Fortify Standalone Report Generator

Developer Workbook

akka-persistence-typed

Table of Contents

Executive Summary
Project Description
Issue Breakdown by Fortify Categories
Results Outline

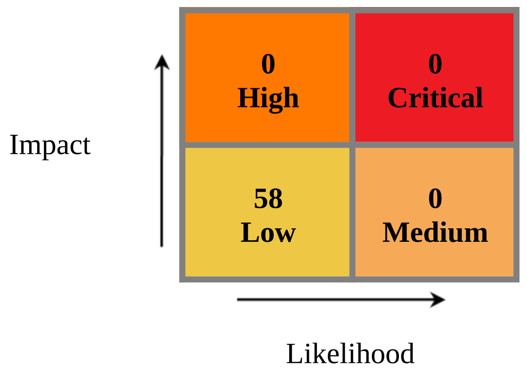
Executive Summary

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-persistence-typed 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-persistence-typed
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, 11:38 AM	**Engine Version:**	21.1.1.0009
Host Name:	Jacks-Work-MBP.local	**Certification:**	VALID
Number of Files:	70	**Lines of Code:**	3,735

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: Constructor Invokes Overridable Function	0	0	0	0 / 22	0 / 22
Code Correctness: Erroneous String Compare	0	0	0	0 / 4	0 / 4
Code Correctness: Non-Static Inner Class Implements Serializable	0	0	0	0 / 23	0 / 23
Dead Code: Expression is Always false	0	0	0	0 / 1	0 / 1
Dead Code: Expression is Always true	0	0	0	0 / 1	0 / 1
Redundant Null Check	0	0	0	0 / 3	0 / 3
System Information Leak: Internal	0	0	0	0 / 4	0 / 4

Results Outline

Code Correctness: Constructor Invokes Overridable Function (22 issues)

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");
    }
  }
}
```

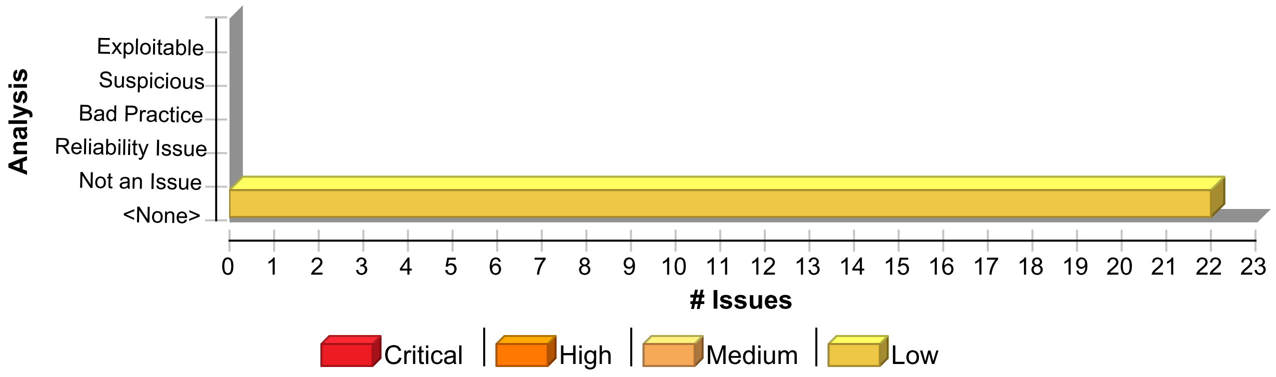
The code in `Example 1` prints "Attack successful!", since the `Attacker` class overrides the `validateUser()` function that is called from the constructor of the superclass `User`, and Java will first look in the subclass for functions called from the constructor.

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	22	0	0	22
Total	**22**	**0**	**0**	**22**

Code Correctness: Constructor Invokes Overridable Function	Low
Package: akka.persistence.typed	
ReplicationId.scala, line 35 (Code Correctness: Constructor Invokes Overridable Function)	**Low**
Issue Details	

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.persistence.typed

ReplicationId.scala, line 35 (Code Correctness: Constructor Invokes Overridable Function) — Low

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: Separator
Enclosing Method: ReplicationId()
File: ReplicationId.scala:35
Taint Flags:

```
32  */
33 final class ReplicationId(val typeName: String, val entityId: String, val replicaId: ReplicaId) {
34  import ReplicationId._
35  if (typeName.contains(Separator))
36  throw new IllegalArgumentException(
37  s"entityTypeHint [$typeName] contains [$Separator] which is a reserved character")
38
```

ReplicationId.scala, line 36 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: Separator
Enclosing Method: ReplicationId()
File: ReplicationId.scala:36
Taint Flags:

```
33 final class ReplicationId(val typeName: String, val entityId: String, val replicaId: ReplicaId) {
34  import ReplicationId._
35  if (typeName.contains(Separator))
36  throw new IllegalArgumentException(
37  s"entityTypeHint [$typeName] contains [$Separator] which is a reserved character")
38
39  if (entityId.contains(Separator))
```

ReplicationId.scala, line 39 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.persistence.typed

ReplicationId.scala, line 39 (Code Correctness: Constructor Invokes Overridable Function) — Low

Sink Details

Sink: FunctionCall: Separator
Enclosing Method: ReplicationId()
File: ReplicationId.scala:39
Taint Flags:

```
36 throw new IllegalArgumentException(
37 s"entityTypeHint [$typeName] contains [$Separator] which is a reserved character")
38
39 if (entityId.contains(Separator))
40 throw new IllegalArgumentException(s"entityId [$entityId] contains [$Separator] which is a reserved character")
41
42 if (replicaId.id.contains(Separator))
```

ReplicationId.scala, line 40 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: Separator
Enclosing Method: ReplicationId()
File: ReplicationId.scala:40
Taint Flags:

```
37 s"entityTypeHint [$typeName] contains [$Separator] which is a reserved character")
38
39 if (entityId.contains(Separator))
40 throw new IllegalArgumentException(s"entityId [$entityId] contains [$Separator] which is a reserved character")
41
42 if (replicaId.id.contains(Separator))
43 throw new IllegalArgumentException(
```

ReplicationId.scala, line 42 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: Separator
Enclosing Method: ReplicationId()

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.persistence.typed

ReplicationId.scala, line 42 (Code Correctness: Constructor Invokes Overridable Function) — Low

File: ReplicationId.scala:42
Taint Flags:

```
39  if (entityId.contains(Separator))
40  throw new IllegalArgumentException(s"entityId [$entityId] contains [$Separator] which is a reserved character")
41
42  if (replicaId.id.contains(Separator))
43  throw new IllegalArgumentException(
44  s"replicaId [${replicaId.id}] contains [$Separator] which is a reserved character")
45
```

ReplicationId.scala, line 43 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: Separator
Enclosing Method: ReplicationId()
File: ReplicationId.scala:43
Taint Flags:

```
40  throw new IllegalArgumentException(s"entityId [$entityId] contains [$Separator] which is a reserved character")
41
42  if (replicaId.id.contains(Separator))
43  throw new IllegalArgumentException(
44  s"replicaId [${replicaId.id}] contains [$Separator] which is a reserved character")
45
46  private val id: String = s"$typeName$Separator$entityId$Separator${replicaId.id}"
```

ReplicationId.scala, line 46 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: Separator
Enclosing Method: ReplicationId()
File: ReplicationId.scala:46
Taint Flags:

