User implements Serializable {
  private int accessLevel;
  static class Registrator implements Serializable {
    ...
  }
}
```

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Code Correctness: Non-Static Inner Class Implements Serializable	1	0	0	1
Total	**1**	**0**	**0**	**1**

Code Correctness: Non-Static Inner Class Implements Serializable — Low

Package: akka.serialization.jackson

main/scala/akka/serialization/jackson/JacksonSerializer.scala, line 86 (Code Correctness: Non-Static Inner Class Implements Serializable) — **Low**

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: JacksonSerializer$LZ4Meta
File: main/scala/akka/serialization/jackson/JacksonSerializer.scala:86
Taint Flags:

```
83 (bytes(1) == (GZIPInputStream.GZIP_MAGIC >> 8).toByte)
84 }
85
86 final case class LZ4Meta(offset: Int, length: Int) {
87 import LZ4Meta._
88
89 def putInto(buffer: ByteBuffer): Unit = {
```