```
43  throw new IllegalArgumentException(
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.persistence.typed

ReplicationId.scala, line 46 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
s"replicaId [${replicaId.id}] contains [$Separator] which is a reserved character")

private val id: String = s"$typeName$Separator$entityId$Separator${replicaId.id}"

def persistenceId: PersistenceId = PersistenceId.ofUniqueId(id)

```

ReplicationId.scala, line 46 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: Separator
Enclosing Method: ReplicationId()
File: ReplicationId.scala:46
Taint Flags:

```
throw new IllegalArgumentException(
s"replicaId [${replicaId.id}] contains [$Separator] which is a reserved character")

private val id: String = s"$typeName$Separator$entityId$Separator${replicaId.id}"

def persistenceId: PersistenceId = PersistenceId.ofUniqueId(id)

```

Package: akka.persistence.typed.internal

internal/RequestingRecoveryPermit.scala, line 37 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: onRequestingRecoveryPermit
Enclosing Method: RequestingRecoveryPermit()
File: internal/RequestingRecoveryPermit.scala:37
Taint Flags:

```
with JournalInteractions[C, E, S]
with SnapshotInteractions[C, E, S] {

```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.persistence.typed.internal

internal/RequestingRecoveryPermit.scala, line 37 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
onRequestingRecoveryPermit(setup.context)

def createBehavior(): Behavior[InternalProtocol] = {
// request a permit, as only once we obtain one we can start replaying
```

internal/FastForwardingFilter.scala, line 36 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: out
Enclosing Method: FastForwardingFilter()
File: internal/FastForwardingFilter.scala:36
Taint Flags:

```
val in = Inlet[EventEnvelope]("FastForwardingFilter.in")
val out = Outlet[EventEnvelope]("FastForwardingFilter.out")

override val shape = FlowShape[EventEnvelope, EventEnvelope](in, out)

override def createLogicAndMaterializedValue(
inheritedAttributes: Attributes): (GraphStageLogic, ReplicationStreamControl) = {
```

internal/ReplayingSnapshot.scala, line 50 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: onRecoveryStart
Enclosing Method: ReplayingSnapshot()
File: internal/ReplayingSnapshot.scala:50
Taint Flags:

```

import InternalProtocol._

onRecoveryStart(setup.context)

def createBehavior(receivedPoisonPillInPreviousPhase: Boolean): Behavior[InternalProtocol] = {
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.persistence.typed.internal

internal/ReplayingSnapshot.scala, line 50 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
53 // protect against snapshot stalling forever because of journal overloaded and such
```

internal/FastForwardingFilter.scala, line 36 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: in
Enclosing Method: FastForwardingFilter()
File: internal/FastForwardingFilter.scala:36
Taint Flags:

```
33 val in = Inlet[EventEnvelope]("FastForwardingFilter.in")
34 val out = Outlet[EventEnvelope]("FastForwardingFilter.out")
35
36 override val shape = FlowShape[EventEnvelope, EventEnvelope](in, out)
37
38 override def createLogicAndMaterializedValue(
39 inheritedAttributes: Attributes): (GraphStageLogic, ReplicationStreamControl) = {
```

internal/BehaviorSetup.scala, line 70 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: snapshotStore
Enclosing Method: BehaviorSetup()
File: internal/BehaviorSetup.scala:70
Taint Flags:

```
67 val journal: ClassicActorRef = persistence.journalFor(settings.journalPluginId)
68 val snapshotStore: ClassicActorRef = persistence.snapshotStoreFor(settings.snapshotPluginId)
69
70 val isSnapshotOptional: Boolean =
71 Persistence(context.system.classicSystem).configFor(snapshotStore).getBoolean("snapshot-is-optional")
72
73 if (isSnapshotOptional && (retention match {
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.persistence.typed.internal

internal/BehaviorSetup.scala, line 67 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: persistence
Enclosing Method: BehaviorSetup()
File: internal/BehaviorSetup.scala:67
Taint Flags:

```
64
65  val persistence: Persistence = Persistence(context.system.toClassic)
66
67  val journal: ClassicActorRef = persistence.journalFor(settings.journalPluginId)
68  val snapshotStore: ClassicActorRef = persistence.snapshotStoreFor(settings.snapshotPluginId)
69
70  val isSnapshotOptional: Boolean =
```

internal/BehaviorSetup.scala, line 68 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: persistence
Enclosing Method: BehaviorSetup()
File: internal/BehaviorSetup.scala:68
Taint Flags:

```
65  val persistence: Persistence = Persistence(context.system.toClassic)
66
67  val journal: ClassicActorRef = persistence.journalFor(settings.journalPluginId)
68  val snapshotStore: ClassicActorRef = persistence.snapshotStoreFor(settings.snapshotPluginId)
69
70  val isSnapshotOptional: Boolean =
71  Persistence(context.system.classicSystem).configFor(snapshotStore).getBoolean("snapshot-is-optional")
```

internal/ReplayingEvents.scala, line 85 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.persistence.typed.internal

internal/ReplayingEvents.scala, line 85 (Code Correctness: Constructor Invokes Overridable Function) — Low

Sink Details

Sink: FunctionCall: state
Enclosing Method: ReplayingEvents()
File: internal/ReplayingEvents.scala:85
Taint Flags:

```
82 import InternalProtocol._
83 import ReplayingEvents.ReplayingState
84
85 replayEvents(state.seqNr + 1L, state.toSeqNr)
86 onRecoveryStart(setup.context)
87
88 @InternalStableApi
```

internal/ReplayingEvents.scala, line 85 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: state
Enclosing Method: ReplayingEvents()
File: internal/ReplayingEvents.scala:85
Taint Flags:

```
82 import InternalProtocol._
83 import ReplayingEvents.ReplayingState
84
85 replayEvents(state.seqNr + 1L, state.toSeqNr)
86 onRecoveryStart(setup.context)
87
88 @InternalStableApi
```

internal/VersionVector.scala, line 19 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: emptyVersions
Enclosing Method: VersionVector()

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.persistence.typed.internal

internal/VersionVector.scala, line 19 (Code Correctness: Constructor Invokes Overridable Function) — Low

File: internal/VersionVector.scala:19
Taint Flags:

```
16 private[akka] object VersionVector {
17
18  private val emptyVersions: TreeMap[String, Long] = TreeMap.empty
19  val empty: VersionVector = ManyVersionVector(emptyVersions)
20
21  def apply(): VersionVector = empty
22
```

internal/ReplayingEvents.scala, line 86 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: onRecoveryStart
Enclosing Method: ReplayingEvents()
File: internal/ReplayingEvents.scala:86
Taint Flags:

```
83  import ReplayingEvents.ReplayingState
84
85  replayEvents(state.seqNr + 1L, state.toSeqNr)
86  onRecoveryStart(setup.context)
87
88  @InternalStableApi
89  def onRecoveryStart(@unused context: ActorContext[_]): Unit = ()
```

internal/BehaviorSetup.scala, line 73 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: isSnapshotOptional
Enclosing Method: BehaviorSetup()
File: internal/BehaviorSetup.scala:73
Taint Flags:

```
70  val isSnapshotOptional: Boolean =
```

Code Correctness: Constructor Invokes Overridable Function — Low

Package: akka.persistence.typed.internal

internal/BehaviorSetup.scala, line 73 (Code Correctness: Constructor Invokes Overridable Function) — Low

```
71 Persistence(context.system.classicSystem).configFor(snapshotStore).getBoolean("snapshot-is-optional")
72
73 if (isSnapshotOptional && (retention match {
74 case SnapshotCountRetentionCriteriaImpl(_, _, true) => true
75 case _ => false
76 })) {
```

Package: akka.persistence.typed.state.internal

state/internal/RequestingRecoveryPermit.scala, line 36 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: onRequestingRecoveryPermit
Enclosing Method: RequestingRecoveryPermit()
File: state/internal/RequestingRecoveryPermit.scala:36
Taint Flags:

```
33 extends StashManagement[C, S]
34 with DurableStateStoreInteractions[C, S] {
35
36 onRequestingRecoveryPermit(setup.context)
37
38 def createBehavior(): Behavior[InternalProtocol] = {
39 // request a permit, as only once we obtain one we can start recovery
```

state/internal/Recovering.scala, line 72 (Code Correctness: Constructor Invokes Overridable Function) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: FunctionCall: onRecoveryStart
Enclosing Method: Recovering()
File: state/internal/Recovering.scala:72
Taint Flags:

```
69 import InternalProtocol._
70 import Recovering.RecoveryState
71
```

Code Correctness: Constructor Invokes Overridable Function	Low

Package: akka.persistence.typed.state.internal

state/internal/Recovering.scala, line 72 (Code Correctness: Constructor Invokes Overridable Function)	Low

```
72 onRecoveryStart(setup.context)
73 internalGet(setup.context)
74
75 override def onMessage(msg: InternalProtocol): Behavior[InternalProtocol] = {
```

Code Correctness: Erroneous String Compare (4 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");
  }
```

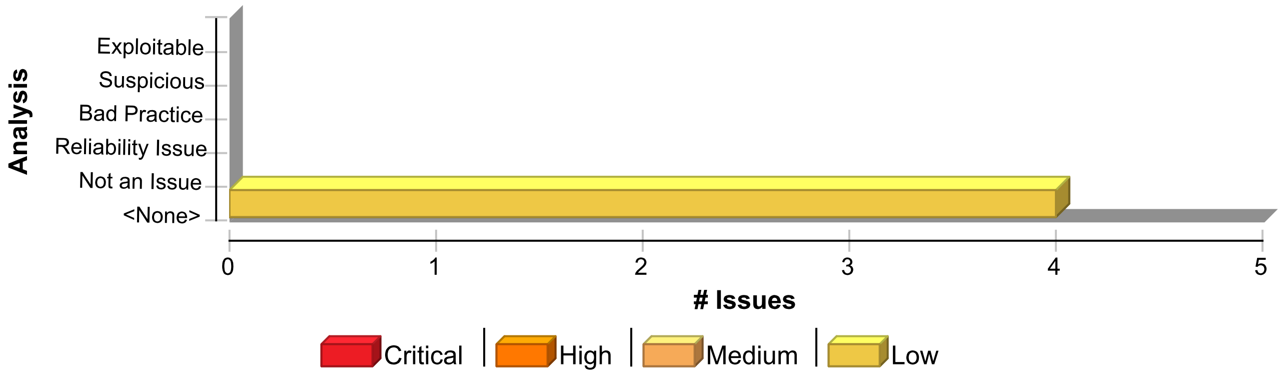
The `==` and `!=` operators will only behave as expected when they are used to compare strings contained in objects that are equal. The most common way for this to occur is for the strings to be interned, whereby the strings are added to a pool of objects maintained by the `String` class. Once a string 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	4	0	0	4
Total	**4**	**0**	**0**	**4**

Code Correctness: Erroneous String Compare — Low

Package: akka.persistence.typed.internal

internal/EventSourcedSettings.scala, line 28 (Code Correctness: Erroneous String Compare) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: apply()
File: internal/EventSourcedSettings.scala:28
Taint Flags:

```
def apply(config: Config, journalPluginId: String, snapshotPluginId: String): EventSourcedSettings = {
val typedConfig = config.getConfig("akka.persistence.typed")

val stashOverflowStrategy = typedConfig.getString("stash-overflow-strategy").toLowerCase match {
case "drop" => StashOverflowStrategy.Drop
case "fail" => StashOverflowStrategy.Fail
case unknown =>
```

internal/EventSourcedSettings.scala, line 28 (Code Correctness: Erroneous String Compare) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: apply()
File: internal/EventSourcedSettings.scala:28
Taint Flags:

```
def apply(config: Config, journalPluginId: String, snapshotPluginId: String): EventSourcedSettings = {
val typedConfig = config.getConfig("akka.persistence.typed")

val stashOverflowStrategy = typedConfig.getString("stash-overflow-strategy").toLowerCase match {
case "drop" => StashOverflowStrategy.Drop
case "fail" => StashOverflowStrategy.Fail
case unknown =>
```

Package: akka.persistence.typed.state.internal

state/internal/DurableStateSettings.scala, line 29 (Code Correctness: Erroneous String Compare) — Low

Issue Details

Kingdom: Code Quality

Code Correctness: Erroneous String Compare — Low

Package: akka.persistence.typed.state.internal

state/internal/DurableStateSettings.scala, line 29 (Code Correctness: Erroneous String Compare) — Low

Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: apply()
File: state/internal/DurableStateSettings.scala:29
Taint Flags:

```
26 def apply(config: Config, durableStateStorePluginId: String): DurableStateSettings = {
27 val typedConfig = config.getConfig("akka.persistence.typed")
28
29 val stashOverflowStrategy = typedConfig.getString("stash-overflow-strategy").toLowerCase match {
30 case "drop" => StashOverflowStrategy.Drop
31 case "fail" => StashOverflowStrategy.Fail
32 case unknown =>
```

state/internal/DurableStateSettings.scala, line 29 (Code Correctness: Erroneous String Compare) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Operation
Enclosing Method: apply()
File: state/internal/DurableStateSettings.scala:29
Taint Flags:

```
26 def apply(config: Config, durableStateStorePluginId: String): DurableStateSettings = {
27 val typedConfig = config.getConfig("akka.persistence.typed")
28
29 val stashOverflowStrategy = typedConfig.getString("stash-overflow-strategy").toLowerCase match {
30 case "drop" => StashOverflowStrategy.Drop
31 case "fail" => StashOverflowStrategy.Fail
32 case unknown =>
```

Code Correctness: Non-Static Inner Class Implements Serializable (23 issues)

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

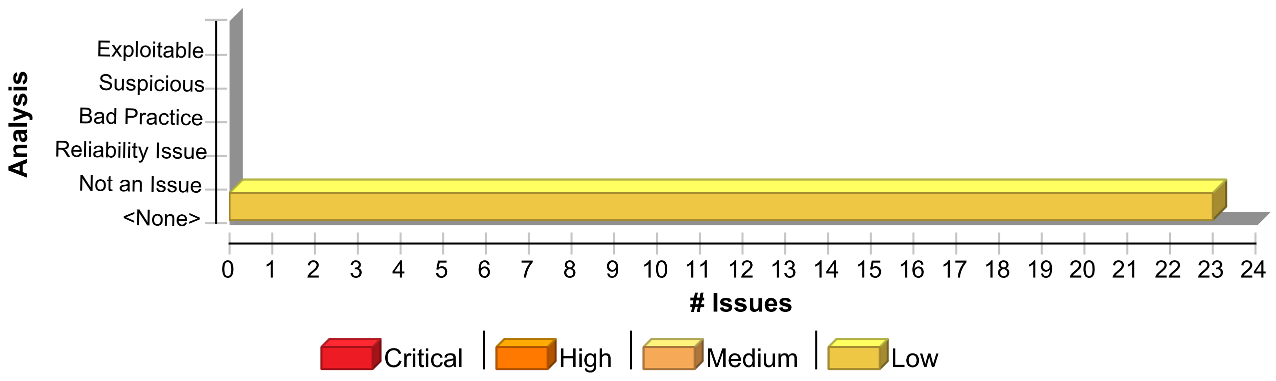
When using inner classes, they should not be serialized, or they should be changed to static-nested classes, since these do not have the drawbacks that non-static inner classes have when serialized. When a nested class is static it inherently has no association with instance variables (including those 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	23	0	0	23
Total	**23**	**0**	**0**	**23**

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

Package: akka.persistence.typed.crdt

crdt/ORSet.scala, line 81 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ORSet$FullStateDeltaOp
File: crdt/ORSet.scala:81
Taint Flags:

```
78 }
79
80 /** INTERNAL API: Used for `clear` but could be used for other cases also */
81 @InternalApi private[akka] final case class FullStateDeltaOp[A](underlying: ORSet[A]) extends AtomicDeltaOp[A] {
82 override def merge(that: DeltaOp): DeltaOp = that match {
83 case _: AtomicDeltaOp[A @unchecked] => DeltaGroup(Vector(this, that))
84 case DeltaGroup(ops) => DeltaGroup(this +: ops)
```

crdt/ORSet.scala, line 91 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ORSet$DeltaGroup

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

Package: akka.persistence.typed.crdt

crdt/ORSet.scala, line 91 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

File: crdt/ORSet.scala:91
Taint Flags:

```
88  /**
89  * INTERNAL API
90  */
91  @InternalApi private[akka] final case class DeltaGroup[A](ops: immutable.IndexedSeq[DeltaOp]) extends DeltaOp {
92  override def merge(that: DeltaOp): DeltaOp = that match {
93  case thatAdd: AddDeltaOp[A @unchecked] =>
94  // merge AddDeltaOp into last AddDeltaOp in the group, if possible
```

crdt/Counter.scala, line 10 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: Counter$Updated
File: crdt/Counter.scala:10
Taint Flags:

```
7 object Counter {
8  val empty: Counter = Counter(0)
9
10  final case class Updated(delta: BigInt) {
11
12  /**
13  * JAVA API
```

crdt/ORSet.scala, line 46 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ORSet$AddDeltaOp
File: crdt/ORSet.scala:46
Taint Flags:

```
43  }
44
45  /** INTERNAL API */
```

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

Package: akka.persistence.typed.crdt

crdt/ORSet.scala, line 46 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
46 @InternalApi private[akka] final case class AddDeltaOp[A](underlying: ORSet[A]) extends AtomicDeltaOp[A] {
47
48 override def merge(that: DeltaOp): DeltaOp = that match {
49 case AddDeltaOp(u) =>
```

crdt/ORSet.scala, line 70 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ORSet$RemoveDeltaOp
File: crdt/ORSet.scala:70
Taint Flags:

```
67 }
68
69 /** INTERNAL API */
70 @InternalApi private[akka] final case class RemoveDeltaOp[A](underlying: ORSet[A]) extends AtomicDeltaOp[A] {
71 if (underlying.size != 1)
72 throw new IllegalArgumentException(s"RemoveDeltaOp should contain one removed element, but was $underlying")
73
```

Package: akka.persistence.typed.delivery

delivery/EventSourcedProducerQueue.scala, line 162 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: EventSourcedProducerQueue$CleanupTick
File: delivery/EventSourcedProducerQueue.scala:162
Taint Flags:

```
159 s"Settings($restartMaxBackoff,$snapshotEvery,$keepNSnapshots,$deleteEvents,$cleanupUnusedAfter,$journalPluginId,
$snapshotPluginId)"
160 }
161
162 private case class CleanupTick[A]() extends DurableProducerQueue.Command[A]
163
164 def apply[A](persistenceId: PersistenceId): Behavior[DurableProducerQueue.Command[A]] = {
```

Code Correctness: Non-Static Inner Class Implements Serializable	Low

Package: akka.persistence.typed.delivery

delivery/EventSourcedProducerQueue.scala, line 162 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

```
Behaviors.setup { context =>
```

Package: akka.persistence.typed.internal

internal/EventSourcedBehaviorImpl.scala, line 64 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: EventSourcedBehaviorImpl$WriterIdentity
File: internal/EventSourcedBehaviorImpl.scala:64
Taint Flags:

```
WriterIdentity(instanceId, writerUuid)
}
}
final case class WriterIdentity(instanceId: Int, writerUuid: String)

/**
* Used by EventSourcedBehaviorTestKit to retrieve the `persistenceId`.
```

internal/ExternalInteractions.scala, line 29 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: JournalInteractions$EventToPersist
File: internal/ExternalInteractions.scala:29
Taint Flags:

```

type EventOrTaggedOrReplicated = Any // `Any` since can be `E` or `Tagged` or a `ReplicatedEvent`

final case class EventToPersist(
adaptedEvent: EventOrTaggedOrReplicated,
manifest: String,
metadata: Option[ReplicatedEventMetadata])
```

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

Package: akka.persistence.typed.internal

internal/EventSourcedBehaviorImpl.scala, line 80 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: EventSourcedBehaviorImpl$GetStateReply
File: internal/EventSourcedBehaviorImpl.scala:80
Taint Flags:

```
/**
 * Used to send a state being `null` as an Actor message
 */
final case class GetStateReply[State](currentState: State)

/**
 * Used to start the replication stream at the correct sequence number
```

internal/EventSourcedBehaviorImpl.scala, line 85 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: EventSourcedBehaviorImpl$GetSeenSequenceNr
File: internal/EventSourcedBehaviorImpl.scala:85
Taint Flags:

```
/**
 * Used to start the replication stream at the correct sequence number
 */
final case class GetSeenSequenceNr(replica: ReplicaId, replyTo: ActorRef[Long]) extends InternalProtocol

}

```

internal/EventSourcedBehaviorImpl.scala, line 69 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

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

Package: akka.persistence.typed.internal

internal/EventSourcedBehaviorImpl.scala, line 69 (Code Correctness: Non-Static Inner Class Implements Serializable) | Low

Sink: Class: EventSourcedBehaviorImpl$GetPersistenceId
File: internal/EventSourcedBehaviorImpl.scala:69
Taint Flags:

```
66 /**
67 * Used by EventSourcedBehaviorTestKit to retrieve the `persistenceId`.
68 */
69 final case class GetPersistenceId(replyTo: ActorRef[PersistenceId]) extends Signal
70
71 /**
72 * Used by EventSourcedBehaviorTestKit to retrieve the state.
```

internal/Running.scala, line 90 (Code Correctness: Non-Static Inner Class Implements Serializable) | Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: Running$RunningState
File: internal/Running.scala:90
Taint Flags:

```
87 def currentSequenceNumber: Long
88 }
89
90 final case class RunningState[State](
91 seqNr: Long,
92 state: State,
93 receivedPoisonPill: Boolean,
```

internal/ReplayingEvents.scala, line 52 (Code Correctness: Non-Static Inner Class Implements Serializable) | Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: ReplayingEvents$ReplayingState
File: internal/ReplayingEvents.scala:52
Taint Flags:

```
49 private[akka] object ReplayingEvents {
50
```

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

Package: akka.persistence.typed.internal

internal/ReplayingEvents.scala, line 52 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
@InternalApi
private[akka] final case class ReplayingState[State](
seqNr: Long,
state: State,
eventSeenInInterval: Boolean,
```

internal/EventSourcedBehaviorImpl.scala, line 75 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: EventSourcedBehaviorImpl$GetState
File: internal/EventSourcedBehaviorImpl.scala:75
Taint Flags:

```
* Used by EventSourcedBehaviorTestKit to retrieve the state.
* Can't be a Signal because those are not stashed.
*/
final case class GetState[State](replyTo: ActorRef[GetStateReply[State]]) extends InternalProtocol

/**
* Used to send a state being `null` as an Actor message
```

Package: akka.persistence.typed.javadsl

javadsl/EventHandler.scala, line 184 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: EventHandlerBuilderByState$EventHandlerCase
File: javadsl/EventHandler.scala:184
Taint Flags:

```
/**
* INTERNAL API
*/
@InternalApi private final case class EventHandlerCase[State, Event](
statePredicate: State => Boolean,
eventPredicate: Event => Boolean,
```

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

Package: akka.persistence.typed.javadsl

javadsl/EventHandler.scala, line 184 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

```
handler: BiFunction[State, Event, State])
```

javadsl/CommandHandlerWithReply.scala, line 195 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: CommandHandlerWithReplyBuilderByState$CommandHandlerCase
File: javadsl/CommandHandlerWithReply.scala:195
Taint Flags:

```
/**
* INTERNAL API
*/
@InternalApi private final case class CommandHandlerCase[Command, Event, State](
commandPredicate: Command => Boolean,
statePredicate: State => Boolean,
handler: BiFunction[State, Command, ReplyEffect[Event, State]])
```

javadsl/CommandHandler.scala, line 185 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: CommandHandlerBuilderByState$CommandHandlerCase
File: javadsl/CommandHandler.scala:185
Taint Flags:

```
/**
* INTERNAL API
*/
@InternalApi private final case class CommandHandlerCase[Command, Event, State](
commandPredicate: Command => Boolean,
statePredicate: State => Boolean,
handler: BiFunction[State, Command, Effect[Event, State]])
```

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

Package: akka.persistence.typed.state.internal

state/internal/Recovering.scala, line 52 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: Recovering$RecoveryState
File: state/internal/Recovering.scala:52
Taint Flags:

```
}

@InternalApi
private[akka] final case class RecoveryState[State](
revision: Long,
state: State,
receivedPoisonPill: Boolean,
```

state/internal/DurableStateBehaviorImpl.scala, line 34 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: DurableStateBehaviorImpl$GetPersistenceId
File: state/internal/DurableStateBehaviorImpl.scala:34
Taint Flags:

```
/**
* Used by DurableStateBehaviorTestKit to retrieve the `persistenceId`.
*/
final case class GetPersistenceId(replyTo: ActorRef[PersistenceId]) extends Signal

/**
* Used by DurableStateBehaviorTestKit to retrieve the state.
```

state/internal/Running.scala, line 48 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

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

Package: akka.persistence.typed.state.internal

state/internal/Running.scala, line 48 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Sink: Class: Running$RunningState
File: state/internal/Running.scala:48
Taint Flags:

```
def currentRevision: Long
}

final case class RunningState[State](revision: Long, state: State, receivedPoisonPill: Boolean) {

def nextRevision(): RunningState[State] =
copy(revision = revision + 1)
```

state/internal/DurableStateBehaviorImpl.scala, line 40 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: DurableStateBehaviorImpl$GetState
File: state/internal/DurableStateBehaviorImpl.scala:40
Taint Flags:

```
* Used by DurableStateBehaviorTestKit to retrieve the state.
* Can't be a Signal because those are not stashed.
*/
final case class GetState[State](replyTo: ActorRef[State]) extends InternalProtocol

}

```

Package: akka.persistence.typed.state.javadsl

state/javadsl/CommandHandler.scala, line 186 (Code Correctness: Non-Static Inner Class Implements Serializable) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: CommandHandlerBuilderByState$CommandHandlerCase
File: state/javadsl/CommandHandler.scala:186
Taint Flags:

Code Correctness: Non-Static Inner Class Implements Serializable	Low

Package: akka.persistence.typed.state.javadsl

state/javadsl/CommandHandler.scala, line 186 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

```
/**
* INTERNAL API
*/
@InternalApi private final case class CommandHandlerCase[Command, State](
commandPredicate: Command => Boolean,
statePredicate: State => Boolean,
handler: BiFunction[State, Command, Effect[State]])
```

state/javadsl/CommandHandlerWithReply.scala, line 196 (Code Correctness: Non-Static Inner Class Implements Serializable)	Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: Class: CommandHandlerWithReplyBuilderByState$CommandHandlerCase
File: state/javadsl/CommandHandlerWithReply.scala:196
Taint Flags:

```
/**
* INTERNAL API
*/
@InternalApi private final case class CommandHandlerCase[Command, State](
commandPredicate: Command => Boolean,
statePredicate: State => Boolean,
handler: BiFunction[State, Command, ReplyEffect[State]])
```

Dead Code: Expression is Always false (1 issue)

Abstract

This expression will always evaluate to `false`.

Explanation

This expression will always evaluate to `false`; the program could be rewritten in a simpler form. The nearby code may be present for debugging purposes, or it may not have been maintained along with the rest of the program. The expression may also be indicative of a bug earlier in the method. **Example 1:** The following method never sets the variable `secondCall` after initializing it to `false`. (The variable `firstCall` is mistakenly used twice.) The result is that the expression `firstCall && secondCall` will always evaluate to `false`, so `setUpDualCall()` will never be invoked.

```
public void setUpCalls() {
  boolean firstCall = false;
  boolean secondCall = false;

  if (fCall > 0) {
    setUpFCall();
    firstCall = true;
  }
  if (sCall > 0) {
    setUpSCall();
    firstCall = true;
  }

  if (firstCall && secondCall) {
    setUpDualCall();
  }
}
```

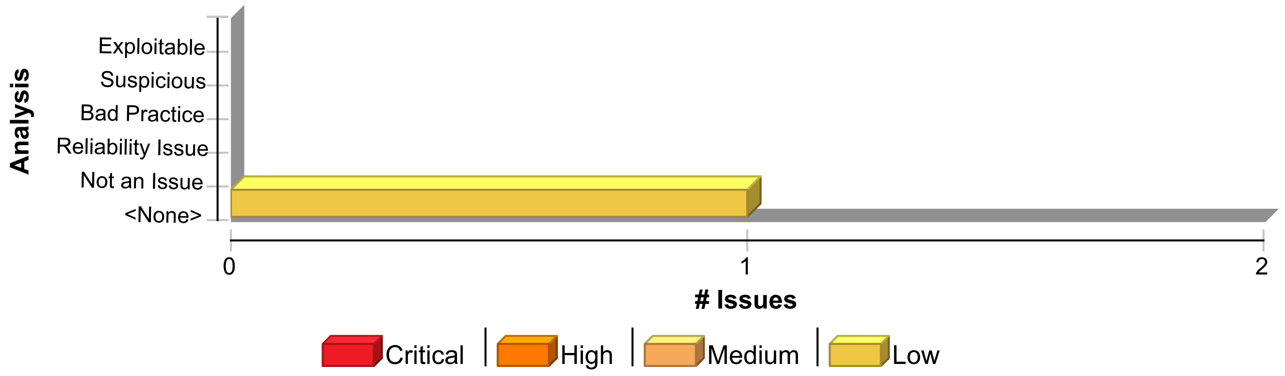
Example 2: The following method never sets the variable `firstCall` to `true`. (The variable `firstCall` is mistakenly set to `false` after the first conditional statement.) The result is that the first part of the expression `firstCall && secondCall` will always evaluate to `false`.

```
public void setUpCalls() {
  boolean firstCall = false;
  boolean secondCall = false;

  if (fCall > 0) {
    setUpFCall();
    firstCall = false;
  }
  if (sCall > 0) {
    setUpSCall();
    secondCall = true;
  }

  if (firstCall && secondCall) {
    setUpForCall();
  }
}
```

Recommendation

In general, you should repair or remove unused code. It causes additional complexity and maintenance burden without contributing to the functionality of the program.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Dead Code: Expression is Always false	1	0	0	1
Total	**1**	**0**	**0**	**1**

Dead Code: Expression is Always false — Low

Package: akka.persistence.typed.state.internal

state/internal/DurableStateBehaviorImpl.scala, line 157 (Dead Code: Expression is Always false) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: withDurableStateStorePluginId()
File: state/internal/DurableStateBehaviorImpl.scala:157
Taint Flags:

```
154
155 override def withDurableStateStorePluginId(id: String): DurableStateBehavior[Command, State] = {
156 require(id != null, "DurableStateBehavior plugin id must not be null; use empty string for 'default' state store")
157 copy(durableStateStorePluginId = if (id != "") Some(id) else None)
158 }
159
160 override def withTag(tag: String): DurableStateBehavior[Command, State] =
```

Dead Code: Expression is Always true (1 issue)

Abstract

This expression will always evaluate to `true`.

Explanation

This expression will always evaluate to `true`; the program could be rewritten in a simpler form. The nearby code may be present for debugging purposes, or it may not have been maintained along with the rest of the program. The expression may also be indicative of a bug earlier in the method. **Example 1:** The following method never sets the variable `secondCall` after initializing it to `true`. (The variable `firstCall` is mistakenly used twice.) The result is that the expression `firstCall || secondCall` will always evaluate to `true`, so `setUpForCall()` will always be invoked.

```
public void setUpCalls() {
  boolean firstCall = true;
  boolean secondCall = true;

  if (fCall < 0) {
    cancelFCall();
    firstCall = false;
  }
  if (sCall < 0) {
    cancelSCall();
    firstCall = false;
  }

  if (firstCall || secondCall) {
    setUpForCall();
  }
}
```

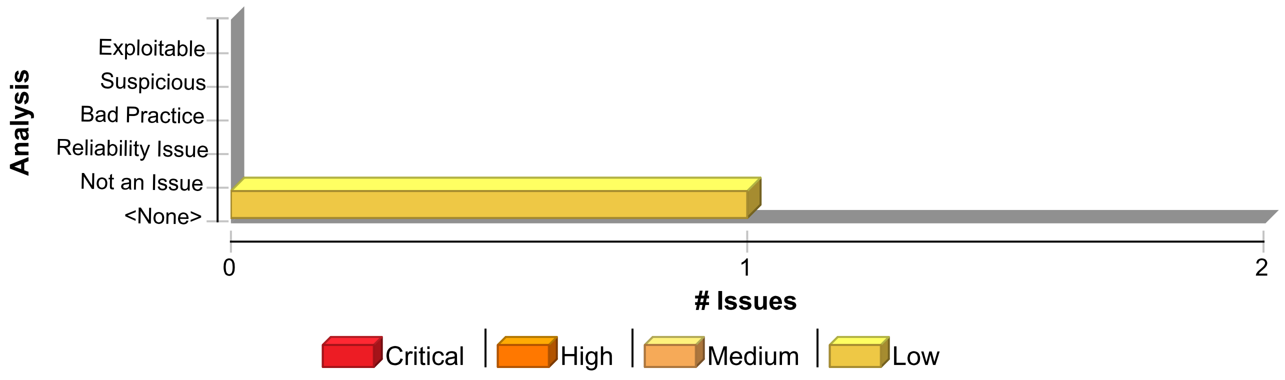
Example 2: The following method tries to check the variables `firstCall` and `secondCall`. (The variable `firstCall` is mistakenly set to `true` instead of being checked.) The result is that the first part of the expression `firstCall = true && secondCall == true` will always evaluate to `true`.

```
public void setUpCalls() {
  boolean firstCall = false;
  boolean secondCall = false;

  if (fCall > 0) {
    setUpFCall();
    firstCall = true;
  }
  if (sCall > 0) {
    setUpSCall();
    secondCall = true;
  }

  if (firstCall = true && secondCall == true) {
    setUpDualCall();
  }
}
```

Recommendation

In general, you should repair or remove unused code. It causes additional complexity and maintenance burden without contributing to the functionality of the program.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Dead Code: Expression is Always true	1	0	0	1
Total	**1**	**0**	**0**	**1**

Dead Code: Expression is Always true — Low

Package: akka.persistence.typed.internal

internal/ReplayingSnapshot.scala, line 151 (Dead Code: Expression is Always true) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Structural)

Sink Details

Sink: IfStatement
Enclosing Method: loadSnapshotResult()
File: internal/ReplayingSnapshot.scala:151
Taint Flags:

```
148 def loadSnapshotResult(snapshot: Option[SelectedSnapshot], toSnr: Long): Behavior[InternalProtocol] = {
149 var state: S = setup.emptyState
150
151 val (seqNr: Long, seenPerReplica, version) = snapshot match {
152 case Some(SelectedSnapshot(metadata, snapshot)) =>
153 state = setup.snapshotAdapter.fromJournal(snapshot)
154 setup.internalLogger.debug("Loaded snapshot with metadata [{}]", metadata)
```

Redundant Null Check (3 issues)

Abstract

The program can dereference a null-pointer, thereby causing a null-pointer exception.

Explanation

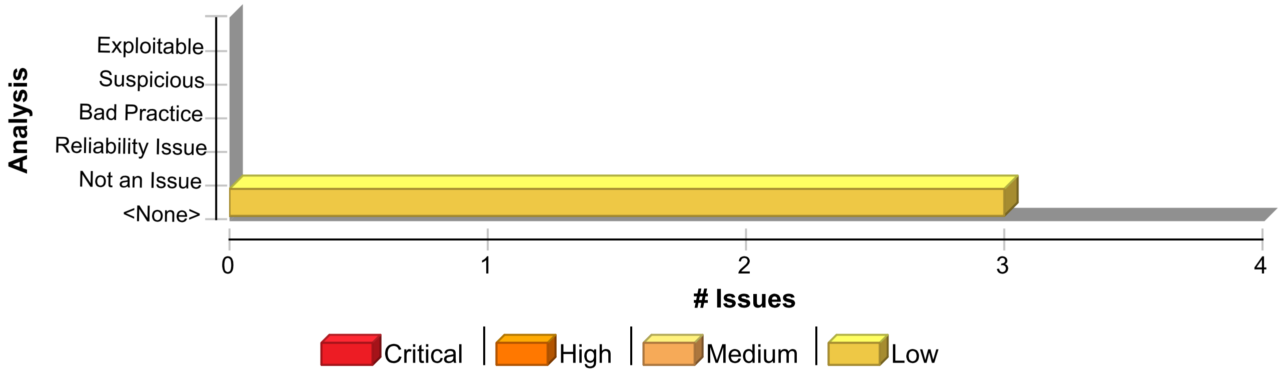
Null-pointer exceptions usually occur when one or more of the programmer's assumptions is violated. Specifically, dereference-after-check errors occur when a program makes an explicit check for `null`, but proceeds to dereference the object when it is known to be `null`. Errors of this type are often the result of a typo or programmer oversight. Most null-pointer issues result in general software reliability problems, but if attackers can intentionally cause the program to dereference a null-pointer, they can use the resulting exception to mount a denial of service attack or to cause the application to reveal debugging information that will be valuable in planning subsequent attacks. **Example 1:** In the following code, the programmer confirms that the variable `foo` is `null` and subsequently dereferences it erroneously. If `foo` is `null` when it is checked in the `if` statement, then a `null` dereference will occur, thereby causing a null-pointer exception.

```
if (foo == null) {
    foo.setBar(val);
    ...
}
```

Recommendation

Implement careful checks before dereferencing objects that might be `null`. When possible, abstract `null` checks into wrappers around code that manipulates resources to ensure that they are applied in all cases and to minimize the places where mistakes can occur.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
Redundant Null Check	3	0	0	3
Total	**3**	**0**	**0**	**3**

Redundant Null Check — Low

Package: akka.persistence.typed.internal

internal/EventSourcedBehaviorImpl.scala, line 251 (Redundant Null Check) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Control Flow)

Sink Details

Sink: Dereferenced : id
Enclosing Method: withSnapshotPluginId()
File: internal/EventSourcedBehaviorImpl.scala:251
Taint Flags:

```
248
249 override def withSnapshotPluginId(id: String): EventSourcedBehavior[Command, Event, State] = {
250 require(id != null, "snapshot plugin id must not be null; use empty string for 'default' snapshot store")
251 copy(snapshotPluginId = if (id != "") Some(id) else None)
252 }
253
254 override def withSnapshotSelectionCriteria(
```

internal/EventSourcedBehaviorImpl.scala, line 246 (Redundant Null Check) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Control Flow)

Sink Details

Sink: Dereferenced : id
Enclosing Method: withJournalPluginId()
File: internal/EventSourcedBehaviorImpl.scala:246
Taint Flags:

```
243
244 override def withJournalPluginId(id: String): EventSourcedBehavior[Command, Event, State] = {
245 require(id != null, "journal plugin id must not be null; use empty string for 'default' journal")
246 copy(journalPluginId = if (id != "") Some(id) else None)
247 }
248
249 override def withSnapshotPluginId(id: String): EventSourcedBehavior[Command, Event, State] = {
```

Package: akka.persistence.typed.state.internal

state/internal/DurableStateBehaviorImpl.scala, line 157 (Redundant Null Check) — Low

Issue Details

Kingdom: Code Quality
Scan Engine: SCA (Control Flow)

Sink Details

Redundant Null Check — Low

Package: akka.persistence.typed.state.internal

state/internal/DurableStateBehaviorImpl.scala, line 157 (Redundant Null Check) — Low

Sink: Dereferenced : id
Enclosing Method: withDurableStateStorePluginId()
File: state/internal/DurableStateBehaviorImpl.scala:157
Taint Flags:

```
154
155  override def withDurableStateStorePluginId(id: String): DurableStateBehavior[Command, State] = {
156  require(id != null, "DurableStateBehavior plugin id must not be null; use empty string for 'default' state store")
157  copy(durableStateStorePluginId = if (id != "") Some(id) else None)
158  }
159
160  override def withTag(tag: String): DurableStateBehavior[Command, State] =
```

System Information Leak: Internal (4 issues)

Abstract

Revealing system data or debugging information helps an adversary learn about the system and form a plan of attack.

Explanation

An internal information leak occurs when system data or debug information is sent to a local file, console, or screen via printing or logging. **Example 1:** The following code writes an exception to the standard error stream:

```
try {
    ...
} catch (Exception e) {
    e.printStackTrace();
}
```

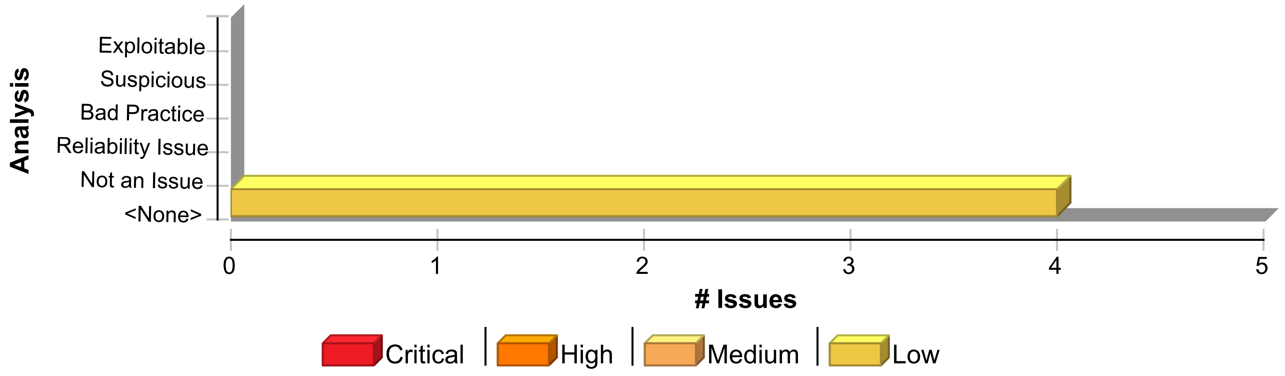
Depending upon the system configuration, this information can be dumped to a console, written to a log file, or exposed to a user. In some cases, the error message provides the attacker with the precise type of attack to which the system is vulnerable. For example, a database error message can reveal that the application is vulnerable to a SQL injection attack. Other error messages can reveal more oblique clues about the system. In `Example 1`, the leaked information could imply information about the type of operating system, the applications installed on the system, and the amount of care that the administrators have put into configuring the program. Information leaks are also a concern in a mobile computing environment. **Example 2:** The following code logs the stack trace of a caught exception on the Android platform.

```
...
try {
  ...
} catch (Exception e) {
    Log.e(TAG, Log.getStackTraceString(e));
}
...
```

Recommendation

Write error messages with security in mind. In production environments, turn off detailed error information in favor of brief messages. Restrict the generation and storage of detailed output that can help administrators and programmers diagnose problems. Debug traces can sometimes appear in non-obvious places (embedded in comments in the HTML for an error page, for example). Even brief error messages that do not reveal stack traces or database dumps can potentially aid an attacker. For example, an "Access Denied" message can reveal that a file or user exists on the system.

Issue Summary



Engine Breakdown

	SCA	WebInspect	SecurityScope	Total
System Information Leak: Internal	4	0	0	4
Total	**4**	**0**	**0**	**4**

System Information Leak: Internal — Low

Package: akka.persistence.typed.internal

internal/ExternalInteractions.scala, line 149 (System Information Leak: Internal) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Data Flow)

Source Details

Source: java.lang.Throwable.getMessage()
From: akka.persistence.typed.internal.ReplayingSnapshot.onRecoveryFailure
File: internal/ReplayingSnapshot.scala:101

```
98  setup.onSignal(setup.emptyState, RecoveryFailed(cause), catchAndLog = true)
99  setup.cancelRecoveryTimer()
100
101  tryReturnRecoveryPermit("on snapshot recovery failure: " + cause.getMessage)
102
103  if (setup.internalLogger.isDebugEnabled)
104  setup.internalLogger.debug("Recovery failure for persistenceId [{}]", setup.persistenceId)
```

Sink Details

Sink: org.slf4j.Logger.debug()
Enclosing Method: tryReturnRecoveryPermit()
File: internal/ExternalInteractions.scala:149
Taint Flags: EXCEPTIONINFO, SYSTEMINFO

```
146  /** Mutates setup, by setting the `holdingRecoveryPermit` to false */
147  protected def tryReturnRecoveryPermit(reason: String): Unit = {
148  if (setup.holdingRecoveryPermit) {
149  setup.internalLogger.debug("Returning recovery permit, reason: {}", reason)
150  setup.persistence.recoveryPermitter.tell(RecoveryPermitter.ReturnRecoveryPermit, setup.selfClassic)
151  setup.holdingRecoveryPermit = false
152  } // else, no need to return the permit
```

internal/EventSourcedBehaviorImpl.scala, line 147 (System Information Leak: Internal) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Data Flow)

Source Details

System Information Leak: Internal — Low

Package: akka.persistence.typed.internal

internal/EventSourcedBehaviorImpl.scala, line 147 (System Information Leak: Internal) — Low

Source: java.lang.Throwable.getMessage()
From: akka.persistence.typed.internal.EventSourcedBehaviorImpl$$anonfun$1.applyOrEl
se
File: internal/EventSourcedBehaviorImpl.scala:147

```
144 internalLogger().debug("Save snapshot successful, snapshot metadata [{}].", meta)
145 case (_, SnapshotFailed(meta, failure)) =>
146 internalLogger()
147 .error(s"Save snapshot failed, snapshot metadata [$meta] due to: ${failure.getMessage}", failure)
148 case (_, DeleteSnapshotsCompleted(DeletionTarget.Individual(meta))) =>
149 internalLogger().debug("Persistent snapshot [{}] deleted successfully.", meta)
150 case (_, DeleteSnapshotsCompleted(DeletionTarget.Criteria(criteria))) =>
```

Sink Details

Sink: org.slf4j.Logger.error()
Enclosing Method: applyOrElse()
File: internal/EventSourcedBehaviorImpl.scala:147
Taint Flags: EXCEPTIONINFO, SYSTEMINFO

```
144 internalLogger().debug("Save snapshot successful, snapshot metadata [{}].", meta)
145 case (_, SnapshotFailed(meta, failure)) =>
146 internalLogger()
147 .error(s"Save snapshot failed, snapshot metadata [$meta] due to: ${failure.getMessage}", failure)
148 case (_, DeleteSnapshotsCompleted(DeletionTarget.Individual(meta))) =>
149 internalLogger().debug("Persistent snapshot [{}] deleted successfully.", meta)
150 case (_, DeleteSnapshotsCompleted(DeletionTarget.Criteria(criteria))) =>
```

internal/ExternalInteractions.scala, line 149 (System Information Leak: Internal) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Data Flow)

Source Details

Source: java.lang.Throwable.getMessage()
From: akka.persistence.typed.internal.ReplayingEvents.onRecoveryFailure
File: internal/ReplayingEvents.scala:249

```
246 onRecoveryFailed(setup.context, cause, event)
247 setup.onSignal(state.state, RecoveryFailed(cause), catchAndLog = true)
248 setup.cancelRecoveryTimer()
249 tryReturnRecoveryPermit("on replay failure: " + cause.getMessage)
250 if (setup.internalLogger.isDebugEnabled) {
251 setup.internalLogger.debug2(
252 "Recovery failure for persistenceId [{}] after {}",
```

System Information Leak: Internal — Low

Package: akka.persistence.typed.internal

internal/ExternalInteractions.scala, line 149 (System Information Leak: Internal) — Low

Sink Details

Sink: org.slf4j.Logger.debug()
Enclosing Method: tryReturnRecoveryPermit()
File: internal/ExternalInteractions.scala:149
Taint Flags: EXCEPTIONINFO, SYSTEMINFO

```
146 /** Mutates setup, by setting the `holdingRecoveryPermit` to false */
147 protected def tryReturnRecoveryPermit(reason: String): Unit = {
148 if (setup.holdingRecoveryPermit) {
149 setup.internalLogger.debug("Returning recovery permit, reason: {}", reason)
150 setup.persistence.recoveryPermitter.tell(RecoveryPermitter.ReturnRecoveryPermit, setup.selfClassic)
151 setup.holdingRecoveryPermit = false
152 } // else, no need to return the permit
```

Package: akka.persistence.typed.state.internal

state/internal/DurableStateStoreInteractions.scala, line 81 (System Information Leak: Internal) — Low

Issue Details

Kingdom: Encapsulation
Scan Engine: SCA (Data Flow)

Source Details

Source: java.lang.Throwable.getMessage()
From: akka.persistence.typed.state.internal.Recovering.onRecoveryFailure
File: state/internal/Recovering.scala:116

```
113 setup.onSignal(setup.emptyState, RecoveryFailed(cause), catchAndLog = true)
114 setup.cancelRecoveryTimer()
115
116 tryReturnRecoveryPermit("on recovery failure: " + cause.getMessage)
117
118 if (setup.internalLogger.isDebugEnabled)
119 setup.internalLogger.debug("Recovery failure for persistenceId [{}]", setup.persistenceId)
```

Sink Details

Sink: org.slf4j.Logger.debug()
Enclosing Method: tryReturnRecoveryPermit()
File: state/internal/DurableStateStoreInteractions.scala:81
Taint Flags: EXCEPTIONINFO, SYSTEMINFO

```
78 /** Mutates setup, by setting the `holdingRecoveryPermit` to false */
79 protected def tryReturnRecoveryPermit(reason: String): Unit = {
80 if (setup.holdingRecoveryPermit) {
```

System Information Leak: Internal	Low
Package: akka.persistence.typed.state.internal	
state/internal/DurableStateStoreInteractions.scala, line 81 (System Information Leak: Internal)	**Low**

```
setup.internalLogger.debug("Returning recovery permit, reason: {}", reason)
setup.persistence.recoveryPermitter.tell(RecoveryPermitter.ReturnRecoveryPermit, setup.selfClassic)
setup.holdingRecoveryPermit = false
} // else, no need to return the permit
```